IBM® DB2 Universal Database™

Common Criteria Certification: Administration and User Documentation

Version 8.2 Revision 05

SC09-7981-00

IBM® DB2 Universal Database™

Common Criteria Certification: Administration and User Documentation

Version 8.2 Revision 05

SC09-7981-00

Before using this information and the product it supports, be sure to read the general information under *Notices*.

This document contains proprietary information of IBM. It is provided under a license agreement and is protected by copyright law. The information contained in this publication does not include any product warranties, and any statements provided in this manual should not be interpreted as such.

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.

© Copyright International Business Machines Corporation 1993-2004. All rights reserved.
US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

© Copyright IBM Corp. 1993-2004

Common Criteria certification of DB2 Universal Database products

For Version 8.2, DB2 Universal Database (DB2 UDB) products are certified according to the Common Criteria evaluation assurance level 4 (EAL4), augmented with Flaw remediation ALC_FLR.1. The following products are certified on the following operating systems:

Table 1. Certified DB2 Universal Database configurations

	Windows® 2000	**Linux SuSE Enterprise Server V8**	**AIX® 5.2**	**Solaris Operating Environment, 8**
Enterprise Server Edition **Note:** Single-partition environment only.	Yes (32-bit only)	Yes (32-bit only)	Yes (64-bit only)	Yes (64-bit only)
Workgroup Server Edition	Yes (32-bit only)	Yes (32-bit only)	Yes (64-bit only)	Yes (64-bit only)
Personal Edition	Yes (32-bit only)	Yes (32-bit only)	N/A	N/A
Express Edition	Yes (32-bit only)	Yes (32-bit only)	N/A	N/A

Notes:

1. DB2 UDB configurations on the Linux SuSE environment are Common Criteria certified on Intel-based hardware only.
2. In a Common Criteria certified DB2 UDB environment, DB2 UDB clients are supported on the following operating systems:
   - Windows 2000
   - Linux SuSE Enterprise Server V8
   - AIX 5.2
   - Solaris Operating Environment, 8

Only 32-bit clients are supported.

For more information about Common Criteria, see the Common Criteria web site at: http://niap.nist.gov/cc-scheme/.

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

- *DB2 Universal Database Common Criteria Certification: Installing DB2 Universal Database Enterprise Server Edition and DB2 Universal Database Workgroup Server Edition*
- *DB2 Universal Database Common Criteria Certification: Installing DB2 Universal Database Personal Edition*
- *DB2 Universal Database Common Criteria Certification: Installing DB2 Universal Database Express Edition*
- *DB2 Universal Database Common Criteria Certification: Administration and User Documentation*

© Copyright IBM Corp. 1993-2004

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 Universal Database interfaces that are used in the Common Criteria evaluation of DB2 Universal Database are as follows:

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

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

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

NOT FENCED routines are not supported.

© Copyright IBM Corp. 1993-2004

About This Book

This book is intended for use by assessors validating that DB2 Universal Database (DB2 UDB) conforms to the Common Critera EAL4 specification. It is also intended for those who want to set up a DB2 UDB environment that conforms to the characteristics of the evaluated environment.

This book is divided into three parts. Part 1, "Administration," on page 1, provides information that is intended for use by database administrators, and includes information on:

- The DB2 UDB process model
- The DB2 UDB security model, and the facilities available to set up and maintain security
- How to set up the DB2 UDB environment so that it conforms to the requirements of the Common Criteria EAL4 specification
- How to audit activity in the environment

Part 1, "Administration," on page 1 also provides background information that you should be familiar with before setting up the DB2 UDB environment.

Part 2, "User Information," on page 833, describes the security-related considerations that are applicable to users of the DB2 UDB environment, including the type of authorization that the administrator must give to a user before that user can work with DB2 utilities. This part of the book also lists the DB2 UDB commands and SQL statements that are appropriate for DB2 UDB users. In addition, the security-related considerations for writing applications that interact with DB2 UDB are also provided.

Part 3, "Security Plug-Ins," on page 1019 provides information on security plug-ins. Note that only the default IBM-supplied operating-system based authentication and group plug-ins are supported in Common Criteria compliant environments.

Note: This book does not provide information on how to install DB2 UDB. For installation information, see the following books:

- *DB2 Universal Database Common Criteria Certification: Installing DB2 Universal Database Enterprise Server Edition and DB2 Universal Database Workgroup Server Edition*
- *DB2 Universal Database Common Criteria Certification: Installing DB2 Universal Database Personal Edition*
- *DB2 Universal Database Common Criteria Certification: Installing DB2 Universal Database Express Edition*

Some topics in this book may link to topics that are not in any of the books listed above. Topics that are referenced outside of the Common Criteria certification documentation are for informational purposes only, and are not required for either installing or configuring a Common Criteria compliant environment.

© Copyright IBM Corp. 1993-2004

Part 1. Administration

The information in this section is provided for use by system administrators who are not careless, willfully negligent, or hostile, and who will follow and abide by the instructions provided by the administrator documentation.

© Copyright IBM Corp. 1993-2004

Chapter 1. Process Overview

DB2 architecture and process overview

General information about DB2® architecture and processes can help you understand detailed information provided for specific topics.

The following figure shows a general overview of the architecture and processes for DB2 UDB.

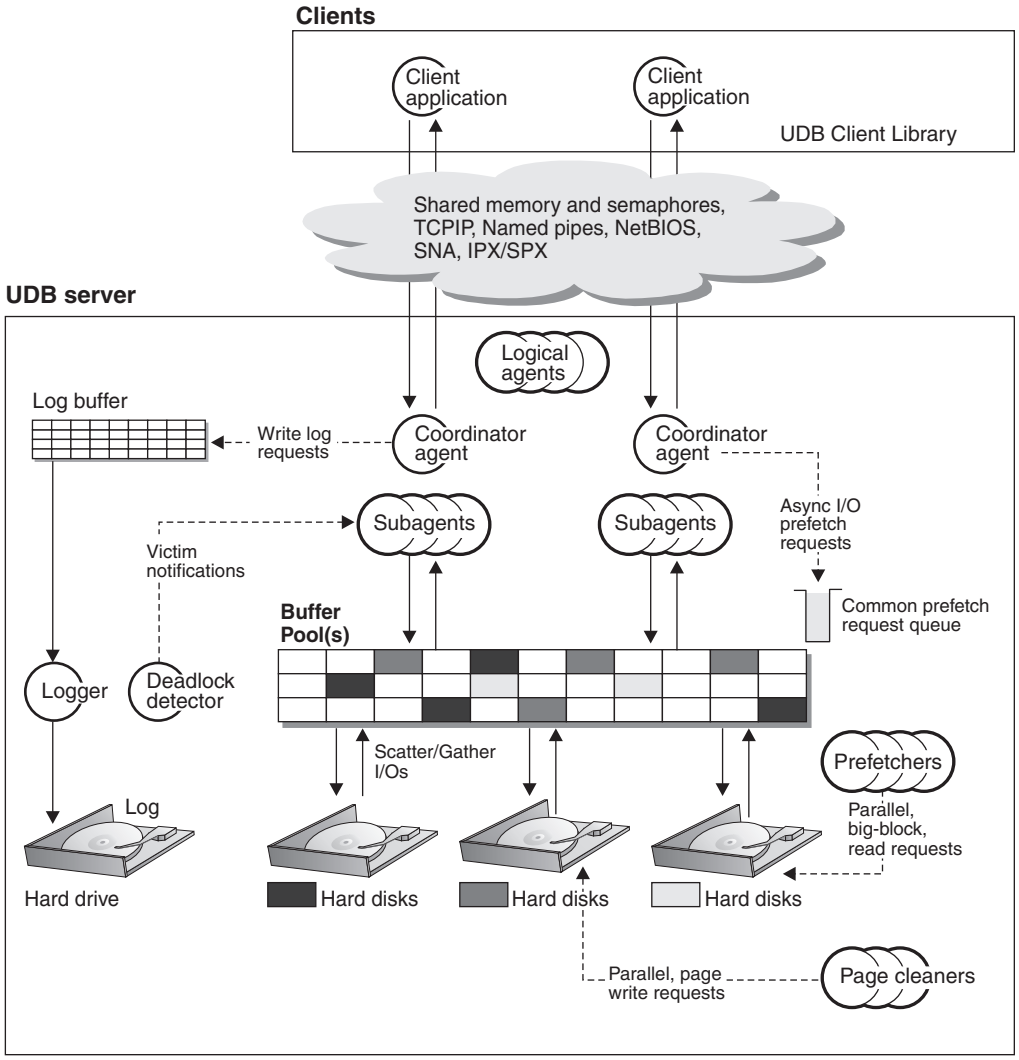


Figure 1. Architecture and Processes Overview

On the client side, either local or remote applications, or both, are linked with the DB2 Universal Database™ client library. Local clients communicate using shared memory and semaphores; remote clients use a protocol such as Named Pipes (NPIPE), TCP/IP, NetBIOS, or SNA.

© Copyright IBM Corp. 1993-2004

On the server side, activity is controlled by engine dispatchable units (EDUs). In all figures in this section, EDUs are shown as circles or groups of circles. EDUs are implemented as threads in a single process on Windows®-based platforms and as processes on UNIX®. DB2 agents are the most common type of EDUs. These agents perform most of the SQL processing on behalf of applications. Prefetchers and page cleaners are other common EDUs.

A set of subagents might be assigned to process the client application requests. Multiple subagents can be assigned if the machine where the server resides has multiple processors or is part of a partitioned database. For example, in a symmetric multiprocessing (SMP) environment, multiple SMP subagents can exploit the many processors.

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

Buffer pools are areas of database server memory where database pages of user table 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. If more of the data needed by applications is present in a buffer pool, less time is required to access the data than to find it on disk.

The configuration of the buffer pools, as well as prefetcher and page cleaner EDUs, controls how quickly data can be accessed and how readily available it is to applications.

- **Prefetchers** retrieve data from disk and move it into the buffer pool before applications need the data. For example, applications needing to scan through large volumes of data would have to wait for data to be moved from disk into the 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 to the buffer pool. If you have multiple disks for storage of the database data, the data can be striped across the disks. Striping data lets the prefetchers use multiple disks at the same time to retrieve data.
- **Page cleaners** move data from the buffer pool back out to disk. Page cleaners are background EDUs that are independent of the application agents. They look for pages from the buffer pool that are no longer needed and write the pages to disk. Page cleaners ensure that there is room in the buffer pool for the pages being retrieved by the prefetchers.

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

Related concepts:

- "Prefetching data into the buffer pool" in the *Administration Guide: Performance*
- "Deadlocks between applications" on page 155
- "Database directories and files" on page 101
- "Log processing" in the *Administration Guide: Performance*
- "Update processing" in the *Administration Guide: Performance*
- "Client-server processing model" on page 5
- "Memory management" in the *Administration Guide: Performance*

- "Connection-concentrator improvements for client connections" in the *Administration Guide: Performance*

Related reference:

- "max_coordagents - Maximum number of coordinating agents configuration parameter" in the *Administration Guide: Performance*
- "max_connections - Maximum number of client connections configuration parameter" in the *Administration Guide: Performance*

Client-server processing model

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 database machine. Local applications are directly attached to the database at the server machine.

Note: How DB2® manages client connections depends on whether the connection concentrator is on or off. The connection concentrator is ON when the *max_connections* database manager configuration parameter is set larger than the *max_coordagents* configuration parameter.

- If the connection concentrator is OFF, each client application is assigned a unique 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 similar applications with many relatively small transactions, the connection concentrator improves performance by allowing many more client applications to be connected. It also reduces system resource use for each connection.

Each of the circles of the following figure represent engine dispatchable units (EDUs) which are known as "processes" on UNIX® platforms, and "threads" on Windows® NT.

A means of communicating between an application and the database manager must be established before the work the application wants done at the database can be carried out.

At A1 in the figure below, a local client establishes communications first through the db2ipccm. At A2, the db2ipccm works with a db2agent EDU, which becomes the coordinator agent for the application requests from the local client. The coordinator agent then contacts the client application at A3 to establish shared memory communications between the client application and the coordinator. The application at the local client is connected to the database at A4.

At B1 in the figure below, a remote client establishes communications through the db2tcpcm EDU. If any other communications protocol is chosen, the appropriate communication manager is used. The db2tcpcm EDU establishes TCP/IP communication between the client application and the db2tcpcm. It then works with a db2agent at B2, which becomes the coordinator agent for the application and passes the connection to this agent. At B3 the coordinator agent contacts the remote client application and is connected to the database.

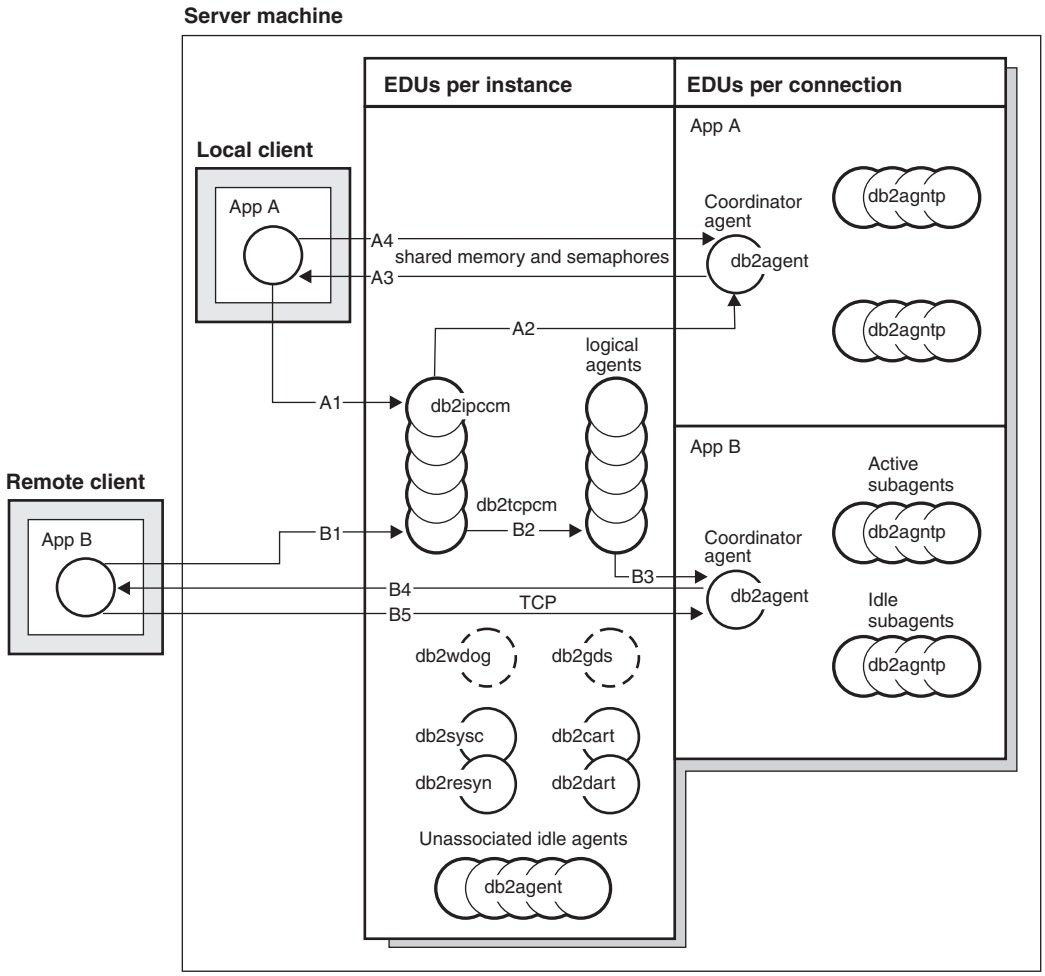


Figure 2. Process model overview

Other things to notice in this figure:

- 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, and enabled intra-partition parallelism environments, the coordinator agents distribute database requests to subagents (db2agntp). The subagents perform the requests for the application.
- There is an agent pool (db2agent) where idle and pooled agents wait for new work.
- Other EDUs manage client connections, logs, two-phase COMMITs, backup and restore tasks, and other tasks.

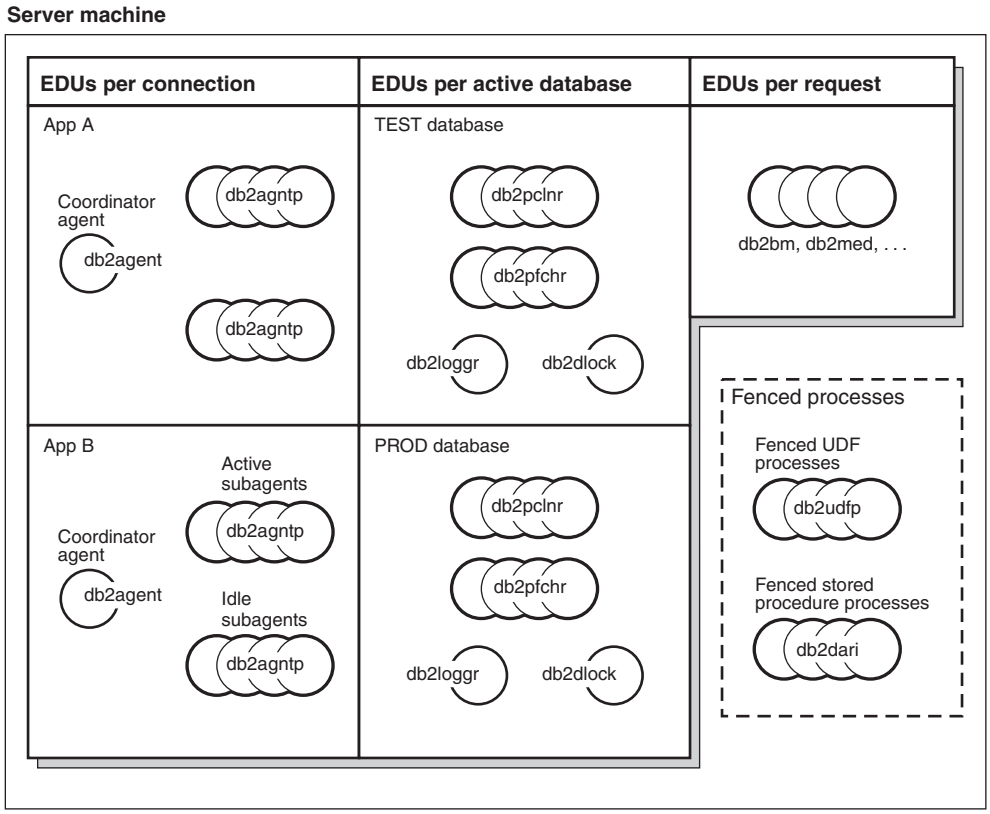


Figure 3. Process model, part 2

This figure shows additional engine dispatchable units (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).

Fenced user-defined functions (UDFs) and stored procedures, which are not shown in the figure, are managed to minimize costs associated with their creation and destruction. The default for the *keepfenced* database manager configuration parameter is "YES", which keeps the stored procedure process available for re-use at the next stored 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 need to be rigorously tested before being used.

The multiple partition processing model is a logical extension of the single partition processing model. In fact, a single common code base supports both modes of operation. The following figure shows the similarities and differences between the single partition processing model as seen in the previous two figures, and the multiple partition processing model.

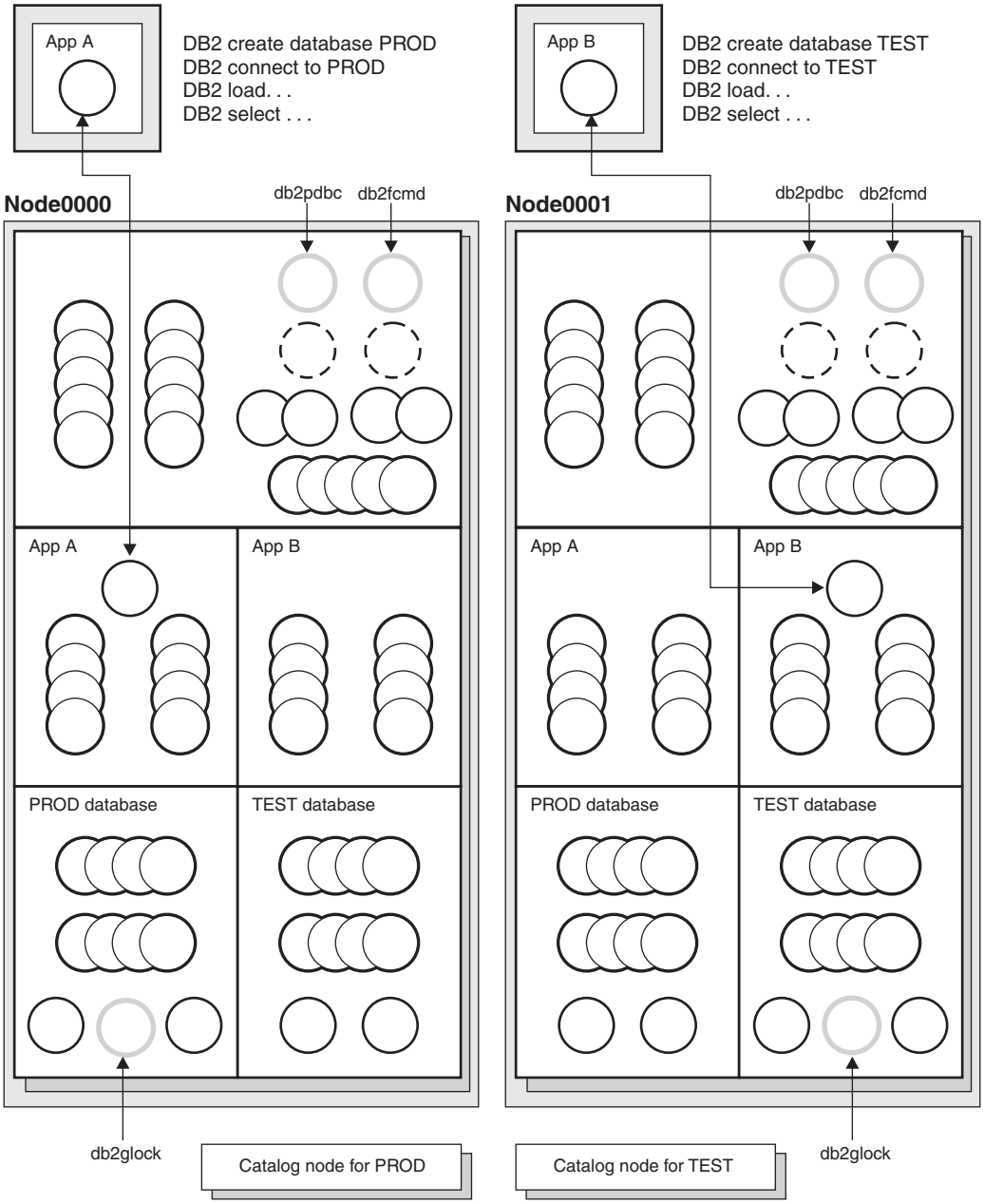


Figure 4. Process model and multiple partitions

Most engine dispatchable units (EDUs) are the same between the single partition processing model and the multiple partition processing model.

In a multiple partition (or node) environment, one of the partitions is the catalog node. The catalog keeps all of the information relating to the objects in the database.

As shown in the figure above, because Application A creates the PROD database on Node0000, the catalog for the PROD database is created on this node. Similarly, because Application B creates the TEST database on Node0001, the catalog for the TEST database is created on this node. You might want to create your databases on different nodes to balance the extra activity associated with the catalogs for each database across the nodes in your system environment.

There are additional EDUs (db2pdbc and db2fcmd) associated with the instance and these are found on each node in a multiple partition database environment. These EDUs are needed to coordinate requests across database partitions and to enable the Fast Communication Manager (FCM).

There is also an additional EDU (db2glock) associated with the catalog node for the database. This EDU controls global deadlocks across the nodes where the active database is located.

Each CONNECT from an application is represented by a connection that is associated with a coordinator agent to handle the connection. The *coordinator agent* is the agent that communicates with the application, receiving requests and sending replies. It can either satisfy the request itself or coordinate multiple subagents to work on the request. The partition where the coordinator agent exists is called the *coordinator node* of that application. The coordinator node can also be set with the SET CLIENT CONNECT_NODE command.

Parts of the database requests from the application are sent by the coordinator node to subagents at the other partitions; and all results from the other partitions are consolidated at the coordinator node before being sent back to the application.

The database partition where the CREATE DATABASE command was issued is called the "catalog node" for the database. It is at this database partition that the catalog tables are stored. Typically, all user tables are partitioned across a set of nodes.

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

Related concepts:

- "DB2 architecture and process overview" on page 3
- "Log processing" in the *Administration Guide: Performance*
- "Update processing" in the *Administration Guide: Performance*
- "Memory management" in the *Administration Guide: Performance*
- "Connection-concentrator improvements for client connections" in the *Administration Guide: Performance*

Database agents

For each database that an application accesses, various processes or threads start to perform the various application tasks. These tasks include logging, communication, and prefetching.

Database agents are engine dispatchable unit (EDU) processes or threads. Database agents do the work in the database manager that applications request. In UNIX® environments, these agents run as processes. In Intel-based operating systems such Windows®, the agents run as threads.

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. The coordinator worker agent has all the information and control blocks required to complete actions within the database manager that were requested by the application.

There are four types of worker agents:

- Idle agents
- Inactive agents
- Active coordinator agents
- Subagents

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.

Inactive agents

An inactive agent is a worker agent that is not in an active transaction, does not have an outbound connection, and does not have a local database connection or an instance attachment. Inactive agents are free to begin doing work for an application connection.

Active coordinator agents

Each process or thread of a client application has a single active agent that coordinates its work on a database. After the coordinator agent is created, it performs all database requests on behalf of its application, and communicates to other agents using inter-process 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 may become an inactive agent.

When a client disconnects from a database or detaches from an instance its coordinating agent will be:

- An active agent. If other connections are waiting, the worker agent becomes an active coordinator agent.
- Freed and marked as idle, if no connections are waiting and the maximum number of pool agents 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

In partitioned database environments and environments with intra-partition parallelism enabled, the coordinator agent distributes database requests to subagents, and these agents 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 on the database.

Agents that are not performing work for any applications 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 for client

programs or for subagents operating for existing coordinator agents. The number of available agents depends on the database manager configuration parameters *maxagents* and *num_poolagents*.

When an agent finishes its work but still has a connection to a database, it is placed in the agent pool. Regardless of whether the connection concentrator is enabled for the database, if an agent is not waked up to serve a new request within a certain period of time and the current number of active and pooled agents is greater than *num_poolagents*, the agent is terminated.

Agents from the agent pool (*num_poolagents*) are re-used as coordinator agents for the following kinds of applications:

- Remote TCP/IP-based applications
- Local applications on UNIX-based operating systems
- Both local and remote applications on Windows operating systems.

Other kinds of remote applications always create a new agent. 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 of 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.

Related concepts:

- "Database agent management" in the *Administration Guide: Performance*
- "Agents in a partitioned database" in the *Administration Guide: Performance*
- "Connection-concentrator improvements for client connections" in the *Administration Guide: Performance*
- "Configuration parameters that affect the number of agents" in the *Administration Guide: Performance*

Chapter 2. Security

Authentications, authorizations, privileges, and authorities

Security

To protect data and resources associated with a database server, DB2® Universal Database 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.

Related concepts:

- "Authentication" on page 13
- "Authorization" on page 15

Authentication

Authentication of a user is completed using a security facility outside of DB2® Universal Database (DB2 UDB). The security facility can be part of the operating

© Copyright IBM Corp. 1993-2004

system, a separate product or, in certain cases, may not exist at all. On UNIX® based systems, the security facility is in the operating system itself.

Note: In a Common Criteria compliant environment, the security facility must be available.

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.

Once authenticated:

- The user must be identified to DB2 UDB 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 based systems, a DB2 UDB *authid* is derived by transforming to uppercase letters a UNIX user ID that follows DB2 UDB 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 UDB authorization names. This mapping is done in a method similar to that used for user IDs.

DB2 UDB uses the security facility to authenticate users in one of two ways:

- DB2 UDB uses a successful security system login as evidence of identity, and allows:
  - Use of local commands to access local data
  - Use of remote connections where the server trusts the client authentication.
- DB2 UDB accepts a user ID and password combination. It uses successful validation of this pair by the security facility 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.

DB2 UDB on AIX® 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.

Creation and management of groups and users:

DB2 requires that user IDs, passwords, and groups be created at the operating system level. In addition, DB2 also requires that these user IDs, passwords, and groups be managed using facilities that are provided by the operating system. If you are responsible for creating or managing user IDs, passwords, or groups, or for assigning user IDs to groups, refer to your operating system documentation for the utilities and procedures for performing these tasks.

Related concepts:

- "Authentication methods for your server" on page 38
- "Privileges, authority levels, and database authorities" on page 15
- "Security" on page 13
- "Authorization" on page 15

Authorization

Authorization is the process whereby DB2® obtains information about an authenticated DB2 user, indicating the database operations that user may perform, and what data objects may be accessed. With each user request, there may be more than one authorization check, depending on the objects and operations involved.

Authorization is performed using DB2 facilities. DB2 tables and configuration files are used to record the permissions associated with each authorization name. The authorization name of an authenticated user, and those of groups to which the user belongs, are compared with the recorded permissions. Based on this comparison, DB2 decides whether to allow the requested access.

There are two types of permissions recorded by DB2 Universal Database™ (DB2 UDB): privileges and authority levels. 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 higher-level database manager maintenance and utility 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.

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 UDB documentation, where applicable.

Related concepts:

- "Authorization and privileges" in the *SQL Reference, Volume 1*
- "Privileges, authority levels, and database authorities" on page 15
- "Security" on page 13

Privileges, authority levels, and database authorities

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. *Database authorities* enable users to perform activities at the database level. Privileges, authority levels, and database authorities can be used together to control access to the database manager and its database objects. Users can access only those objects for which they have the required privilege, authority level, or database authority, which DB2® Universal Database (DB2 UDB) determines when it performs an authorization check for an authenticated user.

The database manager requires that each user be specifically authorized, either implicitly or explicitly, to use each database function needed to perform a specific task. *Explicit* authorities or privileges are granted to the user (GRANTEETYPE of U in the database catalogs). *Implicit* authorities or privileges are granted to a group to

which the user belongs (GRANTEETYPE of G in the database catalogs). Thus, 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 on.

Figure 5 illustrates the relationship between authorities and their span of control (database, database manager).

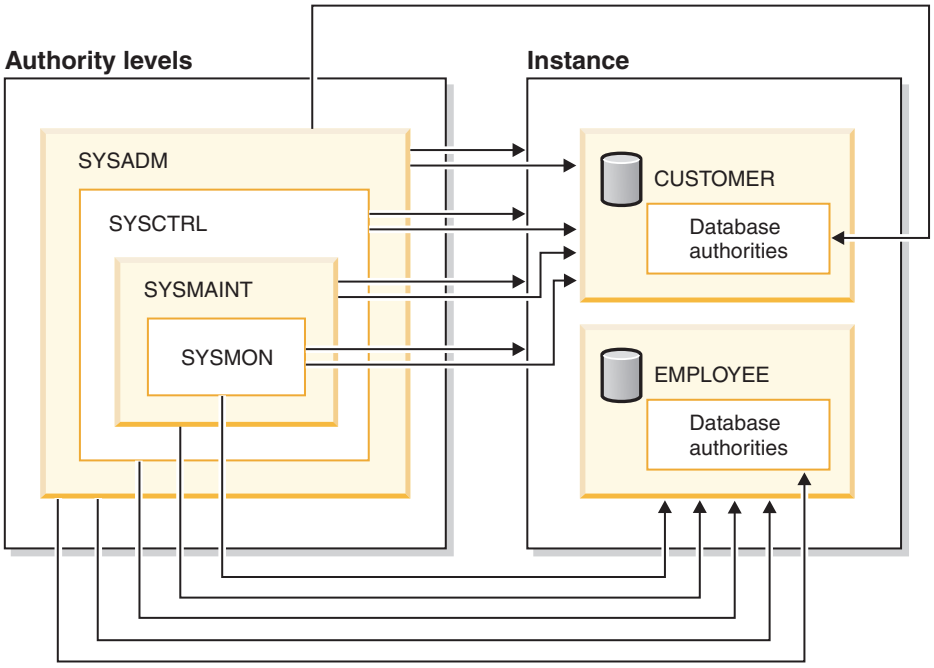


Figure 5. Hierarchy of Authorities

A user or group can have one or more of the following authorities or privileges:

- Administrative authority:
  - SYSADM (system administrator)

    The SYSADM authority level provides control over all the resources created and maintained by the database manager. The system administrator possesses all the authorities of DBADM, SYSCTRL, SYSMAINT, and SYSMON, and the authority to grant and revoke DBADM authority.

    The user who possesses SYSADM authority is responsible both for controlling the database manager, and for ensuring the safety and integrity of the data. SYSADM authority provides implicit privileges on all objects in the database, control over which users can access the database manager, and the extent of this access. For more information about SYSADM authority, see ″System administration authority (SYSADM)″.
  - DBADM (database administrator)

    The DBADM database authority provides administrative authority over a single database. This database administrator possesses the privileges required to create objects, issue database commands, and access table data. The database administrator can also grant and revoke CONTROL and individual privileges. For more information about DBADM authority, see ″Database administration authority (DBADM)″.
- System control authority:
  - SYSCTRL (system control)

The SYSCTRL authority level provides control over operations that affect system resources. For example, a user with SYSCTRL authority can create, update, stop, or drop a database. This user can also stop an instance, but cannot access table data. Users with SYSCTRL authority also have SYSMON authority. For more information about SYSCTRL authority, see ″System control authority (SYSCTRL)″.

– SYSMAINT (system maintenance)

  The SYSMAINT authority level 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. For more information about SYSMAINT authority, see ″System maintenance authority (SYSMAINT)″.

- SYSMON (system monitor authority)

  The SYSMON authority level provides the authority required to use the database system monitor. For more information about SYSMON authority, see ″System monitor authority (SYSMON)″

- Database authorities

  To perform activities such as creating a table or a routine, or for loading data into a table, specific database authorities are required. For more information, see ″Database authorities″.

- Privileges:

  Privileges are required to perform activities on database objects (for example, to create and drop an index). Privileges strictly define the tasks that a user can perform. For example, a user may have the privilege to create an index on a table, but not a trigger on the same table.

  – 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 SYSADM or DBADM authority must grant the CONTROL privilege to that object.

    In some situations, the creator of an object automatically obtains the CONTROL privilege on that object. For more information, see ″Object creation, ownership, and privileges″.

  – Individual privileges can be granted to allow a user to carry out specific tasks on specific objects.

    Users with administrative authority (SYSADM or DBADM) or 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 SYSADM authority, DBADM authority, or the CONTROL privilege to revoke the privilege.

Privileges can also be granted to PUBLIC. PUBLIC privileges apply to all users (authorization names), including any future users, regardless of whether any individual users have previously been granted the privilege.

– Implicit privileges may be granted to a user who has the privilege to execute a package. While users can run the application, they do not necessarily require explicit privileges on the data objects used within the package.

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 is given authorities and privileges and there is no user created with that authorization name. At some later time, a user can be created with that authorization name and automatically receive all of the authorities and privileges associated with that authorization name.

The REVOKE statement is used to revoke previously granted privileges. In DB2 UDB, 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.

Related concepts:

- "System administration authority (SYSADM)" on page 21
- "System control authority (SYSCTRL)" on page 21
- "System maintenance authority (SYSMAINT)" on page 22
- "Database administration authority (DBADM)" on page 25
- "LOAD authority" on page 25
- "Database authorities" on page 24
- "Schema privileges" on page 27
- "Table space privileges" on page 28
- "Table and view privileges" on page 28
- "Package privileges" on page 30
- "Index privileges" on page 31
- "Sequence privileges" on page 31
- "Controlling access to database objects" on page 43
- "Indirect privileges through a package" on page 47
- "Routine privileges" on page 31
- "Object creation, ownership, and privileges" on page 19
- "System monitor authority (SYSMON)" on page 23

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 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.

Related concepts:

- "Privileges, authority levels, and database authorities" on page 15
- "Schema privileges" on page 27
- "Table space privileges" on page 28

Schemas

A *schema* is a collection of named objects. Schemas provide a logical classification of objects in the database. A schema can contain tables, views, nicknames, triggers, functions, packages, and other objects.

A schema is also an object in the database. 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, provided that the user has IMPLICIT_SCHEMA database 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.

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, all users have IMPLICIT_SCHEMA authority. This allows any user to create objects in any schema not already in existence. An implicitly created schema allows any user to create other objects in this schema.The ability to create 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, and drop objects in the schema. A schema owner is initially given all of these privileges on the schema, with the ability to grant them 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 SYSADM or DBADM authority can change the privileges held by users on any

schema. Therefore, access to create, alter, and drop objects in any schema (even one that was implicitly created) can be controlled.

Related concepts:

- "Schema privileges" on page 27

Details on privileges, authorities, and authorization

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

System administration authority (SYSADM)

The SYSADM authority level is the highest level of administrative authority. Users with SYSADM authority can run utilities, issue database and database manager commands, and access the data in any table in any database within the database manager instance. It provides the ability to control all database objects in the instance, including databases, tables, views, indexes, packages, schemas, servers, aliases, data types, functions, procedures, triggers, table spaces, database partition groups, buffer pools, and event monitors.

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:

- Migrate a database
- Change the database manager configuration file (including specifying the groups having SYSCTRL, SYSMAINT, or SYSMON authority)
- Grant DBADM authority.

Note: When a user with SYSADM authority creates a database, that user is automatically granted explicit DBADM authority on the database. If the database creator is removed from the SYSADM group and you want to prevent that user from accessing that database as a DBADM, you must explicitly revoke the user's DBADM authority.

Related concepts:

- "System control authority (SYSCTRL)" on page 21
- "System maintenance authority (SYSMAINT)" on page 22
- "Data encryption" on page 52
- "System monitor authority (SYSMON)" on page 23

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
- Restore to a new 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 DBADM authority 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 a DBADM, you must explicitly revoke this DBADM authority.

Related concepts:

- "System maintenance authority (SYSMAINT)" on page 22
- "Database administration authority (DBADM)" on page 25
- "System monitor authority (SYSMON)" on page 23

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:

- Update database configuration files
- 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 trace
- Take database system monitor snapshots of a database manager instance or its databases.

A user with SYSMAINT, DBADM, or higher 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).

Related concepts:

- "Database administration authority (DBADM)" on page 25
- "System monitor authority (SYSMON)" on page 23

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 ACTIVE DATABASES
- LIST APPLICATIONS
- LIST DCS APPLICATIONS
- 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
- db2ResetMonitor - Reset Monitor

SYSMON authority enables the user use the following SQL table functions:

- All snapshot table functions without previously running SYSPROC.SNAPSHOT_FILEW

  SYSPROC.SNAPSHOT_FILEW 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.

Users with the SYSADM, SYSCTRL, or SYSMAINT authority level also possess SYSMON authority.

Related reference:

- "sysmon_group - System monitor authority group name" on page 790

Database authorities

Figure 6 shows the database authorities.

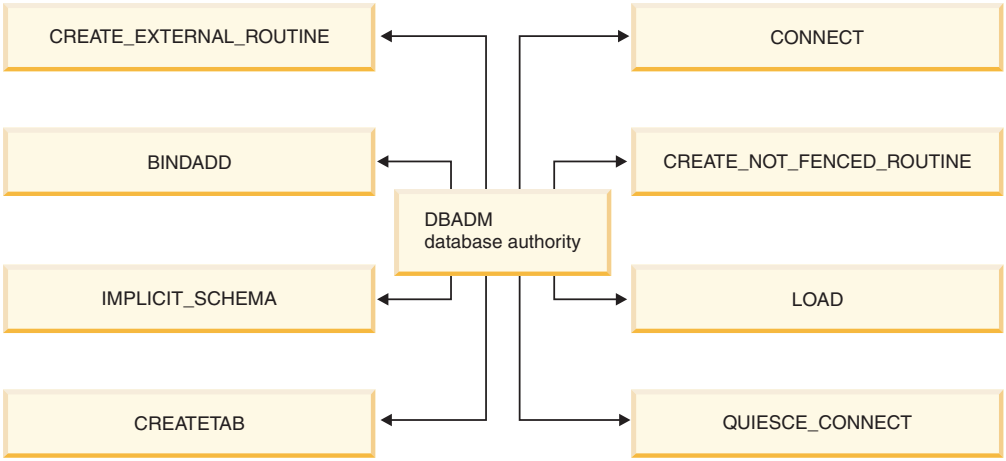


Figure 6. Database authorities

Database authorities involve actions on a database as a whole. Any user with DBADM authority possesses the complete set of database authorities, which are as follows:

- CONNECT allows a user to access the database
- BINDADD allows a user to create new packages in the database
- CREATETAB allows a user to create new tables in the database
- CREATE_EXTERNAL_ROUTINE allows a user to create a procedure for use by applications and other users of the database.
- CREATE_NOT_FENCED_ROUTINE allows a user to create a user-defined function (UDF) or procedure that is "not fenced". UDFs or procedures that are "not fenced" must be extremely well tested because the database manager does not protect its storage or control blocks from these UDFs or procedures. (As a result, a poorly written and tested UDF or procedure that is allowed to run "not fenced" can cause serious problems for your system.)

  Note: CREATE_EXTERNAL_ROUTINE is automatically granted to any user who is granted CREATE_NOT_FENCED_ROUTINE.
- 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 a user to load data into a table.
- QUIESCE_CONNECT allows a user to access the database while it is quiesced.

Only users with SYSADM or DBADM authority can grant and revoke these database authorities to and from other users.

Note: When a database is created, the following database authorities are automatically granted to PUBLIC:

- CREATETAB database authority
- BINDADD database authority
- CONNECT database authority

- IMPLICIT_SCHEMA database authority
- USE privilege on USERSPACE1 table space
- SELECT privilege on the system catalog views.

To remove any database authority, a DBADM or SYSADM must explicitly revoke the database authority from PUBLIC.

Related tasks:

- "Granting privileges" on page 44
- "Revoking privileges" on page 45

Database administration authority (DBADM)

DBADM authority is the second highest level of administrative authority. It applies only to a specific database, and allows the user to run certain utilities, issue database commands, and access the data in any table in the database. When DBADM authority is granted, BINDADD, CONNECT, CREATETAB, CREATE_EXTERNAL_ROUTINE, CREATE_NOT_FENCED_ROUTINE, IMPLICIT_SCHEMA, QUIESCE_CONNECT, and LOAD database authorities are granted as well. Only a user with SYSADM authority can grant or revoke DBADM authority. Users with DBADM authority can grant privileges on the database to others and can revoke any privilege from any user regardless of who granted it.

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

- Read log files
- Create, activate, and drop event monitors.

A user with DBADM, SYSMAINT, or higher 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.

Note: A DBADM can only perform the above functions on the database for which DBADM authority is held.

Related concepts:

- "System administration authority (SYSADM)" on page 21
- "System control authority (SYSCTRL)" on page 21
- "System maintenance authority (SYSMAINT)" on page 22
- "LOAD authority" on page 25
- "Database authorities" on page 24
- "Implicit schema authority (IMPLICIT_SCHEMA) considerations" on page 26

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.

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.

Related concepts:

- "Privileges, authorities, and authorizations required to use Load" on page 838
- "Table and view privileges" on page 28

Related reference:

- "RUNSTATS Command" in the *Command Reference*
- "QUIESCE TABLESPACES FOR TABLE" on page 340
- "LIST TABLESPACES Command" in the *Command Reference*
- "LOAD" on page 304

Implicit schema authority (IMPLICIT_SCHEMA) considerations

When a new database is created, PUBLIC is given IMPLICIT_SCHEMA database authority. With this authority, any 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 (independent of PUBLIC) 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.

Related tasks:

- "Granting privileges" on page 44
- "Revoking privileges" on page 45

Schema privileges

Schema privileges are in the object privilege category. Object privileges are shown in Figure 7.

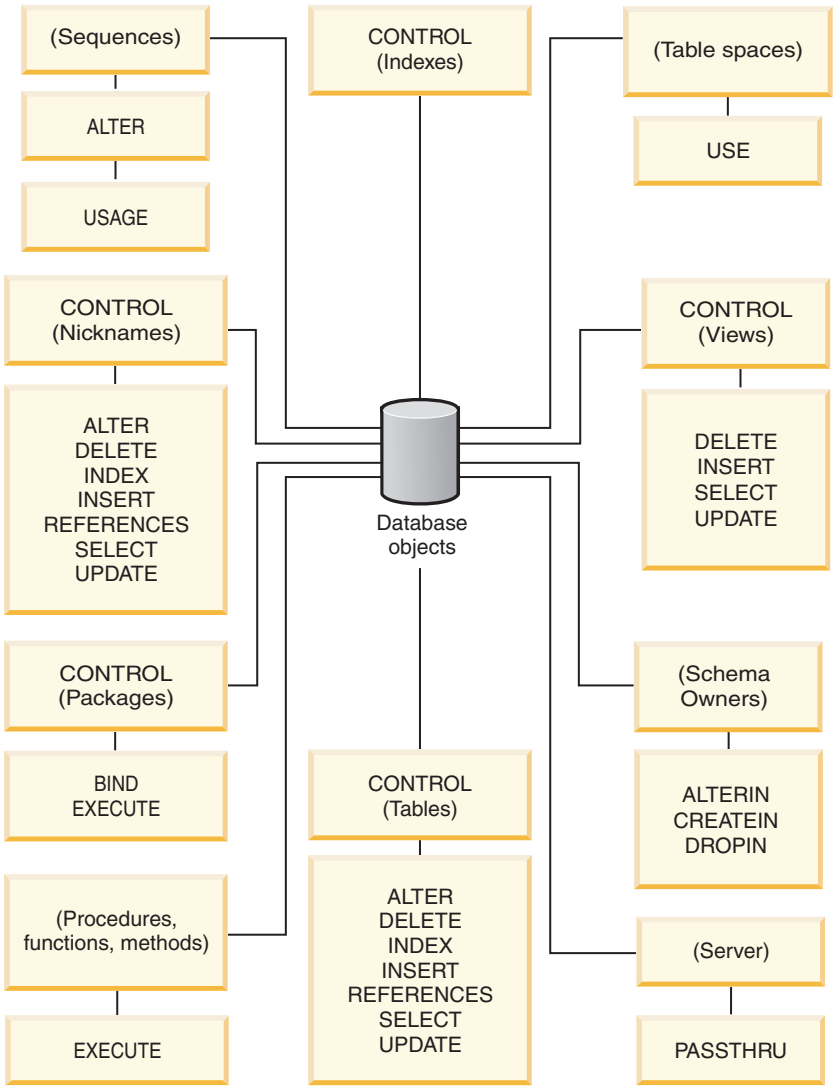


Figure 7. Object Privileges

Schema privileges involve actions on schemas in a database. A user may 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.

Related tasks:

- "Granting privileges" on page 44
- "Revoking privileges" on page 45

Related reference:

- "ALTER SEQUENCE statement" in the *SQL Reference, Volume 2*

Table space privileges

The table space privileges involve actions on the table spaces in a database. A user may 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, typically the creator who has SYSADM or SYSCTRL authority, has the USE privilege and the ability to grant this privilege to others. By default, at database creation time the USE privilege for table space USERSPACE1 is granted to PUBLIC, though this privilege can be revoked.

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

Related tasks:

- "Granting privileges" on page 44
- "Revoking privileges" on page 45

Related reference:

- "CREATE TABLE" on page 591

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 SYSADM or DBADM 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, or they have SYSADM or DBADM authority.
- 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.

Related concepts:

- "Index privileges" on page 31

Related tasks:

- "Granting privileges" on page 44
- "Revoking privileges" on page 45

Related reference:

- "ALTER TABLE" on page 525
- "CREATE VIEW" on page 656
- "SELECT" on page 902

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 SYSADM or DBADM 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 privilege 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® Universal Database (DB2 UDB) 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.

Related concepts:

- "Database authorities" on page 24

Related tasks:

- "Granting privileges" on page 44
- "Revoking privileges" on page 45

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 SYSADM or DBADM 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.

Related concepts:

- "Table and view privileges" on page 28

Related tasks:

- "Granting privileges" on page 44
- "Revoking privileges" on page 45

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.

Related tasks:

- "Granting privileges" on page 44
- "Revoking privileges" on page 45

Related reference:

- "ALTER SEQUENCE statement" in the *SQL Reference, Volume 2*

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.

Related tasks:

- "Granting privileges" on page 44
- "Revoking privileges" on page 45

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
- SYSADM or DBADM 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 the package owner or any user with SYSADM or DBADM authority.

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), SYSADM or DBADM 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.

Notes:

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:

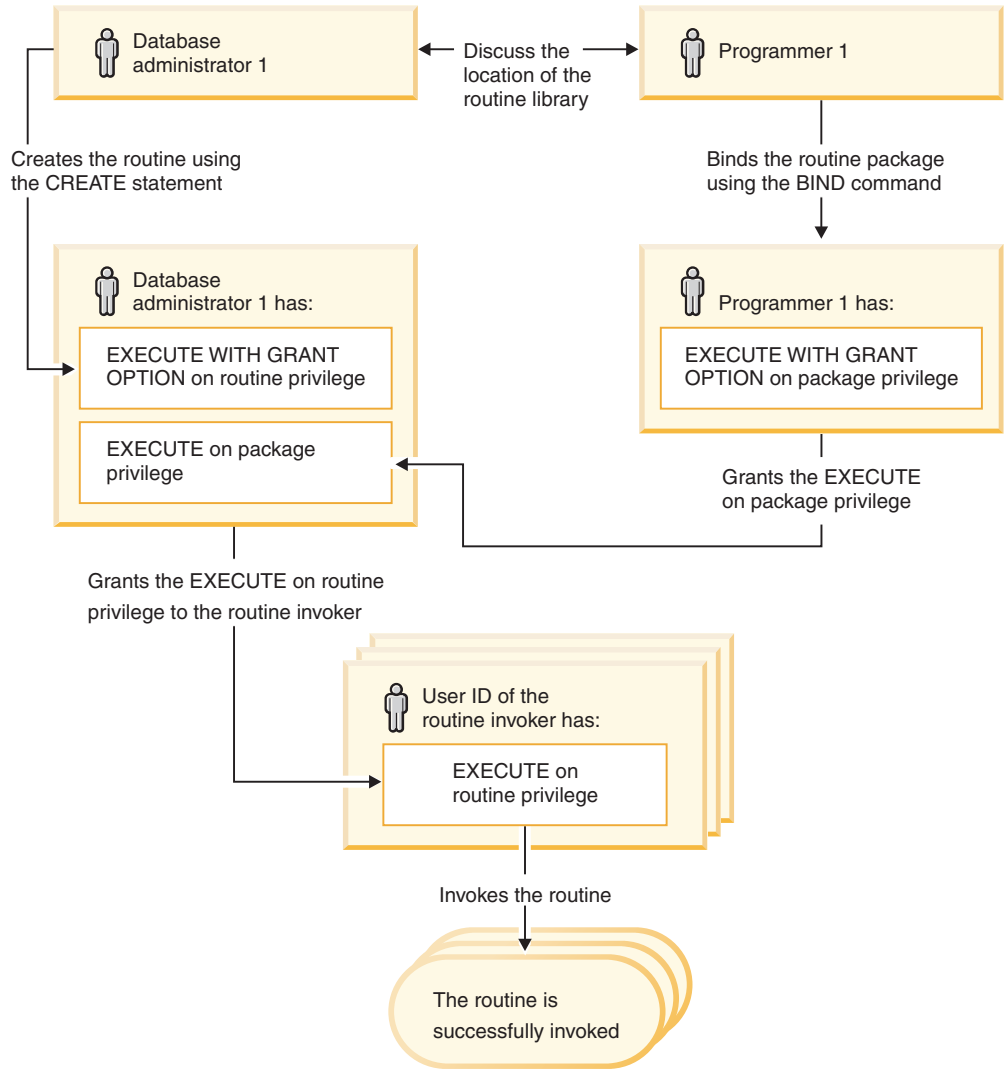


Figure 8. Managing the EXECUTE privilege on routines

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).

Related concepts:

- "Privileges, authority levels, and database authorities" on page 15
- "Effect of DYNAMICRULES bind option on dynamic SQL" on page 952
- "Routine privileges" on page 31

Related reference:

- "CREATE FUNCTION" on page 574
- "CREATE PROCEDURE" on page 588
- "BIND" on page 232

Routines that are migrated from version previous to version 8

As of DB2 Version 8.1, a routine level EXECUTE privilege exists to ensure the safe and manageable use of routines. To successfully invoke a routine, a user will now require EXECUTE privilege on the routine and will no longer require EXECUTE privilege on each of the packages of the routine. The definer of a routine, the user that registers a routine with the CREATE statement, requires EXECUTE privilege on the packages of the routine. When a user invokes a routine, it is now the routine definer's authorization ID that is checked for authority to execute the packages of a routine.

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.

The following topics are discussed:

- "Security issues when installing DB2 Universal Database" on page 36
- "Authentication methods for your server" on page 38
- "Authentication considerations for remote clients" on page 43
- "Introduction to firewall support" on page 209

- "Privileges, authority levels, and database authorities" on page 15
- "Controlling access to database objects" on page 43
- "Tasks and required authorizations" on page 53
- "Using the system catalog for security issues" on page 189.

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 issues when installing DB2 Universal Database

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

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

- On UNIX platforms, the DB2 UDB install program creates different default users for the DAS (`dasusr`), the instance owner (`db2inst`), and the fenced user (`db2fenc`).

  The DB2 UDB 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 UDB 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 UDB 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 UDB install program creates the default user `db2admin` for the DAS user, the instance owner, and fenced users. Unlike 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 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.

Note: You cannot use undefined passwords if you want your database environment to adhere to Common Criteria requirements.

After installing DB2 Universal Database 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:

Windows 9x	Any Windows 98, or Windows ME user.
Other Windows environments	On Windows NT®, Windows 2000, Windows XP, or Windows Server 2003, a valid DB2 UDB username that belongs to the Administrators group.
UNIX platforms	A valid DB2 UDB username that belongs to the primary group of the instance owner.

The SYSADM authority level provides the most powerful set of privileges available within DB2 Universal Database. As a result, you may not want all of these users to have SYSADM authority by default. DB2 UDB provides the administrator with the ability to grant and revoke privileges to groups and individual user IDs.

By updating the database manager configuration parameter *sysadm_group*, the administrator can control which group of users possesses SYSADM authority. You must follow the guidelines below to complete the security requirements for both DB2 UDB 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, or any object within 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.

Related concepts:

- "Naming rules in an NLS environment" on page 99
- "Naming rules in a Unicode environment" on page 100
- "Windows NT platform security considerations for users" on page 56
- "UNIX platform security considerations for users" on page 56
- "Authentication" on page 13
- "Authorization" on page 15
- "Location of the instance directory" in the *Administration Guide: Implementation*
- "General naming rules" on page 95

• "User, user ID and group naming rules" on page 97

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 database manager 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.

The following authentication types are provided:

SERVER
Specifies that authentication occurs on the server using local operating system security. If a user ID and password are specified during the connection or attachment attempt, they are compared to the valid user ID and password combinations at the server to determine if the user is permitted to access the instance. This is the default security mechanism.

Notes:

1. 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.
2. If you are installing DB2® Universal Database (DB2 UDB) to set up a Common Criteria certified configuation, you must specify SERVER.

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.

If the client authentication is SERVER, the client is authenticated by passing the user ID and password to the server. If the client authentication is SERVER_ENCRYPT, the client is authenticated by passing an encrypted user ID and encrypted password.

If SERVER_ENCRYPT is specified at the client and SERVER is specified at the server, an error is returned because of the mismatch in the authentication levels.

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. Specifically, all clients are trusted clients except for Windows® 9x operating systems.

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 except DRDA® clients from DB2 for OS/390® and z/OS™, DB2 for VM and VSE, and DB2 for iSeries™, 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 password is provided and at the server when a password is provided.

Table 2. Authentication Modes using TRUST_ALLCLNTS and TRUST_CLNTAUTH Parameter Combinations.

TRUST_ ALLCLNTS	TRUST_ CLNTAUTH	Untrusted non–DRDA Client Authen-tication no password	Untrusted non–DRDA Client Authen-tication with password	Trusted non–DRDA Client Authen-tication no password	Trusted non–DRDA Client Authen-tication with password	DRDA Client Authen-tication no password	DRDA Client Authen-tication with 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

Table 2. Authentication Modes using TRUST_ALLCLNTS and TRUST_CLNTAUTH Parameter Combinations. (continued)

TRUST_ ALLCLNTS	TRUST_ CLNTAUTH	Untrusted non–DRDA Client Authen-tication no password	Untrusted non–DRDA Client Authen-tication with password	Trusted non–DRDA Client Authen-tication no password	Trusted non–DRDA Client Authen-tication with password	DRDA Client Authen-tication no password	DRDA Client Authen-tication with password
DRDAONLY	CLIENT	SERVER	SERVER	SERVER	SERVER	CLIENT	CLIENT
DRDAONLY	SERVER	SERVER	SERVER	SERVER	SERVER	CLIENT	SERVER

KERBEROS

Used when both the DB2 UDB 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 UDB server. The KERBEROS authentication type is supported on clients and servers running Windows 2000, AIX®, and Solaris operating environment.

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 UDB 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 now be specified as name/instance@REALM. (This is in addition to the current DOMAIN\userID and userID@xxx.xxx.xxx.com formats accepted on Windows 2000 with DB2 UDB Version 7.1 and following.)
3. The client sends this service ticket to the server via 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 authenticaion type of the client cannot be specified as KRB_SERVER_ENCRYPT

Note: The Kerberos authentication types are only supported on clients and servers running Windows 2000, Windows XP, Windows Windows Server 2003, and AIX operating systems, as well as Solaris operating environment. Also, 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.

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. See that authentication type for more information.

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

- SQL statements.
- SQL program variable data.
- Output data from the server processing of 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) 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 on the CATALOG DATABASE command.

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 via 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 UDB-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.

Notes:

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, and those most likely to cause a problem.

   There are some things that can be done to ensure this does not happen: If you do accidentally lock yourself out of the DB2 UDB system, you have a fail-safe option available on all platforms that will allow you to override the usual DB2 UDB 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 UDB command. This special user is identified as follows:
   - UNIX® platforms: the instance owner
   - NT 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

Related concepts:

- "Authentication considerations for remote clients" on page 43
- "Partitioned database authentication considerations" in the *Administration Guide: Implementation*
- "DB2 for Windows NT and Windows NT security introduction" in the *Administration Guide: Implementation*

Related reference:

- "authentication - Authentication type" on page 783
- "trust_allclnts - Trust all clients" on page 790
- "trust_clntauth - Trusted clients authentication" on page 791

Authentication considerations for remote clients

When cataloging a database for remote access, the authentication type can be specified in the database directory entry.

For databases accessed using DB2® Connect: If a value is not specified, SERVER authentication is assumed.

The authentication type is not required. If it is not specified, the client will default 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 Universal Database™ (DB2 UDB) attempts to recover. Recovery may result in more flows to reconcile the difference, or in an error if DB2 UDB cannot recover. In the case of a mismatch, the value at the server is assumed to be correct.

Related concepts:

- "Authentication methods for your server" on page 38

Controlling access to database objects

Controlling data access requires an understanding of direct and indirect privileges, administrative authorities, and packages. This section explains these topics and provides some examples.

Directly granted privileges are stored in the system catalog.

Authorization is controlled in three ways:

- Explicit authorization is controlled through privileges controlled with the GRANT and REVOKE statements
- Implicit authorization is controlled by creating and dropping objects
- Indirect privileges are associated with packages.

Note: A database group name must be 8 characters or less when used in a GRANT or REVOKE statement, or in the Control Center. Even though a

database group name longer than 8 characters is accepted, the longer name results in an error message when users belonging to the group access database objects.

Related concepts:

- "Using the system catalog for security issues" on page 189

Related tasks:

- "Granting privileges" on page 44
- "Revoking privileges" on page 45

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

Restrictions:

To grant privileges on most database objects, the user must have SYSADM authority, DBADM authority, or CONTROL privilege on that object; or, the user must hold the privilege WITH GRANT OPTION. Privileges can be granted only on existing objects. To grant CONTROL privilege to someone else, the user must have SYSADM or DBADM authority. To grant DBADM authority, the user must have SYSADM authority.

Procedure:

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 and GROUP. 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. If the authorization name could be both a user and a group, 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
```

Related concepts:

- "Controlling access to database objects" on page 43

Related tasks:

- "Revoking privileges" on page 45

Related reference:

- "GRANT (Database Authorities)" on page 700
- "GRANT (Index Privileges)" on page 704
- "GRANT (Package Privileges)" on page 705
- "GRANT (Schema Privileges)" on page 711
- "GRANT (Table, View, or Nickname Privileges)" on page 718
- "GRANT (Server Privileges)" on page 715
- "GRANT (Table Space Privileges)" on page 716
- "GRANT (Sequence Privileges)" on page 713
- "GRANT (Routine Privileges)" on page 708

Revoking privileges

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

Restrictions:

To revoke privileges on database objects, you must have DBADM authority, SYSADM authority, or CONTROL privilege on that object. 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 SYSADM or DBADM authority. To revoke DBADM authority, you must have SYSADM authority. Privileges can only be revoked on existing objects.

Note: A user without DBADM 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.

Procedure:

If a privilege has been granted to both a user and a group with the same name, you must specify the GROUP or USER 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.

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.

Related tasks:

- “Granting privileges” on page 44

Related reference:

- “REVOKE (Database Authorities)” on page 733
- “REVOKE (Index Privileges)” on page 738
- “REVOKE (Package Privileges)” on page 740
- “REVOKE (Schema Privileges)” on page 745
- “REVOKE (Table, View, or Nickname Privileges)” on page 752
- “REVOKE (Server Privileges)” on page 749
- “REVOKE (Table Space Privileges)” on page 750

- "REVOKE (Routine Privileges)" on page 742

Managing implicit authorizations by creating and dropping objects

Procedure:

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

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.

Related tasks:

- "Granting privileges" on page 44
- "Revoking privileges" on page 45

Establishing ownership of a package

Procedure:

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. 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.
- An ID with SYSADM or 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 Universal Database™ (DB2 UDB) database products support the OWNER option.

Related reference:

- "BIND" on page 232
- "PRECOMPILE" on page 842

Indirect 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 are used for authorization checking when static SQL is bound. Privileges granted through groups are *not* used for authorization checking when static SQL is bound. The user with a valid *authID* who binds a package must either have been explicitly granted

all the privileges required to execute the static SQL statements in the package or have been implicitly granted the necessary privileges through PUBLIC unless VALIDATE RUN was specified when binding the package. If VALIDATE RUN was specified at BIND time, all authorization failures for any static SQL statements within this package will not cause the BIND to fail, and those SQL statements are revalidated at run time. PUBLIC, group, 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. To process a package with static SQL, a user need only have EXECUTE privilege on the package. This user can then indirectly obtain the privileges of the package binder for any static SQL in the package but only within the restrictions imposed by the package.

If the package includes dynamic SQL, 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 SQL.

Related concepts:

- "Indirect privileges through a package containing nicknames" on page 48
- "Effect of DYNAMICRULES bind option on dynamic SQL" on page 952

Related reference:

- "BIND" on page 232

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 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 statements and a mixture of table and nickname references, DB2® Universal Database (DB2 UDB) 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 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. 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 UDB 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.

Related concepts:

• "Indirect privileges through a package" on page 47

Controlling access to data with views

A view provides a means of controlling access or extending privileges to a table by allowing:

- 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® Universal Database (DB2 UDB) 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 SYSADM authority, DBADM 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, 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

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

Related tasks:

- "Creating a view" on page 146
- "Granting privileges" on page 44

Monitoring access to data using the audit facility

The DB2® Universal Database (DB2 UDB) audit facility generates, and allows you to maintain, an audit trail for a series of predefined database events. While not a facility that prevents access to data, the audit facility can monitor and keep a record of attempts to access or modify data objects.

SYSADM authority is required to use the audit facility administrator tool, `db2audit`.

Related concepts:

- "Introduction to the DB2 Universal Database (DB2 UDB) audit facility" on page 57

Data encryption

One part of your security plan may involve encrypting your data. To do this, you can use encryption and decryption built-in functions: ENCRYPT, DECRYPT_BIN, DECRYPT_CHAR, and GETHINT.

The ENCRYPT 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 32 631).

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.

Related reference:

- "SET ENCRYPTION PASSWORD statement" in the *SQL Reference, Volume 2*
- "DECRYPT_BIN and DECRYPT_CHAR scalar functions" in the *SQL Reference, Volume 1*
- "ENCRYPT scalar function" in the *SQL Reference, Volume 1*
- "GETHINT scalar function" in the *SQL Reference, Volume 1*

Tasks and required authorizations

Not all organizations divide job responsibilities in the same manner. Table 3 lists some other common job titles, the tasks that usually accompany them, and the authorities or privileges that are needed to carry out those tasks.

Table 3. Common Job Titles, Tasks, and Required Authorization

JOB TITLE	TASKS	REQUIRED AUTHORIZATION
Department Administrator	Oversees the departmental system; creates databases	SYSCTRL authority. SYSADM authority if the department has its own instance.
Security Administrator	Authorizes other users for some or all authorizations and privileges	SYSADM or DBADM authority.
Database Administrator	Designs, develops, operates, safeguards, and maintains one or more databases	DBADM and SYSMAINT authority over one or more databases. SYSCTRL authority in some cases.
System Operator	Monitors the database and carries out backup functions	SYSMAINT authority.
Application Programmer	Develops and tests the database manager application programs; may also create tables of test data	BINDADD, BIND on an existing package, CONNECT and CREATETAB on one or more databases, some specific schema privileges, and a list of privileges on some tables. CREATE_EXTERNAL_ROUTINE may also be required.
User Analyst	Defines the data requirements for an application program by examining the system catalog views	SELECT on the catalog views; CONNECT on one or more databases.
Program End User	Executes an application program	EXECUTE on the package; CONNECT on one or more databases. See the note following this table.
Information Center Consultant	Defines the data requirements for a query user; provides the data by creating tables and views and by granting access to database objects	DBADM authority over one or more databases.
Query User	Issues SQL statements to retrieve, add, delete, or change data; may save results as tables	CONNECT on one or more databases; CREATEIN on the schema of the tables and views being created; and, SELECT, INSERT, UPDATE, DELETE on some tables and views.

Note: If an application program contains dynamic SQL statements, the Program End User may need other privileges in addition to EXECUTE and CONNECT (such as SELECT, INSERT, DELETE, and UPDATE).

Related concepts:

- "System administration authority (SYSADM)" on page 21
- "System control authority (SYSCTRL)" on page 21
- "System maintenance authority (SYSMAINT)" on page 22
- "Database administration authority (DBADM)" on page 25
- "LOAD authority" on page 25
- "Database authorities" on page 24

Related tasks:

- "Granting privileges" on page 44
- "Revoking privileges" on page 45

Acquiring Windows users' group information using an access token

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. Once 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. Your choices when updating this registry variable include:

- TOKEN

  This choice enables access token support to lookup all groups that the user belongs to at the location where the user account is defined. This location is typically either at the domain or local to the DB2 Universal Database™ (DB2 UDB) server.
- TOKENLOCAL

  This choice enables access token support to lookup all local groups that the user belongs to on the DB2 UDB server.
- TOKENDOMAIN

This choice enables access token support to lookup all domain groups that the user belongs to on the domain.

When enabling access token support, there are several limitations that affect your account management infrastructure. When this support is enabled, DB2 UDB collects group information about the user who is connecting to the database. Subsequent operations after a successful CONNECT or ATTACH request that have dependencies on other authorization IDs will still need to use conventional group enumeration. The access token advantages of nested global groups, domain local groups, and cached credentials will not be available. For example, if, after a connection, the SET SESSION_USER is used to run under another authorization ID, only the conventional group enumeration is used to check what rights are given to the new authorization ID for the session. You will still need to grant and revoke explicit privileges to individual authorization IDs known to DB2 UDB, as opposed to the granting and revoking of privileges to groups to which the authorization IDs belongs.

If you intend to assign groups to SYSADM, SYSMAINT, or SYSCTRL, you need to ensure that the assigned groups are not nested global groups, nor domain local groups, and then the cached credential capability is not needed.

You should consider using the DB2_GRP_LOOKUP registry variable and specify the group lookup location to indicate where DB2 UDB should lookup groups using the conventional group enumeration methodology. For example,

```
db2set DB2_GRP_LOOKUP=LOCAL,TOKENLOCAL
```

This enables the access token support for enumerating local groups. Group lookup for an authorization ID different from the connected user is performed at the DB2 UDB server.

```
db2set DB2_GRP_LOOKUP=,TOKEN
```

This enables the access token support for enumerating groups at the location where the user ID is defined. Group lookup for an authorization ID different from the connected user is performed where the user ID is defined.

```
db2set DB2_GRP_LOOKUP=DOMAIN,TOKENDOMAIN
```

This enables the access token support for enumerating domain groups. Group lookup for an authorization ID different from the connected user is performed where the user ID is defined.

Applications using dynamic SQL in a package bound using DYNAMICRULES RUN (which is the default) is run under the privileges of the person who runs the application. In this case, the already mentioned limitations do not apply. This would include applications written to use JDBC and DB2 CLI.

Access token support can be enabled with all authentications types except CLIENT authentication.

Note: For Windows® NT 4.0 users, the access token support only supports the security context at the process level if your DB2 UDB applications are using local implicit connections. That is, all threads in the applications are treated as if they were running under the security context of the user executing the applications. If you require different user security contexts for different threads, you should consider moving to Windows 2000 or later; or consider changing your DB2 UDB applications to use explicit connections.

Related concepts:

- "Security issues when installing DB2 Universal Database" on page 36

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.

Windows NT platform security considerations for users

System Administration (SYSADM) authority is granted to any valid DB2® Universal Database (DB2 UDB) user account which belongs to the local Administrators group on the machine where the account is defined.

By default in a Windows® domain environment, only domain users that belong to the Administrators group at the Domain Controller have SYSADM authority on an instance. Since DB2 UDB always performs authorization at the machine where the account is defined, adding a domain user to the local Administrators group on the server does not grant the domain user SYSADM authority to the group.

Note: In a domain environment such as is found in Windows NT®, DB2 UDB only authenticates the first 64 groups that meet the requirements and restrictions, and to which a user ID belongs. You may have more than 64 groups.

To avoid adding a domain user to the Administrators group at the PDC, you should create a global group and add the users (both domain and local) that you want to grant SYSADM authority. To do this, enter the following commands:

```
DB2STOP
DB2 UPDATE DBM CFG USING SYSADM_GROUP global_group
DB2START
```

Related concepts:

- "UNIX platform security considerations for users" on page 56

UNIX platform security considerations for users

DB2® Universal Database (DB2 UDB) does not support root acting directly as a database administrator. You should use `su - <instance owner>` as the database administrator.

For security reasons, we recommend you 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 will be 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.

Related concepts:

- "Windows NT platform security considerations for users" on page 56

Chapter 3. Auditing DB2 Universal Database™ (DB2 UDB) activities

Introduction to the DB2 Universal Database (DB2 UDB) audit facility

Authentication, authorities, and privileges can be used to control known or anticipated access to data, but these methods may be insufficient to prevent unknown or unanticipated access to data. To assist in the detection of this latter type of data access, DB2® Universal Database (DB2 UDB) provides an 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 the 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 UDB 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 which would identify system misuse. Once identified, actions can be taken to reduce or eliminate such system misuse.

The audit facility acts at an instance level, recording all instance level activities and database level activities.

When working in a partitioned database environment, many of the auditable events occur at the partition at which the user is connected (the coordinator node) or at the catalog node (if they are not the same partition). The implication of this is that audit records can be generated by more than one partition. Part of each audit record contains information on the coordinator node and originating node identifiers.

The audit log (db2audit.log) and the audit configuration file (db2audit.cfg) are located in the instance's `security` subdirectory. 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.

Users of the audit facility administrator tool, db2audit, must have SYSADM authority.

The 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 UDB 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.

Authorized users of the audit facility can control the following actions within the audit facility:

- Start recording auditable events within the DB2 UDB instance.
- Stop recording auditable events within the DB2 UDB instance.

© Copyright IBM Corp. 1993-2004

- Configure the behavior of the audit facility, including selecting the categories of the auditable events to be recorded.
- Request a description of the current audit configuration.
- Flush any pending audit records from the instance and write them to the audit log.
- Extract audit records by formatting and copying them from the audit log to a flat file or ASCII delimited files. Extraction is done for one of two reasons: in preparation for analysis of log records or in preparation for pruning of log records.
- Prune audit records from the current audit log.

Note: 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 UDB 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 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, an SQL statement for dynamic SQL, a package identifier for static SQL, or an indicator of the type of operation being performed, such as CONNECT, can provide needed context when analyzing audit results.

  Note: The SQL 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.
- You can audit failures, successes, or both.

Any operation on the database 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.

Related concepts:

- "Audit facility behavior" on page 59

- "Audit facility record layouts (introduction)" on page 72
- "Audit facility tips and techniques" on page 88

Related tasks:

- "Controlling DB2 UDB audit facility activities" on page 89

Related reference:

- "Audit facility usage" on page 60
- "Audit facility messages" on page 72

Audit facility behavior

The audit facility records auditable events including those affecting database instances. For this reason, the audit facility is an independent part of DB2® Universal Database (DB2 UDB) that can operate even if the DB2 UDB instance is stopped. If the audit facility is active, then when a stopped instance is started, auditing of database events in the instance resumes.

The timing of the writing of audit records to the audit log can have a significant impact on the performance of databases in the instance. The writing of the audit records 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 this parameter is zero (0), the writing is done synchronously. The event generating the audit record will wait until the record is written to disk. The wait associated with each record causes the performance of DB2 UDB 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.

Note: The time stamp on an audit record is the same, regardless of whether the records are written synchronously or asynchronously.

In the asynchronous case, it could be possible for audit records to remain in an unfilled buffer for up to 10 minutes. To prevent this from happening for an extended period, the database manager will force the writing of the audit records regularly. An authorized user of the audit facility may also flush the audit buffer with an explicit request.

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.

The setting of the ERRORTYPE audit facility parameter controls how errors are managed between DB2 UDB 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 UDB. 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.

Depending on the API or SQL statement and the audit settings for the DB2 UDB instance, 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 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 statements in the package is carried out at the time the package is precompiled or bound. The execution of the static SQL statements within the package is not auditable. 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 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.

Related concepts:

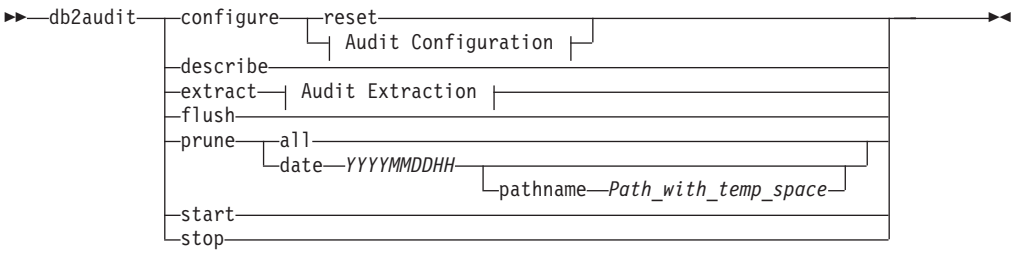
- "Introduction to the DB2 Universal Database (DB2 UDB) audit facility" on page 57

Related reference:

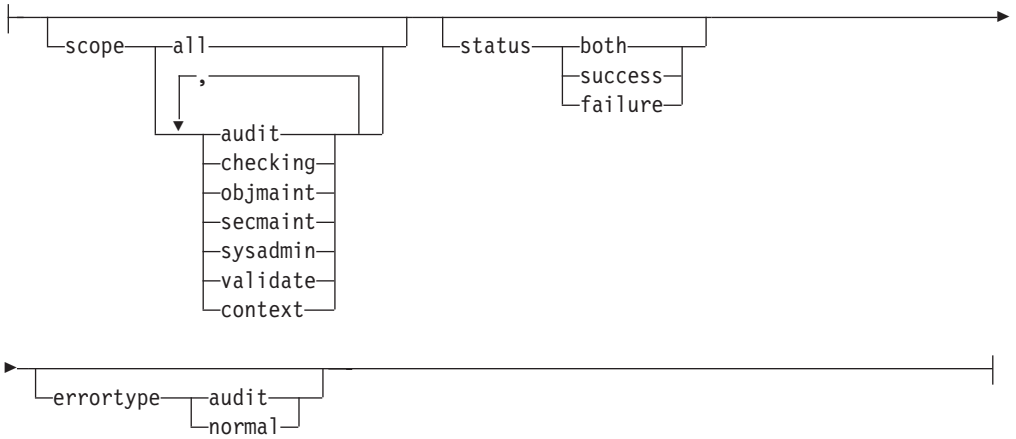
- "audit_buf_sz - Audit buffer size" on page 782
- "Audit facility usage" on page 60

Audit facility usage

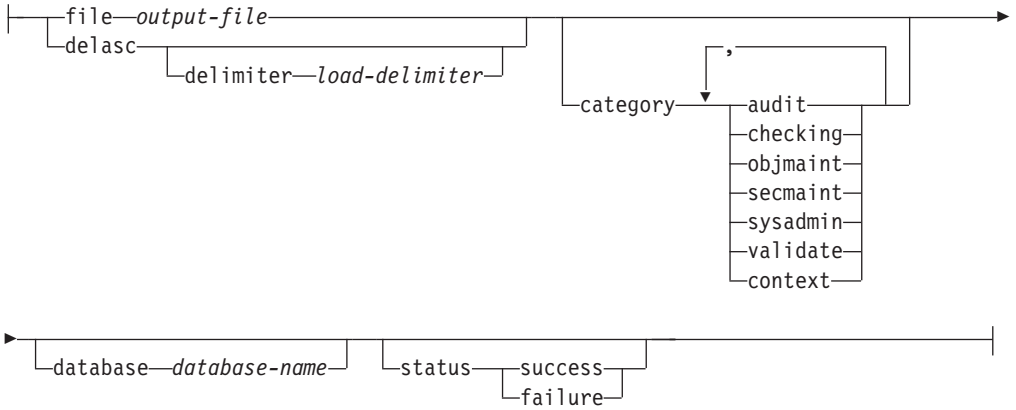
A review of each part of the following syntax diagrams will assist you in the understanding of how the audit facility can be used.



Audit Configuration:



Audit Extraction:



The following is a description and the implied use of each parameter:

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 shut down. Updates occurring when the instance is active dynamically affect the auditing being done by DB2 Universal Database™ (DB2 UDB) across all partitions. 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.

Note: The db2audit.cfg file should be created as part of the installation. You should set the permission flags (on applicable platforms) so that only the instance owner has the read/write privilege to this file.

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 is FAILURE, ERRORTYPE is NORMAL, and the audit facility is OFF). This action will create a new audit configuration file if the original has been lost or damaged.
- SCOPE. This action specifies which category or categories of events are to be audited. This action also allows a particular focus for auditing and reduces the growth of the log. It is recommended that the number and type of events being logged be limited as much as possible, otherwise the audit log will grow rapidly.

  Note: Please notice that 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.
- STATUS. This action specifies whether only successful or failing events, or both successful and failing events, should be logged.

  Note: Context events occur before the status of an operation is known. Therefore, such events are logged regardless of the value associated with this parameter.
- 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 UDB 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.

describe
This parameter displays to standard output the current audit configuration information and status.

extract This parameter allows the movement of audit records from the audit log to an indicated destination. If no optional clauses are specified, all of the audit records are extracted and placed in a flat report file. If *output_file* already exists, an error message is returned.

The following are the possible options that can be used when extracting:

- FILE. The extracted audit records are placed in a file (*output_file*). If no file name is specified, records are written to the db2audit.out file in the security subdirectory of sqllib. If no directory is specified, *output_file* is written to the current working directory.
- DELASC. The extracted audit records are placed in a delimited ASCII format suitable for loading into DB2 UDB relational tables. The output is placed in separate files: one for each category. The filenames are:
  - audit.del
  - checking.del
  - objmaint.del

– secmaint.del
– sysadmin.del
– validate.del
– context.del

These files are always written to the security subdirectory of sqllib.

The DELASC choice also allows you to override the default audit character string delimiter (“0xff”) when extracting from the audit log. You would use DELASC DELIMITER followed by the new delimiter that you wish 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-byte string representing a hexadecimal number (such as 0xff).

- CATEGORY. The audit records for the specified categories of audit events are to be extracted. If not specified, all categories are eligible for extraction.
- DATABASE. The audit records for a specified database are to be extracted. If not specified, all databases 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.

flush This parameter forces any pending audit records to be written to the audit log. Also, the audit state is reset in the engine from “unable to log” to a state of “ready to log” if the audit facility is in an error state.

prune This parameter allows for the deletion of audit records from the audit log. If the audit facility is active and the “audit” category of events has been specified for auditing, then an audit record will be logged after the audit log is pruned.

The following are the possible options that can be used when pruning:

- ALL. All of the audit records in the audit log are to be deleted.
- DATE yyyymmddhh. The user can specify that all audit records that occurred on or before the date/time specified are to be deleted from the audit log. The user may optionally supply a

  ```
  pathname
  ```

  which the audit facility will use as a temporary space when pruning the audit log. This temporary space allows for the pruning of the audit log when the disk it resides on is full and does not have enough space to allow for a pruning operation.

start This parameter causes the audit facility to begin auditing events based on the contents of the db2audit.cfg file. In a partitioned DB2 UDB instance, auditing will begin on all 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.

stop This parameter causes the audit facility to stop auditing events. In a partitioned DB2 UDB instance, auditing will be stopped on all 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.

Related concepts:

- “Introduction to the DB2 Universal Database (DB2 UDB) audit facility” on page 57

- "Audit facility tips and techniques" on page 88

Related reference:
- "db2audit - Audit Facility Administrator Tool" on page 260

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.

Working with DB2 audit data in DB2 tables

When you use the DB2 audit facility to maintain an audit trail of database activities, by default the audit facility places the audit records in a log file. If you want, you can write the audit records from the log file to a text file, or you can write the audit records from the log file to delimited ASCII files, then load the contents of the ASCII files into DB2 tables. When the audit data is in DB2 tables, you can select the data from the tables to answer questions that you may have about activity on your DB2 instance.

Procedure:

To work with audit data in DB2 tables:
1. Create tables to hold the DB2 audit data.
2. Create the DB2 audit data files.
3. Use the load utility to populate the tables with the data.
4. Select the table data.

Related concepts:
- "Audit facility behavior" on page 59
- "Audit facility tips and techniques" on page 88

Related tasks:
- "Creating tables to hold the DB2 audit data" on page 64
- "Creating DB2 audit data files" on page 67
- "Loading DB2 audit data into tables" on page 69
- "Selecting DB2 audit data from tables" on page 71

Related reference:
- "Audit facility usage" on page 60

Creating tables to hold the DB2 audit data

Before you can work with audit data in 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.

Prerequisites:
- 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.)

Procedure:

The examples that follow show how to the create tables that will hold all of the records from all of the ASCII 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 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. Issue the following command. 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. Issue the following command:

   ```
   SET CURRENT SCHEMA = 'AUDIT'
   ```

4. To create the table that will contain records from the audit.del file, issue the following SQL statement:

   ```
   CREATE TABLE AUDIT (TIMESTAMP CHAR(26),
                       CATEGORY CHAR(8),
                       EVENT VARCHAR(32),
                       CORRELATOR INTEGER,
                       STATUS INTEGER,
                       USERID VARCHAR(1024),
                       AUTHID VARCHAR(128))
   ```

5. To create the table that will contain records from the checking.del file, issue the following SQL statement:

   ```
   CREATE TABLE CHECKING (TIMESTAMP CHAR(26),
                          CATEGORY CHAR(8),
                          EVENT VARCHAR(32),
                          CORRELATOR INTEGER,
                          STATUS INTEGER,
                          DATABASE CHAR(8),
                          USERID VARCHAR(1024),
                          AUTHID VARCHAR(128),
                          NODENUM SMALLINT,
                          COORDNUM SMALLINT,
                          APPID VARCHAR(255),
                          APPNAME VARCHAR(1024),
                          PKGSCHEMA VARCHAR(128),
                          PKGNAME VARCHAR(128),
                          PKGSECNUM SMALLINT,
                          OBJSCHEMA VARCHAR(128),
                          OBJNAME VARCHAR(128),
                          OBJTYPE VARCHAR(32),
                          ACCESSAPP CHAR(18),
                          ACCESSATT CHAR(18),
                          PKGVER VARCHAR(64))
   ```

6. To create the table that will contain records from the objmaint.del file, issue the following SQL statement:

```
CREATE TABLE OBJMAINT (TIMESTAMP CHAR(26),
                       CATEGORY CHAR(8),
                       EVENT VARCHAR(32),
                       CORRELATOR INTEGER,
                       STATUS INTEGER,
                       DATABASE CHAR(8),
                       USERID VARCHAR(1024),
                       AUTHID VARCHAR(128),
                       NODENUM SMALLINT,
                       COORDNUM SMALLINT,
                       APPID VARCHAR(255),
                       APPNAME VARCHAR(1024),
                       PKGSCHEMA VARCHAR(128),
                       PKGNAME VARCHAR(128),
                       PKGSECNUM SMALLINT,
                       OBJSCHEMA VARCHAR(128),
                       OBJNAME VARCHAR(128),
                       OBJTYPE VARCHAR(32),
                       PACKVER VARCHAR(64))
```

7. To create the table that will contain records from the secmaint.del file, issue the following SQL statement:

```
CREATE TABLE SECMAINT (TIMESTAMP CHAR(26),
                       CATEGORY CHAR(8),
                       EVENT VARCHAR(32),
                       CORRELATOR INTEGER,
                       STATUS INTEGER,
                       DATABASE CHAR(8),
                       USERID VARCHAR(1024),
                       AUTHID VARCHAR(128),
                       NODENUM SMALLINT,
                       COORDNUM SMALLINT,
                       APPID VARCHAR(255),
                       APPNAME VARCHAR(1024),
                       PKGSCHEMA VARCHAR(128),
                       PKGNAME VARCHAR(128),
                       PKGSECNUM SMALLINT,
                       OBJSCHEMA VARCHAR(128),
                       OBJNAME VARCHAR(128),
                       OBJTYPE VARCHAR(32),
                       GRANTOR VARCHAR(128),
                       GRANTEE VARCHAR(128),
                       GRANTEETYPE VARCHAR(32),
                       PRIVAUTH CHAR(18),
                       PKGVER VARCHAR(64))
```

8. To create the table that will contain records from the sysadmin.del file, issue the following SQL statement:

```
CREATE TABLE SYSADMIN (TIMESTAMP CHAR(26),
                       CATEGORY CHAR(8),
                       EVENT VARCHAR(32),
                       CORRELATOR INTEGER,
                       STATUS INTEGER,
                       DATABASE CHAR(8),
                       USERID VARCHAR(1024),
                       AUTHID VARCHAR(128),
                       NODENUM SMALLINT,
                       COORDNUM SMALLINT,
                       APPID VARCHAR(255),
                       APPNAME VARCHAR(1024),
                       PKGSCHEMA VARCHAR(128),
                       PKGNAME VARCHAR(128),
                       PKGSECNUM SMALLINT,
                       PKGVER VARCHAR(64))
```

9. To create the table that will contain records from the validate.del file, issue the following SQL statement:

```
CREATE TABLE VALIDATE (TIMESTAMP CHAR(26),
                       CATEGORY CHAR(8),
                       EVENT VARCHAR(32),
                       CORRELATOR INTEGER,
                       STATUS INTEGER,
                       DATABASE CHAR(8),
                       USERID VARCHAR(1024),
                       AUTHID VARCHAR(128),
                       EXECID VARCHAR(1024),
                       NODENUM SMALLINT,
                       COORDNUM SMALLINT,
                       APPID VARCHAR(255),
                       APPNAME VARCHAR(1024),
                       AUTHTYPE VARCHAR(32),
                       PKGSCHEMA VARCHAR(128),
                       PKGNAME VARCHAR(128),
                       PKGSECNUM SMALLINT,
                       PKGVER VARCHAR(64)
                       PLUGINNAME VARCHAR(32))
```

10. To create the table that will contain records from the context.del file, issue the following SQL statement:

```
CREATE TABLE CONTEXT (TIMESTAMP CHAR(26),
                      CATEGORY CHAR(8),
                      EVENT VARCHAR(32),
                      CORRELATOR INTEGER,
                      DATABASE CHAR(8),
                      USERID VARCHAR(1024),
                      AUTHID VARCHAR(128),
                      NODENUM SMALLINT,
                      COORDNUM SMALLINT,
                      APPID VARCHAR(255),
                      APPNAME VARCHAR(1024),
                      PKGSCHEMA VARCHAR(128),
                      PKGNAME VARCHAR(128),
                      PKGSECNUM SMALLING,
                      STMTTEXT CLOB(2M),
                      PKGVER VARCHAR(64))
```

11. After creating the tables, issue the COMMIT statement to ensure that the table definitions are written to disk.
12. When you have created the tables, you are ready to extract the audit records from the db2audit.log file to delimited ASCII files.

Related tasks:

- "Creating DB2 audit data files" on page 67
- "Setting a schema" on page 141

Related reference:

- "CREATE SCHEMA" on page 588
- "CREATE TABLE" on page 591

Creating DB2 audit data files

By default, the DB2 audit facility writes audit data to the db2audit.log file. The records in this file cannot be loaded into tables. You must extract the audit records to delimited ASCII files, which can you use to populate tables.

Prerequisites:

You require SYSADM authority to use the `db2audit` command.

Procedure:

To write the audit facility records to delimited ASCII files:

1. Review the topic on audit facility usage to determine the type of DB2 activities that you want to audit. When you are satisfied with the configuration that you have set up for the audit facility, issue the following command to begin auditing:

   ```
   db2audit start
   ```

2. Issue the following command to ensure that all audit records are flushed from memory to the db2audit.log file:

   ```
   db2audit flush
   ```

3. Issue the following command to move the audit records from the db2audit.log to delimited ASCII files:

   ```
   db2audit extract delasc
   ```

   The following files are created in the security subdirectory of sqllib. If you are not auditing a particular type of event, the file for that event is created, but the file is empty.
   - audit.del
   - checking.del
   - objmaint.del
   - secmaint.del
   - sysadmin.del
   - validate.del
   - context.del

4. Issue the following command to delete the audit records from the db2audit.log file that you just extracted:

   ```
   db2audit prune date YYYYMMDDHH
   ```

   Where *YYYYMMDDHH* is the current year, month, day, and hour. Write down the value that you use because you will require this information in the next step when you populate the tables with the audit data.

   The audit facility will continue to write new audit records to the db2audit.log file, and these records will have a timestamp that is later than *YYYYMMDDHH*. Pruning records from the db2audit.log file that you have already extracted prevents you from extracting the same records a second time. All audit records that are written after *YYYYMMDDHH* will be written to the .del files the next time you extract the audit data.

5. After you create the audit data files, the next step is to use the load utility to populate the tables with the audit data.

Related tasks:

- "Loading DB2 audit data into tables" on page 69

Related reference:

- "db2audit - Audit Facility Administrator Tool" on page 260
- "Audit facility usage" on page 60
- "Audit record layout for AUDIT events" on page 73
- "Audit record layout for CHECKING events" on page 74
- "Audit record layout for OBJMAINT events" on page 79
- "Audit record layout for SECMAINT events" on page 80
- "Audit record layout for SYSADMIN events" on page 84
- "Audit record layout for VALIDATE events" on page 85

- "Audit record layout for CONTEXT events" on page 87

Loading DB2 audit data into tables

When you have created the tables to hold the audit data, you then load the data in the ASCII files into the tables.

Prerequisites:

See the topic on the privileges, authorities, and authorizations required to use the load utility for more information.

Procedure:

Use the load utility to load the 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 (`0xff`) when you extracted the audit data, you must also modify the version of the LOAD command that you use (see the topic ″ File type modifiers for load″ for more information).

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 CHARDEL0xff INSERT INTO schema.AUDIT
   ```

   Note: When specifying the file name, usen the fully qualified path name. For example, if you have DB2 UDB 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.

   After loading the AUDIT table, issue the following DELETE statement to ensure that you do not load duplicate rows into the table the next time you load it. When you extracted the audit records from the db2audit.log file, all records in the file were written to the .del files. Likely, the .del files contained records that were written after the hour to which the audit log was subsequently pruned (because the `db2audit prune` command only prunes records to a specified hour). The next time you extract the audit records, the new .del files will contain records that were previously extracted, but not deleted by the `db2audit prune` command (because they were written after the hour specified for the prune operation). Deleting rows from the table to the same hour to which the db2audit.log file was pruned ensures that the table does not contain duplicate rows, and that no audit records are lost.

   ```
   DELETE FROM schema.AUDIT WHERE TIMESTAMP > TIMESTAMP('YYYYMMDDHH0000')
   ```

   Where *YYYYMMDDHH* is the value that you specified when you pruned the db2audit.log file. Because the DB2 audit facility continues to write audit records to the db2audit.log file after it is pruned, you must specify `0000` for the minutes and seconds to ensure that audit records that were written after the db2audit.log file was pruned are not deleted from the table.
3. To load the CHECKING table, issue the following command:

   ```
   LOAD FROM checking.del OF del MODIFIED BY CHARDEL0xff INSERT INTO
     schema.CHECKING
   ```

After loading the CHECKING table, issue the following SQL statement to ensure that you do not load duplicate rows into the table the next time you load it:

```
DELETE FROM schema.CHECKING WHERE TIMESTAMP > TIMESTAMP('YYYYMMDDHH0000')
```

Where *YYYYMMDDHH* is the value that you specified when you pruned the log file.

4. To load the OBJMAINT table, issue the following command:

   ```
   LOAD FROM objmaint.del OF del MODIFIED BY CHARDEL0xff INSERT INTO
     schema.OBJMAINT
   ```

   After loading the OBJMAINT table, issue the following SQL statement to ensure that you do not load duplicate rows into the table the next time you load it:

   ```
   DELETE FROM schema.OBJMAINT WHERE TIMESTAMP > TIMESTAMP('YYYYMMDDHH0000')
   ```

   Where *YYYYMMDDHH* is the value that you specified when you pruned the log file.

5. To load the SECMAINT table, issue the following command:

   ```
   LOAD FROM secmaint.del OF del MODIFIED BY CHARDEL0xff INSERT INTO
     schema.SECMAINT
   ```

   After loading the SECMAINT table, issue the following SQL statement to ensure that you do not load duplicate rows into the table the next time you load it:

   ```
   DELETE FROM schema.SECMAINT WHERE TIMESTAMP > TIMESTAMP('YYYYMMDDHH0000')
   ```

   Where *YYYYMMDDHH* is the value that you specified when you pruned the log file.

6. To load the SYSADMIN table, issue the following command:

   ```
   LOAD FROM sysadmin.del OF del MODIFIED BY CHARDEL0xff INSERT INTO
     schema.SYSADMIN
   ```

   After loading the SYSADMIN table, issue the following SQL statement to ensure that you do not load duplicate rows into the table the next time you load it:

   ```
   DELETE FROM schema.SYSADMIN WHERE TIMESTAMP > TIMESTAMP('YYYYMMDDHH0000')
   ```

   Where *YYYYMMDDHH* is the value that you specified when you pruned the log file.

7. To load the VALIDATE table, issue the following command:

   ```
   LOAD FROM validate.del OF del MODIFIED BY CHARDEL0xff INSERT INTO
     schema.VALIDATE
   ```

   After loading the VALIDATE table, issue the following SQL statement to ensure that you do not load duplicate rows into the table the next time you load it:

   ```
   DELETE FROM schema.VALIDATE WHERE TIMESTAMP > TIMESTAMP('YYYYMMDDHH0000')
   ```

   Where *YYYYMMDDHH* is the value that you specified when you pruned the log file.

8. To load the CONTEXT table, issue the following command:

   ```
   LOAD FROM context.del OF del MODIFIED BY CHARDEL0xff INSERT INTO
     schema.CONTEXT
   ```

   After loading the CONTEXT table, issue the following SQL statement to ensure that you do not load duplicate rows into the table the next time you load it:

```
DELETE FROM schema.CONTEXT WHERE TIMESTAMP > TIMESTAMP('YYYYMMDDHH0000')
```

Where *YYYYMMDDHH* is the value that you specified when you pruned the log file.

9. After you finish loading the data into the tables, delete the .del files from the security subdirectory of the sqllib directory.
10. When you have loaded the audit data into the tables, you are ready to select data from these tables.

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. In either situation, remember both to flush the audit records to the db2audit.log file before extracting the records to the .del files, and to prune the db2audit.log file after extracting the records so that you do not load the same records into the tables more than once.

Related concepts:

- "Privileges, authorities, and authorizations required to use Load" on page 838

Related tasks:

- "Selecting DB2 audit data from tables" on page 71

Related reference:

- "File type modifiers for load" on page 326

Selecting DB2 audit data from tables

When the audit data is successfully loaded into the tables, you can select data from these tables for further analysis.

Prerequisites:

See the topic on the SELECT statement for information about the authorities and privileges required to select data from a table.

Procedure:

To select all the rows in a table:

1. Issue the `db2` command to open a DB2 command window.
2. Issue an SQL statement of the following form for each table from which you want to select audit data:

   ```
   SELECT * FROM schema.table
   ```

For example, to select all the data from the `CHECKING` table in the `AUDIT` schema, use the following statement:

```
SELECT * FROM AUDIT.CHECKING
```

The select that you perform should reflect the type of analysis that you want to do on the data. For example, you can select records according to an authorization ID (authid) to determine the type of activities that this authorization ID has been performing:

```
SELECT * FROM AUDIT.CHECKING WHERE AUTHID = authorization ID
```

Where *authorization ID* is the user ID for which you want to analyze the data.

For a description of the values that can be included in audit data, see the corresponding audit record layout topic for the table, and the list of possible returned values for the table.

Related reference:

- "Subselect" on page 904
- "SELECT" on page 902
- "Audit record layout for AUDIT events" on page 73
- "Audit record layout for CHECKING events" on page 74
- "List of possible CHECKING access approval reasons" on page 76
- "List of possible CHECKING access attempted types" on page 77
- "Audit record layout for OBJMAINT events" on page 79
- "Audit record layout for SECMAINT events" on page 80
- "List of possible SECMAINT privileges or authorities" on page 81
- "Audit record layout for SYSADMIN events" on page 84
- "List of possible SYSADMIN audit events" on page 84
- "Audit record layout for VALIDATE events" on page 85
- "Audit record layout for CONTEXT events" on page 87
- "List of possible CONTEXT audit events" on page 87

Audit facility messages

SQL1322N An error occurred when writing to the audit log file.

Explanation: The DB2 Universal Database™ (DB2 UDB) audit facility encountered an error when invoked to record an audit event to the audit log file. There is no space on the file system where the audit log resides.

User Response: The system administrator should free up space on this file system or prune the audit log to reduce its size.

When more space is available, use db2audit to flush out any data in memory, and to reset the auditor to a ready state. Ensure that appropriate extracts have occurred, or a copy of the log has been made before pruning the log, as deleted records are not recoverable.

sqlcode: -1322

sqlstate: 50830

SQL1323N An error occurred when accessing the audit configuration file.

Explanation: The audit configuration file (db2audit.cfg) could not be opened, or was invalid. Possible reasons for this error are that the db2audit.cfg file either does not exist, or has been damaged.

User Response: Take one of the following actions:

- Restore from a saved version of the file.
- Reset the audit facility configuration file by issuing

```
db2audit reset
```

sqlcode: -1323

sqlstate: 57019

Related concepts:

- "Introduction to the DB2 Universal Database (DB2 UDB) audit facility" on page 57

Audit facility record layouts (introduction)

When an audit record is extracted from the audit log using the DELASC extract option, each record will have one of the formats shown in the following tables. Each table will begin by showing the contents of a sample record. The description of each item of the record is shown one row at a time in the associated table. If the

item is important, the name of the item will be highlighted (**bold**). These items contain information that are of most interest to you.

Notes:

1. Not all fields in the sample records will have values.
2. Some fields such as "Access Attempted" are stored in the delimited ASCII format as bitmaps. In this flat report file, however, these fields will appear as a set of strings representing the bitmap values.
3. A new field called "Package Version" has been added to the record layout for the CHECKING, OBJMAINT, SECMAINT, SYSADMIN, VALIDATE, and CONTEXT events.

Related reference:

- "Audit record layout for AUDIT events" on page 73
- "Audit record layout for CHECKING events" on page 74
- "Audit record layout for OBJMAINT events" on page 79
- "Audit record layout for SECMAINT events" on page 80
- "Audit record layout for SYSADMIN events" on page 84
- "Audit record layout for VALIDATE events" on page 85
- "Audit record layout for CONTEXT events" on page 87

Details on audit facility record layouts

The various audit facility record layouts are shown in this section.

Audit record layout for AUDIT events

Table 4. Audit Record Layout for AUDIT Events

`timestamp=1998-06-24-11.54.05.151232;`**`category=AUDIT;audit event=START;`**
`event correlator=0;event status=0;`
`userid=boss;authid=BOSS;`		

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. Possible values include: CONFIGURE, DB2AUD, EXTRACT, FLUSH, PRUNE, START, STOP, and UPDATE_ADMIN_CFG
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.

Related concepts:

- "Audit facility record layouts (introduction)" on page 72

Audit record layout for CHECKING events

Table 5. Audit record layout for CHECKING events

```
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;
```

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. Possible values include: CHECKING_OBJECT and CHECKING_FUNCTION
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.
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”.

Table 5. Audit record layout for CHECKING events (continued)

```
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;
```

NAME	FORMAT	DESCRIPTION
Package Version	VARCHAR (64)	Version of the package in use at the time that the audit event occurred.

Related concepts:

- "Audit facility record layouts (introduction)" on page 72

Related reference:

- "List of possible CHECKING access approval reasons" on page 76
- "List of possible CHECKING access attempted types" on page 77
- "Audit record object types" on page 75

Audit record object types

Table 6. Audit Record Object Types Based on Audit Events

Object type	CHECKING events	OBJMAINT events	SECMAINT events
NONE	X	X	X
TABLE	X	X	X
VIEW	X	X	X
ALIAS	X	X	
FUNCTION	X	X	X
INDEX	X	X	X
INDEX EXTENSION		X	
PACKAGE	X	X	X
PACKAGE CACHE	X		
DATA_TYPE		X	
NODEGROUP	X	X	
SCHEMA	X	X	X
STORED_PROCEDURE	X	X	X
METHOD_BODY	X	X	X
BUFFERPOOL	X	X	
SEQUENCE	X	X	
TABLESPACE	X	X	X
EVENT_MONITOR	X	X	
TRIGGER		X	
DATABASE	X		X
INSTANCE	X		
FOREIGN_KEY		X	
PRIMARY_KEY		X	

Table 6. Audit Record Object Types Based on Audit Events (continued)

Object type	CHECKING events	OBJMAINT events	SECMAINT events
UNIQUE_CONSTRAINT		X	
CHECK_CONSTRAINT		X	
WRAPPER	X	X	
SERVER	X	X	X
NICKNAME	X	X	X
USER MAPPING	X	X	
SERVER OPTION	X	X	
TYPE&TRANSFORM	X	X	
TYPE MAPPING	X	X	
FUNCTION MAPPING	X	X	
SUMMARY TABLES	X	X	X
JAR_FILE		X	
ALL	X		
REOPT_VALUES	X		

Related reference:

- "Audit record layout for CHECKING events" on page 74
- "Audit record layout for OBJMAINT events" on page 79
- "Audit record layout for SECMAINT events" on page 80

List of possible CHECKING access approval reasons

The following is the list of possible CHECKING access approval reasons:

0x0000000000000001 ACCESS DENIED
Access is not approved; rather, it was denied.

0x0000000000000002 SYSADM
Access is approved; the application/user has SYSADM authority.

0x0000000000000004 SYSCTRL
Access is approved; the application/user has SYSCTRL authority.

0x0000000000000008 SYSMAINT
Access is approved; the application/user has SYSMAINT authority.

0x0000000000000010 DBADM
Access is approved; the application/user has DBADM authority.

0x0000000000000020 DATABASE PRIVILEGE
Access is approved; the application/user has an explicit privilege on the database.

0x0000000000000040 OBJECT PRIVILEGE
Access is approved; the application/user has an explicit privilege on the object or function.

0x0000000000000080 DEFINER
Access is approved; the application/user is the definer of the object or function.

0x0000000000000100 OWNER
Access is approved; the application/user is the owner of the object or function.

0x0000000000000200 CONTROL
Access is approved; the application/user has CONTROL privilege on the object or function.

0x0000000000000400 BIND
Access is approved; the application/user has bind privilege on the package.

0x0000000000000800 SYSQUIESCE
Access is approved; if the instance or database is in quiesce mode, the application/user may connect or attach.

0x0000000000001000 SYSMON
Access is approved; the application/user has SYSMON authority.

Related reference:

- "Audit record layout for CHECKING events" on page 74
- "List of possible CHECKING access attempted types" on page 77

List of possible CHECKING access attempted types

The following is the list of possible CHECKING access attempted types:

0x0000000000000002 ALTER
Attempt to alter an object.

0x0000000000000004 DELETE
Attempt to delete an object.

0x0000000000000008 INDEX
Attempt to use an index.

0x0000000000000010 INSERT
Attempt to insert into an object.

0x0000000000000020 SELECT
Attempt to query a table or view.

0x0000000000000040 UPDATE
Attempt to update data in an object.

0x0000000000000080 REFERENCE
Attempt to establish referential constraints between objects.

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.

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 on an object to another user ID.

0x0000000000080000 REVOKE
Attempt to revoke privileges on an object from a user ID.

0x0000000000100000 LOCK
Attempt to lock an object.

0x0000000000200000 RENAME
Attempt to rename an object.

0x0000000000400000 CONNECT
Attempt to connect to an object.

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.

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.

Related reference:

- "Audit record layout for CHECKING events" on page 74
- "List of possible CHECKING access approval reasons" on page 76

Audit record layout for OBJMAINT events

Table 7. Audit Record Layout for OBJMAINT Events

```
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;
```

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. Possible values include: CREATE_OBJECT, RENAME_OBJECT, and DROP_OBJECT
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.
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.

Related concepts:

- “Introduction to the DB2 Universal Database (DB2 UDB) audit facility” on page 57

Related reference:

- "Audit record object types" on page 75

Audit record layout for SECMAINT events

Table 8. Audit Record Layout for SECMAINT Events

```
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;
```

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. Possible values include: GRANT, REVOKE, IMPLICIT_GRANT, IMPLICIT_REVOKE, SET_SESSION_USER, and UPDATE_DBM_CFG.
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.
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".
Grantor	VARCHAR (128)	Grantor ID.
Grantee	VARCHAR (128)	Grantee ID for which a privilege or authority was granted or revoked.

Table 8. Audit Record Layout for SECMAINT Events (continued)

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;**		

NAME	**FORMAT**	**DESCRIPTION**
Grantee Type	VARCHAR (32)	Type of the grantee that was granted to or revoked from. Possible values include: USER, GROUP, or BOTH.
Privilege or Authority	CHAR(18)	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".
Package Version	VARCHAR (64)	Version of the package in use at the time the audit event occurred.

Related concepts:

- "Audit facility record layouts (introduction)" on page 72

Related reference:

- "List of possible SECMAINT privileges or authorities" on page 81
- "Audit record object types" on page 75

List of possible SECMAINT privileges or authorities

The following is the list of possible SECMAINT privileges or authorities:

0x0000000000000001 Control Table
Control privilege granted or revoked on a table or view.

0x0000000000000002 ALTER TABLE
Privilege granted or revoked to alter a table.

0x0000000000000004 ALTER TABLE with GRANT
Privilege granted or revoked to alter a table with granting of privileges allowed.

0x0000000000000008 DELETE TABLE
Privilege granted or revoked to drop a table or view.

0x0000000000000010 DELETE TABLE with GRANT
Privilege granted or revoked to drop a table with granting of privileges allowed.

0x0000000000000020 Table Index
Privilege granted or revoked on an index.

0x0000000000000040 Table Index with GRANT
Privilege granted or revoked on an index with granting of privileges allowed.

0x0000000000000080 Table INSERT
Privilege granted or revoked on an insert on a table or view.

0x0000000000000100 Table INSERT with GRANT
Privilege granted or revoked on an insert on a table with granting of privileges allowed.

0x0000000000000200 Table SELECT
Privilege granted or revoked on a select on a table.

0x0000000000000400 Table SELECT with GRANT
Privilege granted or revoked on a select on a table with granting of privileges allowed.

0x0000000000000800 Table UPDATE
Privilege granted or revoked on an update on a table or view.

0x0000000000001000 Table UPDATE with GRANT
Privilege granted or revoked on an update on a table or view with granting of privileges allowed.

0x0000000000002000 Table REFERENCE
Privilege granted or revoked on a reference on a table.

0x0000000000004000 Table REFERENCE with GRANT
Privilege granted or revoked on a reference on a table with granting of privileges allowed.

0x0000000000020000 CREATEIN Schema
CREATEIN privilege granted or revoked on a schema.

0x0000000000040000 CREATEIN Schema with GRANT
CREATEIN privilege granted or revoked on a schema with granting of privileges allowed.

0x0000000000080000 DROPIN Schema
DROPIN privilege granted or revoked on a schema.

0x0000000000100000 DROPIN Schema with GRANT
DROPIN privilege granted or revoked on a schema with granting of privileges allowed.

0x0000000000200000 ALTERIN Schema
ALTERIN privilege granted or revoked on a schema.

0x0000000000400000 ALTERIN Schema with GRANT
ALTERIN privilege granted or revoked on a schema with granting of privileges allowed.

0x0000000000800000 DBADM Authority
DBADM authority granted or revoked.

0x0000000001000000 CREATETAB Authority
Createtab authority granted or revoked.

0x0000000002000000 BINDADD Authority
Bindadd authority granted or revoked.

0x0000000004000000 CONNECT Authority
CONNECT authority granted or revoked.

0x0000000008000000 Create not fenced Authority
Create not fenced authority granted or revoked.

0x0000000010000000 Implicit Schema Authority
Implicit schema authority granted or revoked.

0x0000000020000000 Server PASSTHRU
Privilege granted or revoked to use the pass-through facility with this server (federated database data source).

0x0000000100000000 Table Space USE
Privilege granted or revoked to create a table in a table space.

0x0000000200000000 Table Space USE with GRANT
Privilege granted or revoked to create a table in a table space with granting of privileges allowed.

0x0000000400000000 Column UPDATE
Privilege granted or revoked on an update on one or more specific columns of a table.

0x0000000800000000 Column UPDATE with GRANT
Privilege granted or revoked on an update on one or more specific columns of a table with granting of privileges allowed.

0x0000001000000000 Column REFERENCE
Privilege granted or revoked on a reference on one or more specific columns of a table.

0x0000002000000000 Column REFERENCE with GRANT
Privilege granted or revoked on a reference on one or more specific columns of a table with granting of privileges allowed.

0x0000004000000000 LOAD Authority
LOAD authority granted or revoked.

0x0000008000000000 Package BIND
BIND privilege granted or revoked on a package.

0x0000010000000000 Package BIND with GRANT
BIND privilege granted or revoked on a package with granting of privileges allowed.

0x0000020000000000 EXECUTE
EXECUTE privilege granted or revoked on a package or a routine.

0x0000040000000000 EXECUTE with GRANT
EXECUTE privilege granted or revoked on a package or a routine with granting of privileges allowed.

0x0000080000000000 EXECUTE IN SCHEMA
EXECUTE privilege granted or revoked for all routines in a schema.

0x0000100000000000 EXECUTE IN SCHEMA with GRANT
EXECUTE privilege granted or revoked for all routines in a schema with granting of privileges allowed.

0x0000020000000000 EXECUTE IN TYPE
EXECUTE privilege granted or revoked for all routines in a type.

0x0000400000000000 EXECUTE IN TYPE with GRANT
EXECUTE privilege granted or revoked for all routines in a type with granting of privileges allowed.

0x0000080000000000 CREATE EXTERNAL ROUTINE
CREATE EXTERNAL ROUTINE privilege granted or revoked.

0x0001000000000000 QUIESCE_CONNECT
QUIESCE_CONNECT privilege granted or revoked.

Related reference:

- "Audit record layout for SECMAINT events" on page 80

Audit record layout for SYSADMIN events

Table 9. Audit Record Layout for SYSADMIN Events

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;		

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. Possible values include: Those shown in the list following this table.
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.
Package Version	VARCHAR (64)	Version of the package in use at the time the audit event occurred.

Related concepts:

- "Audit facility record layouts (introduction)" on page 72

Related reference:

- "List of possible SYSADMIN audit events" on page 84

List of possible SYSADMIN audit events

The following is the list of possible SYSADMIN audit events:

Table 10. SYSADMIN Audit Events

START_DB2
STOP_DB2
CREATE_DATABASE
DROP_DATABASE
UPDATE_DBM_CFG
UPDATE_DB_CFG
CREATE_TABLESPACE
DROP_TABLESPACE
ALTER_TABLESPACE
RENAME_TABLESPACE
CREATE_NODEGROUP
DROP_NODEGROUP
ALTER_NODEGROUP
CREATE_BUFFERPOOL
DROP_BUFFERPOOL
ALTER_BUFFERPOOL
CREATE_EVENT_MONITOR
DROP_EVENT_MONITOR
ENABLE_MULTIPAGE
MIGRATE_DB_DIR
DB2TRC
DB2SET
ACTIVATE_DB
ADD_NODE
BACKUP_DB
CATALOG_NODE
CATALOG_DB
CATALOG_DCS_DB
CHANGE_DB_COMMENT
DEACTIVATE_DB
DROP_NODE_VERIFY
FORCE_APPLICATION
GET_SNAPSHOT
LIST_DRDA_INDOUBT_TRANSACTIONS
MIGRATE_DB
RESET_ADMIN_CFG
RESET_DB_CFG
RESET_DBM_CFG
RESET_MONITOR
RESTORE_DB
ROLLFORWARD_DB
SET_RUNTIME_DEGREE
SET_TABLESPACE_CONTAINERS
UNCATALOG_DB
UNCATALOG_DCS_DB
UNCATALOG_NODE
UPDATE_ADMIN_CFG
UPDATE_MON_SWITCHES
LOAD_TABLE
DB2AUDIT
SET_APPL_PRIORITY
CREATE_DB_AT_NODE
KILLDBM
MIGRATE_SYSTEM_DIRECTORY
DB2REMOT
DB2AUD
MERGE_DBM_CONFIG_FILE
UPDATE_CLI_CONFIGURATION
OPEN_TABLESPACE_QUERY
SINGLE_TABLESPACE_QUERY
CLOSE_TABLESPACE_QUERY
FETCH_TABLESPACE
OPEN_CONTAINER_QUERY
FETCH_CONTAINER_QUERY
CLOSE_CONTAINER_QUERY
GET_TABLESPACE_STATISTICS
DESCRIBE_DATABASE
ESTIMATE_SNAPSHOT_SIZE
READ_ASYNC_LOG_RECORD
PRUNE_RECOVERY_HISTORY
UPDATE_RECOVERY_HISTORY
QUIESCE_TABLESPACE
UNLOAD_TABLE
UPDATE_DATABASE_VERSION
CREATE_INSTANCE
DELETE_INSTANCE
SET_EVENT_MONITOR
GRANT_DBADM
REVOKE_DBADM
GRANT_DB_AUTHORITIES
REVOKE_DB_AUTHORITIES
REDIST_NODEGROUP

Related reference:

- "Audit record layout for SYSADMIN events" on page 84

Audit record layout for VALIDATE events

Table 11. Audit Record Layout for VALIDATE Events

```
timestamp=1998-06-24-08.42.11.527490;category=VALIDATE;audit event=CHECK_GROUP_MEMBERSHIP;
  event correlator=2;event status=-1092;
  database=FOO;userid=boss;authid=BOSS;execution id=newton;
  application id=*LOCAL.newton.980624124210;application name=testapp;
  auth type=SERVER;
```

NAME	FORMAT	DESCRIPTION
Timestamp	CHAR(26)	Date and time of the audit event.

Table 11. Audit Record Layout for VALIDATE Events (continued)

timestamp=1998-06-24-08.42.11.527490;**category=VALIDATE;audit event=CHECK_GROUP_MEMBERSHIP;**
event correlator=2;event status=-1092;
database=FOO;userid=boss;authid=BOSS;**execution id=newton;**
application id=*LOCAL.newton.980624124210;application name=testapp;
auth type=SERVER;		

NAME	**FORMAT**	**DESCRIPTION**
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, and VALIDATE_USER.
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.

Related concepts:

• "Audit facility record layouts (introduction)" on page 72

Audit record layout for CONTEXT events

Table 12. Audit Record Layout for CONTEXT Events

```
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);
```

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. Possible values include: Those shown in the list following this table.
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.
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.
Statement Text (statement)	CLOB (2M)	Text of the SQL statement, if applicable. Null if no SQL statement text is available.
Package Version	VARCHAR (64)	Version of the package in use at the time the audit event occurred.

Related concepts:

- "Audit facility record layouts (introduction)" on page 72

Related reference:

- "List of possible CONTEXT audit events" on page 87

List of possible CONTEXT audit events

The following is the list of possible CONTEXT audit events:

Table 13. CONTEXT Audit Events

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

Related reference:

- "Audit record layout for CONTEXT events" on page 87

Audit facility tips and techniques

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 will indicate the access attempted was "ALTER" and the object type being checked was "TABLE" (note: not the column since it is table privileges that must be 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 delete 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 will have an access attempt type of "index" rather than "create".

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.

SQL 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, will indicate the nature of the completion of the attempted operation.

When extracting audit records in a delimited ASCII format suitable for loading into a DB2® Universal Database (DB2 UDB) relational table, you should be clear regarding the delimiter used within the statement text field. This can be done when extracting the delimited ASCII file and is done 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 “0xff”.

Related concepts:

- “Audit facility record layouts (introduction)” on page 72

Related reference:

- “Audit facility usage” on page 60

Controlling DB2 UDB audit facility activities

Procedure:

As part of our discussion on the control of the audit facility activities, we will use a simple scenario: A user, *newton*, runs an application called *testapp* that connects and creates a table. This same application is used in each of the examples discussed below.

We begin by presenting an extreme example: You have determined to audit all successful and unsuccessful audit events, therefore you will configure the audit facility in the following way:

```
db2audit configure scope all status both
```

Note: This creates audit records for every possible auditable event. As a result, many records are written to the audit log and this reduces the performance of your database manager. This extreme case is shown here for demonstration purposes only; there is no recommendation that you configure the audit facility with the command shown above.

After beginning the audit facility with this configuration (using "db2audit start"), and then running the *testapp* application, the following records are generated and placed in the audit log. By extracting the audit records from the log, you will see the following records generated for the two actions carried out by the application:

Action Type of Record Created

CONNECT

```
timestamp=1998-06-24-08.42.10.555345;category=CONTEXT;
audit event=CONNECT;event correlator=2;database=FOO;
application id=*LOCAL.newton.980624124210;
application name=testapp;

timestamp=1998-06-24-08.42.10.944374;category=VALIDATE;
audit event=AUTHENTICATION;event correlator=2;event status=0;
database=FOO;userid=boss;authid=BOSS;execution id=newton;
application id=*LOCAL.newton.980624124210;application name=testapp;
auth type=SERVER;

timestamp=1998-06-24-08.42.11.527490;category=VALIDATE;
audit event=CHECK_GROUP_MEMBERSHIP;event correlator=2;
event status=-1092;database=FOO;userid=boss;authid=BOSS;
execution id=newton;application id=*LOCAL.newton.980624124210;
application name=testapp;auth type=SERVER;

timestamp=1998-06-24-08.42.11.561187;category=VALIDATE;
audit event=CHECK_GROUP_MEMBERSHIP;event correlator=2;
event status=-1092;database=FOO;userid=boss;authid=BOSS;
execution id=newton;application id=*LOCAL.newton.980624124210;
application name=testapp;auth type=SERVER;

timestamp=1998-06-24-08.42.11.594620;category=VALIDATE;
audit event=CHECK_GROUP_MEMBERSHIP;event correlator=2;
event status=-1092;database=FOO;userid=boss;authid=BOSS;
execution id=newton;application id=*LOCAL.newton.980624124210;
application name=testapp;auth type=SERVER;

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;
object name=FOO;object type=DATABASE;access approval reason=DATABASE;
access attempted=CONNECT;

timestamp=1998-06-24-08.42.11.801554;category=CONTEXT;
audit event=COMMIT;event correlator=2;database=FOO;userid=boss;
authid=BOSS;application id=*LOCAL.newton.980624124210;
application name=testapp;

timestamp=1998-06-24-08.42.41.450975;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=SQLC28A1;object schema=NULLID;
object name=SQLC28A1;object type=PACKAGE;
access approval reason=OBJECT;access attempted=EXECUTE;
```

CREATE TABLE

```
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);

timestamp=1998-06-24-08.42.41.539692;category=CHECKING;
audit event=CHECKING_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;
access approval reason=DATABASE;access attempted=CREATE;

timestamp=1998-06-24-08.42.41.570876;category=CHECKING;
audit event=CHECKING_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 name=BOSS;object type=SCHEMA;access approval reason=DATABASE;
access attempted=CREATE;

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;

timestamp=1998-06-24-08.42.42.018900;category=CONTEXT;
audit event=COMMIT;event correlator=3;database=FOO;userid=boss;
authid=BOSS;application id=*LOCAL.newton.980624124210;
application name=testapp;package schema=NULLID;
package name=SQLC28A1;
```

As you can see, there are a significant number of audit records generated from the audit configuration that requests the auditing of all possible audit events and types.

In most cases, you will configure the audit facility for a more restricted or focused view of the events you wish to audit. For example, you may want to only audit those events that fail. In this case, the audit facility could be configured as follows:

```
db2audit configure scope audit,checking,objmaint,secmaint,sysadmin,
   validate status failure
```

Note: This configuration is the initial audit configuration or the one that occurs when the audit configuration is reset.

After beginning the audit facility with this configuration, and then running the *testapp* application, the following records are generated and placed in the audit log. (And we assume *testapp* has not been run before.) By extracting the audit records from the log, you will see the following records generated for the two actions carried out by the application:

Action Type of Record Created

CONNECT

```
timestamp=1998-06-24-08.42.11.527490;category=VALIDATE;
audit event=CHECK_GROUP_MEMBERSHIP;event correlator=2;
event status=-1092;database=FOO;userid=boss;authid=BOSS;
execution id=newton;application id=*LOCAL.newton.980624124210;
application name=testapp;auth type=SERVER;
```

```
timestamp=1998-06-24-08.42.11.561187;category=VALIDATE;
audit event=CHECK_GROUP_MEMBERSHIP;event correlator=2;
event status=-1092;database=FOO;userid=boss;authid=BOSS;
execution id=newton;application id=*LOCAL.newton.980624124210;
application name=testapp;auth type=SERVER;

timestamp=1998-06-24-08.42.11.594620;category=VALIDATE;
audit event=CHECK_GROUP_MEMBERSHIP;event correlator=2;
event status=-1092;database=FOO;userid=boss;authid=BOSS;
execution id=newton;application id=*LOCAL.newton.980624124210;
application name=testapp;auth type=SERVER;
```

CREATE TABLE

```
(none)
```

The are far fewer audit records generated from the audit configuration that requests the auditing of all possible audit events (except CONTEXT) but only when the event attempt fails. By changing the audit configuration you can control the type and nature of the audit records that are generated.

The audit facility can allow you to create audit records when those you want to audit have been successfully granted privileges on an object. In this case, you could configure the audit facility as follows:

```
db2audit configure scope checking status success
```

After beginning the audit facility with this configuration, and then running the *testapp* application, the following records are generated and placed in the audit log. (And we assume *testapp* has not been run before.) By extracting the audit records from the log, you will see the following records generated for the two actions carried out by the application:

Action Type of Record Created

CONNECT

```
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;

timestamp=1998-06-24-08.42.41.450975;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=SQLC28A1;object schema=NULLID;
object name=SQLC28A1;object type=PACKAGE;
access approval reason=OBJECT;access attempted=EXECUTE;

timestamp=1998-06-24-08.42.41.539692;category=CHECKING;
audit event=CHECKING_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;
access approval reason=DATABASE;access attempted=CREATE;

timestamp=1998-06-24-08.42.41.570876;category=CHECKING;
audit event=CHECKING_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 name=BOSS;object type=SCHEMA;access approval reason=DATABASE;
access attempted=CREATE;
```

CREATE TABLE
(none)

Related concepts:

- "Audit facility record layouts (introduction)" on page 72

Related reference:

- "Audit facility usage" on page 60

Chapter 4. Naming rules

General naming rules

Rules exist for the naming of all objects and users. Some of these rules are specific to the platform you are working on. For example, there is a rule regarding the use of upper and lower case letters in a name.

- On UNIX® platforms, names must be in lower case.
- On Windows® platforms, names can be in upper, lower, and mixed-case.

Unless otherwise specified, all names can include the following characters:

- A through Z. When used in most names, characters A through Z are converted from lowercase to uppercase.
- 0 through 9.
- ! % ( ) { } . – ^ ~ _ (underscore) @, #, $, and space.
- \ (backslash).

Names cannot begin with a number or with the underscore character.

Do not use SQL reserved words to name tables, views, columns, indexes, or authorization IDs.

There are other special characters that might work separately depending on your operating system and where you are working with DB2® Universal Database (DB2 UDB). 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.

You also need to consider object naming rules, workstation naming rules, naming rules in an NLS environment, and naming rules in a Unicode environment.

Related concepts:

- "DB2 UDB object naming rules" on page 95
- "Workstation naming rules" on page 98
- "User, user ID and group naming rules" on page 97
- "Federated database object naming rules" in the *Administration Guide: Implementation*

DB2 UDB object naming rules

All objects follow the General Naming Rules. In addition, some objects have additional restrictions shown in the accompanying tables.

© Copyright IBM Corp. 1993-2004

Table 14. 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 UNIX®-based implementations of DB2® Universal Database (DB2 UDB), this location is a directory path, while 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 names can have up to 8 bytes. • On Windows NT®, Windows 2000, Windows XP and Windows Server 2003 systems, 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 15. Database object naming rules

Objects	Guidelines
• Aliases • Buffer pools • Columns • Event monitors • Indexes • Methods • Nodegroups • Packages • Package versions • Schemas • Stored procedures • Tables • Table spaces • Triggers • UDFs • UDTs • Views	Can contain up to 18 bytes *except* for the following: • Table names (including view names, summary table names, alias names, and correlation names), which can contain up to 128 bytes • Column names can contain up to 30 bytes • Package names, which can contain up to 8 bytes • Schema names, which can contain up to 30 bytes • Package versions, which can contain up to 64 bytes • 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 (:).

Table 16. Federated database object naming rules

Objects	Guidelines
• Function mappings • Index specifications • Nicknames • Servers • Type mappings • User mappings • Wrappers	• Nicknames, mappings, index specifications, servers, and wrapper names cannot exceed 128 bytes. • Server and nickname options and option settings are limited to 255 bytes. • 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 8 bytes.
- The following schema names are reserved words and must not be used: SYSCAT, SYSFUN, SYSIBM, SYSSTAT.
- To avoid potential migration problems 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.

Related concepts:

- "General naming rules" on page 95

User, user ID and group naming rules

Table 17. User, user ID and group naming rules

Objects	Guidelines
• Group names • User names • User IDs	• Group names can contain up to 30 characters. • User IDs on Linux and UNIX®-based systems can contain up to 8 characters. • User names on Windows® can contain up to 30 characters. Windows NT®, Windows 2000, Windows XP and Windows Server 2003 currently have a practical limit of 20 characters. • When not using Client authentication, non-Windows 32-bit clients connecting to Windows NT, Windows 2000, Windows XP and Windows Server 2003 with user names longer than 8 characters 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. – Include accented characters.

Notes:

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 8 characters. 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.

Related concepts:

- "General naming rules" on page 95
- "Federated database object naming rules" in the *Administration Guide: Implementation*

Workstation naming rules

A *workstation name* specifies the NetBIOS name for a database server, database client, or DB2® Universal Database (DB2 UDB) Personal Edition that resides on the local workstation. This name is stored in the database manager configuration file. The workstation name is known as the *workstation nname*.

In addition, the name you specify:

- Can contain 1 to 8 characters
- Cannot include &, #, or @
- Must be unique within the network

In a partitioned database system, there is still only one workstation *nname* that represents the entire partitioned database system, but each node has its own derived unique NetBIOS *nname*.

The workstation *nname* that represents the partitioned database system is stored in the database manager configuration file for the database partition server that owns the instance.

Each node's unique *nname* is a derived combination of the workstation *nname* and the node number.

If a node does not own an instance, its NetBIOS *nname* is derived as follows:

1. The first character of the instance-owning machine's workstation *nname* is used as the first character of the node's NetBIOS *nname*.
2. The next 1 to 3 characters represent the node number. The range is from 1 to 999.
3. The remaining characters are taken from the instance-owning machine's workstation *nname*. The number of remaining characters depends on the length of the instance-owning machine's workstation *nname*. This number can be from 0 to 4.

For example:

Instance-Owning Machine's *Workstation nname*	**Node Number**	**Derived Node NetBIOS** *nname*
GEORGE	3	G3ORGE
A	7	A7

Instance-Owning Machine's *Workstation nname*	**Node Number**	**Derived Node NetBIOS** *nname*
B2	94	B942
N0076543	21	N216543
GEORGE5	1	G1RGE5

If you have changed the default workstation *nname* during the installation, the workstation *nname*'s last 4 characters should be unique across the NetBIOS network to minimize the chance of deriving a conflicting NetBIOS *nname*.

Related concepts:

• "General naming rules" on page 95

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

Related concepts:

- "General naming rules" on page 95
- "DB2 UDB object naming rules" on page 95

- "Workstation naming rules" on page 98

Naming rules in a Unicode environment

In a UCS-2 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 DB2® Universal Database (DB2 UDB).

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 UCS-2 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, while 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 UCS-2 database is to be accessed from Latin-1, Arabic, and Japanese environments, all identifiers should realistically be limited to ASCII.

Related concepts:

- "General naming rules" on page 95
- "DB2 UDB object naming rules" on page 95
- "Workstation naming rules" on page 98

Chapter 5. 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.

It is recommended that you explicitly state where you would like the database created.

In the directory you specify 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 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_directory>/<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. Each file has a duplicate copy to provide a backup.
- The files SQLSPCS.1 and SQLSPCS.2 contain table space information. Each file has a duplicate copy to provide a backup.
- The SQLDBCON file contains database configuration information. Do not edit this file. To change configuration parameters, use either the Control Center or the command-line statements UPDATE DATABASE CONFIGURATION and RESET DATABASE CONFIGURATION.
- 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 DB2TSCHNG.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

© Copyright IBM Corp. 1993-2004

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 and SQLOGMIR.LFH, contain information about the active logs.

  Recovery processing uses information from this file 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

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*.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)

Related concepts:

- "Comparison of SMS and DMS table spaces" on page 117
- "DMS device considerations" in the *Administration Guide: Performance*
- "SMS table spaces" in the *Administration Guide: Performance*
- "DMS table spaces" in the *Administration Guide: Performance*
- "Illustration of the DMS table-space address map" in the *Administration Guide: Performance*
- "Understanding the recovery history file" in the *Data Recovery and High Availability Guide and Reference*

Related reference:

- "CREATE DATABASE" on page 252

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.

From the Control Center, you can access a number of utilities that are designed to assist you in determining the size requirements of various database objects:

- You can select an object and then use the "Estimate Size" utility. This utility can tell you the current size of an existing object, such as a table. You can then change the object, and the utility will calculate new estimated values for the object. The utility will help you approximate storage requirements, taking future growth into account. It gives more than a single estimate of the size of the object. It also provides possible size ranges for the object: both the smallest size, based on current values, and the largest possible size.
- You can determine the relationships between objects by using the "Show Related" window.
- You can select any database object on the instance and request "Generate DDL". This function uses the `db2look` utility to generate data definition statements for the database.

In each of these cases, either the "Show SQL" or the "Show Command" button is available to you. You can also save the resulting SQL statements or commands in script files to be used later. All of these utilities have online help to assist you.

Keep these utilities in mind as you work through the planning of your physical database requirements.

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

- System Catalog Tables
- User Table Data
- Long Field Data

- Large Object (LOB) Data
- Index Space
- Log File Space
- Temporary Work Space

Space requirements related to the following are not discussed:
- 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

Related concepts:
- "Space requirements for system catalog tables" on page 104
- "Space requirements for user table data" on page 105
- "Space requirements for long field data" on page 106
- "Space requirements for large object data" on page 107
- "Space requirements for indexes" on page 108
- "Space requirements for log files" on page 110
- "Space requirements for temporary tables" in the *Administration Guide: Planning*

Related reference:
- "db2look - DB2 Statistics and DDL Extraction Tool Command" in the *Command Reference*

Space requirements for system catalog tables

System catalog tables are created when a database is created. The system tables grow as database objects and privileges are added to the database. Initially, they use approximately 3.5 MB of disk space.

The amount of space allocated for the catalog tables depends on the type of table space, and the extent size of the table space containing the catalog tables. For example, if a DMS table space with an extent size of 32 is used, the catalog table space will initially be allocated 20 MB of space.

Note: For databases with multiple partitions, the catalog tables reside only on the partition from which the CREATE DATABASE command was issued. Disk space for the catalog tables is only required for that partition.

Related concepts:
- "Space requirements for database objects" on page 103
- "Definition of system catalog tables" on page 136

Space requirements for user table data

By default, table data is stored on 4 KB pages. Each page (regardless of page size) contains 68 bytes of overhead for the database manager. This leaves 4028 bytes to hold user data (or rows), although no row on a 4 KB page can exceed 4005 bytes in length. 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, or DBCLOB data types. The rows in a table data page do, however, contain a descriptor for these columns.

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 APPEND ON statement is invoked, 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, DB2® Universal Database (DB2 UDB) will attempt to physically cluster the data according to the key order of that clustering index. When a row is inserted into the table, DB2 UDB will first look up its key value in the clustering index. If the key value is found, DB2 UDB 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, DB2 UDB 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, DB2 UDB 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 only provides an estimate. Accuracy of the estimate 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 512 GB, 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.

Having 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, which perform random row reads and writes, a smaller page size is better, because it wastes 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. An exception occurs when the row size is smaller than the page size divided by 255. In such a case, there is wasted space on each page. (There is a maximum of only 255 rows per page.) To reduce this wasted space, a smaller page size may be more appropriate.

You cannot restore a backup to a different page size.

You cannot import IXF data files that represent more than 755 columns.

Declared temporary tables can only be created in their own ″user temporary″ table space type. There is no default user temporary table space. Temporary tables cannot have LONG data. The tables are dropped implicitly when an application disconnects from the database, and estimates of their space requirements should take this into account.

Related concepts:

- “Space requirements for database objects” on page 103

Space requirements for long field data

Long field data is stored in a separate table object that is structured differently than the storage space for other data types.

Data is stored in 32 KB areas that are broken up into segments whose sizes are ″powers of two″ times 512 bytes. (Hence these segments can be 512 bytes, 1024 bytes, 2048 bytes, and so on, up to 32 768 bytes.)

Long field data types (LONG VARCHAR or LONG VARGRAPHIC) are stored in a way that enables free space to be reclaimed easily. Allocation and free space information is stored in 4 KB allocation pages, which appear infrequently throughout the object.

The amount of unused space in the object depends on the size of the long field data, and whether this size is relatively constant across all occurrences of the data. For data entries larger than 255 bytes, this unused space can be up to 50 percent of the size of the long field data.

If character data is less than the page size, and it fits into the record along with the rest of the data, the CHAR, GRAPHIC, VARCHAR, or VARGRAPHIC data types should be used instead of LONG VARCHAR or LONG VARGRAPHIC.

Related concepts:

• "Space requirements for database objects" on page 103

Space requirements for large object data

Large Object (LOB) data is stored in two separate table objects that are structured differently than the storage space for other data types.

To estimate the space required by LOB data, you need to consider the two table objects used to store data defined with these data types:

- **LOB Data Objects**

  Data is stored in 64 MB areas that are broken up into segments whose sizes are ″powers of two″ times 1024 bytes. (Hence these segments can be 1024 bytes, 2048 bytes, 4096 bytes, and so on, up to 64 MB.)

  To reduce the amount of disk space used by LOB data, you can specify the COMPACT option on the *lob-options* clause of the CREATE TABLE and the ALTER TABLE statements. The COMPACT option minimizes the amount of disk space required by allowing the LOB data to be split into smaller segments. This process does not involve data compression, but simply uses the minimum amount of space, to the nearest 1 KB boundary. Using the COMPACT option may result in reduced performance when appending to LOB values.

  The amount of free space contained in LOB data objects is influenced by the amount of update and delete activity, as well as the size of the LOB values being inserted.

- **LOB Allocation Objects**

  Allocation and free space information is stored in 4 KB allocation pages that are separated from the actual data. The number of these 4 KB pages is dependent on the amount of data, including unused space, allocated for the large object data. The overhead is calculated as follows: one 4 KB page for every 64 GB, plus one 4 KB page for every 8 MB.

If character data is less than the page size, and it fits into the record along with the rest of the data, the CHAR, GRAPHIC, VARCHAR, or VARGRAPHIC data types should be used instead of BLOB, CLOB, or DBCLOB.

Related concepts:

• "Space requirements for database objects" on page 103

Related reference:

• "Large objects (LOBs)" in the *SQL Reference, Volume 1*

Space requirements for indexes

For each index, the space needed can be estimated as:

```
(average index key size + 9) * 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. Do not use the maximum declared size.)
- The factor of ″2″ is for overhead, such as non-leaf pages and free space.

Notes:

1. For every column that allows NULLs, 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”.

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 + 9) * number of rows * 3.2
```

where the factor of ″3.2″ is for index overhead, and space required for sorting during index creation.

Note: In the case of non-unique indexes, only five bytes are required to store duplicate key entries. The estimates shown above assume no duplicates. The space required to store an index may be over-estimated by the formula shown above.

The following two formulas can be used to estimate the number of leaf pages (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 is 12 KB. For DMS table spaces, the minimum is an extent.

- A rough estimate of the average number of keys per leaf page is:

```
(.9 * (U - (M*2))) * (D + 1)
----------------------------
      K + 7 + (5 * 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.
  – M = U / (9 + *minimumKeySize*)
  – D = average number of duplicates per key value
  – K = *averageKeySize*

  Remember that *minimumKeySize* and *averageKeysize* must have 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 `.9` can be replaced by any (100 - pctfree)/100 value, if a percent free value other than the default value of ten percent was specified during index creation.

- A more accurate estimate of the average number of keys per leaf page is:

  ```
  L = number of leaf pages = X / (avg number of keys on leaf page)
  ```

  where X is the total number of rows in the table.

  You can estimate the original size of an index as:

  ```
  (L + 2L/(average number of keys on leaf page)) * pagesize
  ```

  For DMS table spaces, add together 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, which may result in page splits.

  Use the following calculations 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 may 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 * (U - (M*2))) * (D + 1)
  ----------------------------
        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 may want to simplify the calculation by setting the value to 0).
  – M = U / (9 + *minimumKeySize* for non-leaf pages)
  – K = *averageKeySize* for non-leaf pages

  The *minimumKeySize* and the *averageKeySize* for non-leaf pages will be the same as for leaf pages, except when there are include columns. Include columns are not stored on non-leaf pages.

  You should not replace `.9` with `(100 - pctfree)/100`, unless this value is greater than `.9`, because a maximum of 10 percent free space will be left on non-leaf pages during index creation.

  The number of non-leaf pages can be estimated as follows:

  ```
  if L > 1 then {P++; Z++}
  While (Y > 1)
  {
     P = P + Y
     Y = Y / N
    Z++
  }
  ```

  where:

  – P is the number of pages (0 initially).
  – L is the number of leaf pages.
  – N is the number of keys for each non-leaf page.
  – Y = L / N
  – Z is the number of levels in the index tree (1 initially).

  Total number of pages is:

  ```
  T = (L + P + 2) * 1.0002
  ```

  The additional 0.02 percent is for overhead, including space map pages.

  The amount of space required to create the index is estimated as:

```
T * pagesize
```

Related concepts:

- "Indexes" in the *SQL Reference, Volume 1*
- "Space requirements for database objects" on page 103
- "Index cleanup and maintenance" in the *Administration Guide: Performance*

Space requirements for log files

You will require 32 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) * (logfilsiz + 2 ) * 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.

If the database is enabled for circular logging, the result of this formula will provide a sufficient amount of disk space.

If the database is enabled for roll-forward recovery, special log space requirements should be taken into consideration:

- With the *logretain* configuration parameter enabled, 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 *userexit* configuration parameter enabled, 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.

If the database is enabled for infinite logging (that is, you set *logsecond* to *-1*), the *userexit* configuration parameter must be enabled, so you will have the same disk space considerations. DB2® Universal Database (DB2 UDB) will keep at least the number of active log files specified by *logprimary* in the log path, so you should not use the value of -1 for *logsecond* in the above formula. Ensure you provide extra disk space to allow the delay caused by archiving log files.

If you are mirroring the log path, you will need to double the estimated log file space requirements.

Related concepts:

- "Space requirements for database objects" on page 103

- "Understanding recovery logs" on page 804
- "Log mirroring" in the *Data Recovery and High Availability Guide and Reference*

Related reference:

- "logfilsiz - Size of log files configuration parameter" in the *Administration Guide: Performance*
- "logprimary - Number of primary log files configuration parameter" in the *Administration Guide: Performance*
- "logsecond - Number of secondary log files configuration parameter" in the *Administration Guide: Performance*
- "mirrorlogpath - Mirror log path configuration parameter" in the *Administration Guide: Performance*

Table space design

A table space is a storage structure containing tables, indexes, large objects, and long data. Table spaces reside in database partition groups. They allow you to assign the location of database and table data directly onto containers. (A container can be a directory name, a device name, or a file name.) This can provide improved performance and more flexible configuration.

Since table spaces reside in database partition groups, the table space selected to hold a table defines how the data for that table is distributed across the database partitions in a database partition group. A single table space can span several containers. It is possible for multiple containers (from one or more table spaces) to be created on the same physical disk (or drive). For improved performance, each container should use a different disk. Figure 9 illustrates the relationship between tables and table spaces within a database, and the containers associated with that database.

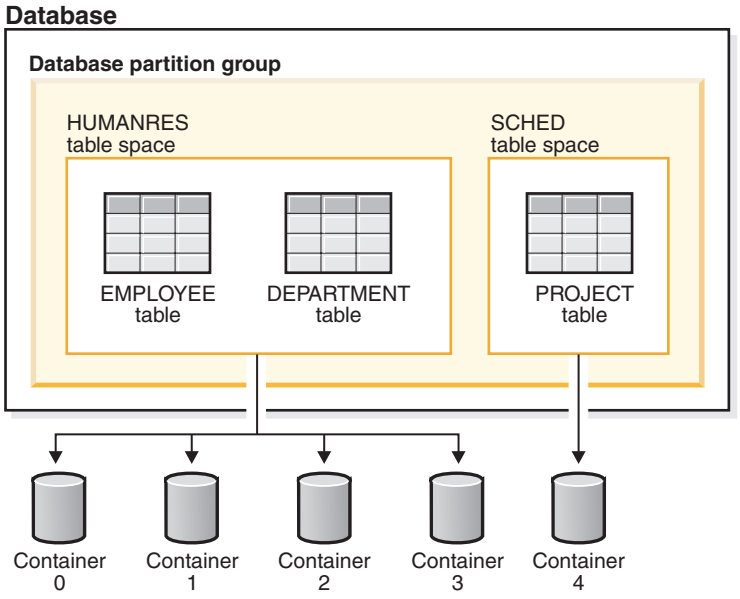


Figure 9. Table spaces and tables in a database

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 10 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.

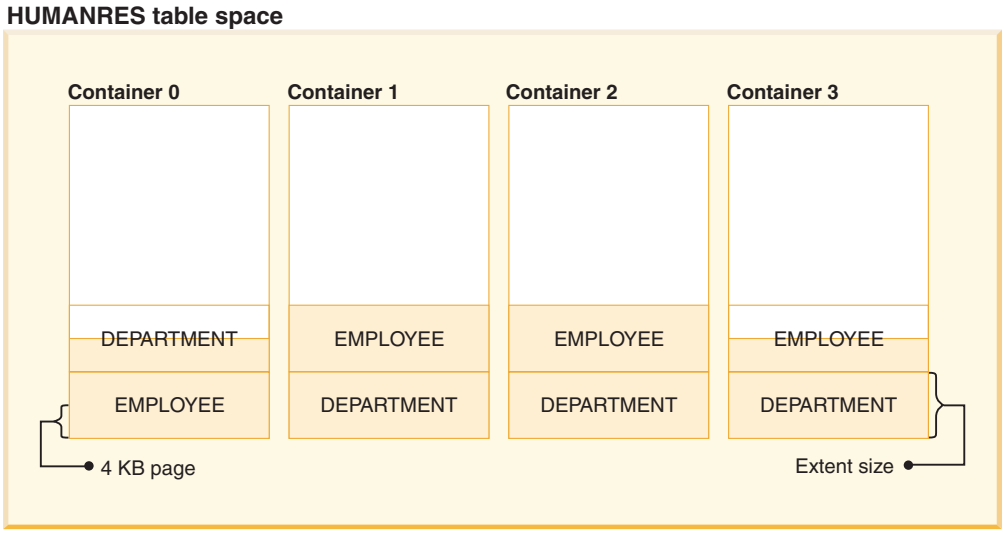


Figure 10. Containers and extents in a table space

A database must contain at least three table spaces:

- One *catalog table space*, which contains all of the system catalog tables for the database. This table space is called SYSCATSPACE, and it cannot be dropped. IBMCATGROUP is the default database partition group for this table space.
- One or more *user table spaces*, which contain all user defined tables. By default, one table space, USERSPACE1, is created. IBMDEFAULTGROUP is the default database partition group for this table space.

  You should specify a table space name when you create a table, or the results may not be what you intend.

  A table's page size is determined either by row size, or the number of columns. The maximum allowable length for a row is dependent upon the page size of the table space in which the table is created. Possible values for page size are 4 KB (the default), 8 KB, 16 KB, and 32 KB. You can use a table space with one page size for the base table, and a different table space with a different page size for long or LOB data. (Recall that SMS does not support tables that span table spaces, but that DMS does.) If the number of columns or the row size exceeds the limits for a table space's page size, an error is returned (SQLSTATE 42997).
- One or more *temporary table spaces*, which contain temporary tables. Temporary table spaces can be *system temporary table spaces* or *user temporary table spaces*. A

database must have at least one system temporary table space; by default, one system temporary table space called TEMPSPACE1 is created at database creation time. IBMTEMPGROUP is the default database partition group for this table space. User temporary table spaces are *not* created by default at database creation time.

If a database uses more than one temporary table space and a new temporary object is needed, the optimizer will choose an appropriate page size for this object. That object will then be allocated to the temporary table space with the corresponding page size. If there is more than one temporary table space with that page size, then the table space will be chosen in a round-robin fashion. In most circumstances, it is not recommended to have more than one temporary table space of any one page size.

If queries are running against tables in table spaces that are defined with a page size larger than the 4 KB default (for example, an ORDER BY on 1012 columns), some of them may fail. This will occur if there are no temporary table spaces defined with a larger page size. You may need to create a temporary table space with a larger page size (8 KB, 16 KB, or 32 KB). Any DML (Data Manipulation Language) statement could fail unless there exists a temporary table space with the same page size as the largest page size in the user table space.

You should define a single SMS temporary table space with a page size equal to the page size used in the majority of your user table spaces. This should be adequate for typical environments and workloads.

In a partitioned database environment, the catalog node will contain all three default table spaces, and the other database partitions will each contain only TEMPSPACE1 and USERSPACE1.

There are two types of table space, both of which can be used in a single database:

- System managed space, in which the operating system's file manager controls the storage space.
- Database managed space, in which the database manager controls the storage space.

Related concepts:

- "Table spaces and other storage structures" in the *SQL Reference, Volume 1*
- "System managed space" on page 114
- "Database managed space" on page 116
- "Comparison of SMS and DMS table spaces" on page 117
- "Table space disk I/O" in the *Administration Guide: Planning*
- "Workload considerations in table space design" in the *Administration Guide: Planning*
- "Extent size" in the *Administration Guide: Planning*
- "Relationship between table spaces and buffer pools" on page 118
- "Relationship between table spaces and database partition groups" in the *Administration Guide: Planning*
- "Temporary table space design" in the *Administration Guide: Planning*
- "Catalog table space design" on page 119

Related tasks:

- "Creating a table space" on page 137

- "Optimizing table space performance when data is on RAID devices" in the *Administration Guide: Planning*

Related reference:

- "CREATE TABLE" on page 591
- "CREATE TABLESPACE" on page 648

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. The storage model typically consists of many files, representing table objects, stored in the file system space. The user decides on the location of the files, DB2® Universal Database (DB2 UDB) controls their names, 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. By default, the initial table spaces created at database creation time are SMS.

Each table has at least one SMS physical file associated with it.

If you need improved insert performance, you should consider enabling multipage file allocation. This allows the system to allocate or extend the file by a full extent instead of one page at a time. For performance reasons, if you will be storing multidimensional (MDC) tables in your SMS table space, you should enable multipage file allocation. Starting in version 8.2, when you create a database (including a partitioned database), multipage file allocation is enabled by default. However, multipage file allocation may not be the default when creating a new database if you have turned on the DB2_NO_MPFA_FOR_NEW_DB registry variable. The `db2empfa` utility is used to enable multipage file allocation for a database. In a partitioned database environment, this utility must be run on each database partition. Once multipage file allocation is enabled, it cannot be disabled.

SMS table spaces are defined using the MANAGED BY SYSTEM option on the CREATE DATABASE command, or on the CREATE TABLESPACE statement. You must consider two key factors when you design your SMS table spaces:

- Containers for the table space.

  You must specify the number of containers that you want to use for your table space. 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. In a partitioned database environment, when a new partition is added to the database partition group for an SMS table space, the ALTER TABLESPACE statement can be used to add containers for the new partition.

  Each container used for an SMS table space identifies an absolute or relative directory name. Each of these directories can be located on a different file system (or physical disk). The maximum size of the table space can be estimated by:

  ```
  number of containers * (maximum file system size
     supported by the operating system)
  ```

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

Note: Care must be taken when defining the containers. If there are existing files or directories on the containers, an error (SQL0298N) is returned.

- Extent size for the table space.

  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 *dft_extent_sz* parameter is not specified on the CREATE DATABASE command, the default extent size will be set to 32.

To choose appropriate values for the 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 (`SQL00001.DAT`) is created in the first container specified for the table space, and this file is allowed to grow to the extent size. After it reaches this size, the database manager writes data to `SQL00001.DAT` in the next container. This process continues until all of the containers contain `SQL00001.DAT` files, at which time the database manager returns to the first 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 is also used for index (`SQL`*`nnnnn`*`.INX`), long field (`SQL`*`nnnnn`*`.LF`), and LOB (`SQL`*`nnnnn`*`.LB` and `SQL`*`nnnnn`*`.LBA`) files.

  Note: The SMS table space is 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.

  To help distribute data across the containers more evenly, the database manager determines which container to use first by taking the table identifier (`SQL00001.DAT` in the above example) and factoring into account the number of containers. Containers are numbered sequentially, starting at 0.

Related concepts:

- "Table space design" on page 111
- "Database managed space" on page 116
- "Comparison of SMS and DMS table spaces" on page 117

Related reference:

- "db2empfa - Enable Multipage File Allocation Command" in the *Command Reference*

Database managed space

In a DMS (Database Managed Space) table space, the database manager controls the storage space. The storage model consists of a limited number of devices or files whose space is managed by DB2® Universal Database (DB2 UDB). The database administrator decides which devices and files to use, and DB2 UDB manages the space on those devices and files. The table space is essentially an implementation of a special purpose file system designed to best meet the needs of the database manager.

A DMS table space containing user defined tables and data can be defined as:

- A *regular* table space to store any table data and optionally index data
- A *large* table space to store long field or LOB data or index data.

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.
- The maximum size of regular table spaces is 64 GB for 4 KB pages; 128 GB for 8 KB pages; 256 GB for 16 KB pages; and 512 GB for 32 KB pages. The maximum size of large table spaces is 2 TB.
- Unlike SMS table spaces, the containers that make up a DMS table space do not need 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. If, for example, the device is allocated with 5 000 pages, and the device container is defined to allocate 3 000 pages, 2 000 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. Attempting to create a table space smaller than five extents will result in an error (SQL1422N).
  - 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.)
- 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. No operational difference exists between a file and a device; however, a file can be less efficient because of the run-time overhead associated with the file system. Files are useful when:

  - Devices are not directly supported
  - A device is not available
  - Maximum performance is not required
  - You do not want to set up devices.
- If your workload involves LOBs or LONG VARCHAR data, you may derive performance benefits from file system caching. Note that LOBs and LONG VARCHARs are not buffered by DB2 UDB's buffer pool.
- Some operating systems allow you to have physical devices greater than 2 GB in size. You should consider partitioning the physical device into multiple logical devices, so that no container is larger than the size allowed by the operating system.

Related concepts:

- "Table space design" on page 111
- "System managed space" on page 114
- "Comparison of SMS and DMS table spaces" on page 117
- "Table space maps" in the *Administration Guide: Planning*
- "How containers are added and extended in DMS table spaces" in the *Administration Guide: Planning*

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.
- Creating a table space requires less initial work, because you do not have to predefine the containers.

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.
- A table can be split across multiple table spaces, based on the type of data being stored:
  - Long field and 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, you could have a table with 64 GB of regular table data, 64 GB of index data and 2 TB of long data. If you are using 8 KB pages, the table data and the index data can be as much as 128 GB. If you are using 16 KB pages, it can be as much as 256 GB. If you are using 32 KB pages, the table data and the index data can be as much as 512 GB.
- The location of the data on the disk can be controlled, if this is allowed by the operating system.
- If all table data is in a single table space, a table space can be dropped and redefined with less overhead than dropping and redefining a table.

- In general, a well-tuned set of DMS table spaces will outperform SMS table spaces.

Note: On the Solaris™ Operating Environment, DMS table spaces with raw devices are strongly recommended for performance-critical workloads.

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 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.

Related concepts:

- "Table space design" on page 111
- "System managed space" on page 114
- "Database managed space" on page 116

Relationship between table spaces and buffer pools

Each table space is associated with a specific buffer pool. The default buffer pool is IBMDEFAULTBP. If another buffer pool is to be associated with a table space, the buffer pool must exist (it is defined with the CREATE BUFFERPOOL statement), it must have the same page size, and the association is defined when the table space is created (using the CREATE TABLESPACE statement). The association between the table space and the buffer pool can be changed using the ALTER TABLESPACE statement.

Having more than one buffer pool allows you to configure the memory used by the database to improve overall performance. For example, if you have a table space with one or more large (larger than available memory) tables that are accessed randomly by users, the size of the buffer pool can be limited, because caching the data pages might not be beneficial. The table space for an online transaction application might be associated with a larger buffer pool, so that the data pages used by the application can be cached longer, resulting in faster response times. Care must be taken in configuring new buffer pools.

Note: If you have determined that a page size of 8 KB, 16 KB, or 32 KB is required by your database, each table space with one of these page sizes must be mapped to a buffer pool with the same page size.

The storage required for all the buffer pools must be available to the database manager when the database is started. If DB2® Universal Database (DB2 UDB) is unable to obtain the required storage, the database manager will start up with default buffer pools (one each of 4 KB, 8 KB, 16 KB, and 32 KB page sizes), and issue a warning.

In a partitioned database environment, you can create a buffer pool of the same size for all partitions in the database. You can also create buffer pools of different sizes on different partitions.

Related concepts:

- "Table spaces and other storage structures" in the *SQL Reference, Volume 1*

Related reference:
- "ALTER BUFFERPOOL statement" in the *SQL Reference, Volume 2*
- "ALTER TABLESPACE" on page 557
- "CREATE BUFFERPOOL statement" in the *SQL Reference, Volume 2*
- "CREATE TABLESPACE" on page 648

Catalog table space design

An SMS table space is recommended for database catalogs, for the following reasons:

- The database catalog consists of many tables of varying sizes. When using a DMS table space, a minimum of two extents are allocated for each table object. Depending on the extent size chosen, a significant amount of allocated and unused space may result. When using a DMS table space, a small extent size (two to four pages) should be chosen; otherwise, an SMS table space should be used.
- There are large object (LOB) columns in the catalog tables. LOB data is not kept in the buffer pool with other data, but is read from disk each time it is needed. Reading LOBs from disk reduces performance. Since a file system usually has its own cache, using an SMS table space, or a DMS table space built on file containers, makes avoidance of I/O possible if the LOB has previously been referenced.

Given these considerations, an SMS table space is a somewhat better choice for the catalogs.

Another factor to consider is whether you will need to enlarge the catalog table space in the future. While some platforms have support for enlarging the underlying storage for SMS containers, and while you can use redirected restore to enlarge an SMS table space, the use of a DMS table space facilitates the addition of new containers.

Related concepts:
- "Definition of system catalog tables" on page 136
- "Table space design" on page 111
- "System managed space" on page 114
- "Database managed space" on page 116

Chapter 6. Before Creating the Database

Starting DB2 UDB on UNIX

You may need to start or stop DB2 Universal Database™ (DB2 UDB) during normal business operations; for example, you must start an instance before you can perform the following tasks:

- Connect to a database on the instance
- Precompile an application
- Bind a package to a database
- Access host databases.

Prerequisites:

To start a DB2 UDB instance on your 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:

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

   where `INSTHOME` is the home directory of the instance you want to use.

Procedure:

Use one of these two methods to start the instance:

1. To start the instance using the Control Center:

1. Expand the object tree until you see the **Instances** folder. 2. Right-click the instance that you want to start, and select **start** from the pop-up menu.

2. To start the instance using the command line, enter:

```
db2start
```

Related tasks:

- "Stopping an instance on UNIX" on page 123
- "Setting the current instance" in the *Administration Guide: Implementation*
- "Starting DB2 UDB on Windows" on page 122

© Copyright IBM Corp. 1993-2004

Starting DB2 UDB on Windows

You may need to start or stop DB2 Universal Database™ (DB2 UDB) during normal business operations; for example, you must start an instance before you can perform the following tasks:

- Connect to a database on the instance
- Precompile an application
- Bind a package to a database
- Access host databases.

Prerequisites:

In order to successfully launch DB2 UDB as a service from `db2start`, the user account must have the correct privilege as defined by the Windows NT operating system to start a Windows service. The user account can be a member of the Administrators, Server Operators, or Power Users group.

Procedure:

Use one of these two methods to start the instance:

1. To start the instance using the Control Center:

   1. Expand the object tree until you see the **Instances** folder.
   2. Right-click the instance that you want to start, and select **start** from the pop-up menu.

2. To start the instance using the command line, enter:

   ```
   db2start
   ```

The `db2start` command will launch DB2 UDB as a Windows service. DB2 UDB on Windows can still be run as a process by specifying the ″/D″ switch when invoking `db2start`. DB2 UDB can also be started as a service using the Control Panel or ″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 DB2 as a process in a partitioned database environment.

Related tasks:

- “Starting DB2 UDB on UNIX” on page 121
- “Stopping an instance on UNIX” on page 123
- “Stopping an instance on Windows” on page 124

Grouping objects by schema

Database object names may be made up of a single identifier or they may 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 wish 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.

In dynamic SQL statements, a schema qualified object name implicitly uses the CURRENT SCHEMA special register value as the qualifier for unqualified object name references. In static SQL statements, the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified database object names.

Before creating your own objects, you need to 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.

Related concepts:

- "Definition of system catalog tables" on page 136

Related tasks:

- "Creating a schema" on page 140

Related reference:

- "SET SCHEMA" on page 902
- "CURRENT SCHEMA" on page 801

Stopping an instance on UNIX

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

Prerequisites:

To stop an instance on your 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.

Restrictions:

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.

Procedure:

Use one of these two methods to stop the instance:

1. To stop the instance using the Control Center:

   1. Expand the object tree until you find the **Instances** folder.
   2. Click each instance you want to stop.
   3. Right-click any of the selected instances, and select **stop** from the pop-up menu.
   4. On the Confirm stop window, click **OK**.

2. To stop the instance using the command line, enter:

   ```
   db2stop
   ```

You can use the db2stop command to stop, or drop, individual partitions within a partitioned database environment. When working in a partitioned database 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.

Related reference:

- "db2stop - Stop DB2 Command" in the *Command Reference*
- "TERMINATE Command" in the *Command Reference*

Stopping an instance on Windows

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

Prerequisites:

To stop an instance on your system, you must do the following:

1. The user account stopping the DB2 Universal Database™ (DB2 UDB) 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.

Restrictions:

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 DB2 UDB 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.

Procedure:

To stop an instance on your system, use one of the following methods:

- `db2stop`
- Stop the service using the Control Center

1. Expand the object tree until you find the **Instances** folder.
2. Click each instance you want to stop.
3. Right-click any of the selected instances, and select **stop** from the pop-up menu.
4. On the Confirm stop window, click **OK**.

- Stop using the "NET STOP" command.
- Stop the instance from within an application.

Recall that when you are using DB2 UDB in a partitioned database environment, each database partition server is started as a service. Each service must be stopped.

Related reference:

- "db2stop - Stop DB2 Command" in the *Command Reference*

Instance creation

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. 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.

It should be noted that multiple instances have some minor disadvantages:

- Additional system resources (virtual memory and disk space) are required for each instance.
- More administration is required 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® Universal Database (DB2 UDB) processes.

On UNIX® operating systems, the instance directory is located in the INSTHOME/sqllib directory, where INSTHOME is the home directory of the instance owner.

On Windows® operating systems, the instance directory is located in the /sqllib sub-directory, in the directory where DB2 UDB was installed.

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

As part of your installation procedure, you create an initial instance of DB2 UDB called “DB2”. On UNIX, the initial instance can be called anything you want within the naming rules guidelines. The instance name is used to set up the directory structure.

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

- The environment variable DB2INSTANCE is set to “DB2”.
- The DB2 registry variable DB2INSTDEF is set to “DB2”.

On UNIX, the default can be called anything you want within the naming rules guidelines.

On Windows, the instance name is the same as the name of the service, so it should not conflict. You must have the correct authorization to create a service.

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

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

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

The sample script contains statements to:

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

Related concepts:

- "Multiple instances on a UNIX operating system" in the *Administration Guide: Implementation*
- "Multiple instances on a Windows operating system" in the *Administration Guide: Implementation*

Related tasks:

- "Add an instance" in the *Administration Guide: Implementation*
- "UNIX details when creating instances" on page 128
- "Windows details when creating instances" on page 129
- "Setting the current instance" in the *Administration Guide: Implementation*
- "Auto-starting instances" in the *Administration Guide: Implementation*
- "Running multiple instances concurrently" in the *Administration Guide: Implementation*
- "Listing instances" in the *Administration Guide: Implementation*
- "Creating additional instances" in the *Administration Guide: Implementation*

Setting the DB2 UDB environment automatically on UNIX

By default, the scripts that set up the database environment when you create an instance affect the user environment for the duration of the current session only. You can change the `.profile` file to enable it to run the `db2profile` script automatically when the user logs on using the Bourne or Korn shell. For users of the C shell, you can change the `.login` file to enable it to run the `db2shrc` script file.

Procedure:

Add one of the following statements to the `.profile` or `.login` script files:

- For users who share one version of the script, add:

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

  where `INSTHOME` is the home directory of the instance that you wish to use.

- For users who have a customized version of the script in their home directory, add:

```
. USERHOME/db2profile         (for Bourne or Korn shell)
source USERHOME/db2cshrc      (in C shell)
```

  where `USERHOME` is the home directory of the user.

Related tasks:

- "Setting the DB2 UDB environment manually on UNIX" on page 128

Setting the DB2 UDB environment manually on UNIX

Procedure:

To choose which instance you want to use, enter one of the following statements at a command prompt. The period (.) and the space are required.

- For users who share one version of the script, add:

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

  where `INSTHOME` is the home directory of the instance that you wish to use.

- For users who have a customized version of the script in their home directory, add:

```
. USERHOME/db2profile         (for Bourne or Korn shell)
source USERHOME/db2cshrc      (in C shell)
```

  where `USERHOME` is the home directory of the user.

If you want to work with more than one instance at the same time, run the script for each instance that you want to use in separate windows. For example, assume that you have two instances called `test` and `prod`, and their home directories are `/u/test` and `/u/prod`.

In window 1:

- In Bourne or Korn shell, enter:

```
. /u/test/sqllib/db2profile
```

- In C shell, enter:

```
source /u/test/sqllib/db2cshrc
```

In window 2:

- In Bourne or Korn shell, enter:

```
. /u/prod/sqllib/db2profile
```

- In C shell, enter:

```
source /u/prod/sqllib/db2cshrc
```

Use window 1 to work with the `test` instance and window 2 to work with the `prod` instance.

Note: Enter the `which db2` command to ensure that your search path has been set up correctly. This command returns the absolute path of the CLP executable. Verify that it is located under the instance's `sqllib` directory.

Related tasks:

- "Setting the DB2 UDB environment automatically on UNIX" on page 127

UNIX details when creating instances

When working with UNIX operating systems, the `db2icrt` command has the following optional parameters:

- –h or –?

  This parameter is used to display a help menu for the command.

- –d

  This parameter sets the debug mode for use during problem determination.
- –a AuthType

  This parameter specifies the authentication type for the instance. Valid authentication types are SERVER, SERVER_ENCRYPT, or CLIENT. If not specified, the default is SERVER, if a DB2 Universal Database™ (DB2 UDB) server is installed. Otherwise, it is set to CLIENT.

  Notes:

  1. The authentication type of the instance applies to all databases owned by the instance.
  2. On UNIX operating systems, the authentication type DCE is not a valid choice.
- –u FencedID

  This parameter is the user under which the fenced user-defined functions (UDFs) and stored procedures will execute. This is not required if you install the DB2 UDB client or the DB2 UDB Application Development Client. For other DB2 UDB products, this is a required parameter.

  Note: FencedID may not be “root” or “bin”.
- –p PortName

  This parameter specifies the TCP/IP service name or port number to be used. This value will then be set in the instance’s database configuration file for every database in the instance.
- –s InstType

  Allows different types of instances to be created. Valid instance types are: `ese`, `wse`, `client`, and `standalone`.

Examples:

- To add an instance for a DB2 UDB server, you can use the following command:

  ```
  db2icrt -u db2fenc1 db2inst1
  ```
- If you installed the DB2 Connect Enterprise Edition only, you can use the instance name as the Fenced ID also:

  ```
  db2icrt -u db2inst1 db2inst1
  ```
- To add an instance for a DB2 UDB client, you can use the following command:

  ```
  db2icrt db2inst1 –s client –u fencedID
  ```

DB2 UDB client instances are created when you want a workstation to connect to other database servers and you have no need for a local database on that workstation.

Related reference:

- “db2icrt - Create Instance” on page 260

Windows details when creating instances

When working with the Windows operating systems, the `db2icrt` command has the following optional parameters:

- –s InstType

  Allows different types of instances to be created. Valid instance types are: `ese`, `wse`, `client`, and `standalone`.

- –p:InstProf_Path

  This is an optional parameter to specify a different instance profile path. If you do not specify the path, the instance directory is created under the SQLLIB directory, and given the shared name DB2 concatenated to the instance name. Read and write permissions are automatically granted to everyone in the domain. Permissions can be changed to restrict access to the directory.

  If you do specify a different instance profile path, you must create a shared drive or directory. This will allow the opportunity for everyone in the domain to access the instance directory unless permissions have been changed.
- –u:username,password

  When creating a partitioned database environment, you must declare the domain/user account name and password of the DB2 Universal Database service.
- –r:base_port,end_port

  This is an optional parameter to specify the TCP/IP port range for the Fast Communications Manager (FCM). If you specify the TCP/IP port range, you must ensure that the port range is available on all machines in the partition database system.

The following example could be used, on DB2 Universal Database (DB2 UDB) Enterprise Server Edition for Windows:

```
db2icrt inst1 –s ese
   –p:\\machineA\db2mpp
   –u:<user account name>,<password> –r:9010,9015
```

Note: If you change the service account; that is, if you no longer use the default service created when the first instance was created during product installation, then you must grant the domain/user account name used to create the instance the following advanced rights:

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

The instance requires these user rights to access the shared drive, authenticate the user account, and run DB2 UDB as a Windows service. The "Lock page in memory" right is needed for Address Windowing Extensions (AWE) support.

Related reference:

- "db2icrt - Create Instance" on page 260

License management

The management of licenses for your DB2® Universal Database (DB2 UDB) 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, registered users, and current users for each of the installed products.

When the Control Center cannot be used, the `db2licm` Licensed Management Tool command performs basic license functions. With this command, you are able to add, remove, list, and modify licenses and policies installed on your local system.

Related reference:

- "db2licm - License Management Tool Command" in the *Command Reference*

Chapter 7. Creating a Database and Database Objects

Creating a database

Prerequisites:

You should have spent sufficient time designing the contents, layout, potential growth, and use of your database before you create it.

Procedure:

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

The following database privileges are automatically granted to PUBLIC: CREATETAB, BINDADD, CONNECT, IMPLICIT_SCHEMA, and SELECT privilege on the system catalog views.

To create a database using the Control Center:

1. Expand the object tree until you find the **Databases** folder.
2. Right-click the **Databases** folder, and select **Create —> Standard** or **Create —> With Automatic Maintenance** from the pop-up menu.
3. Follow the steps to complete this task.

The following command line processor command creates a database called `personl`, in the default location, with the associated comment ″Personnel DB for BSchiefer Co″.

```
CREATE DATABASE personl
   WITH "Personnel DB for BSchiefer Co"
```

When creating a database, you can also request to use the Configuration Advisor to assist the configuration of the database instead of accepting the default values for all of the configuration parameters. You can do this by using the AUTOCONFIGURE option on the `CREATE DATABASE` command:

```
CREATE DATABASE <database name>
   AUTOCONFIGURE
```

There are several options on the AUTOCONFIGURE clause. You cannot use the AUTOCONFIGURE clause when creating a database in a partitioned environment.

© Copyright IBM Corp. 1993-2004

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.

You have the ability to create a database in a different, possibly remote, database manager instance. In this type of environment you have the ability to perform instance-level administration against an instance other than your default instance, including remote instances.

Related concepts:

- "What to record in a database" in the *Administration Guide: Planning*
- "Multiple instances of the database manager" in the *Administration Guide: Implementation*
- "Database authorities" on page 24
- "Additional database design considerations" in the *Administration Guide: Planning*

Related reference:

- "CREATE DATABASE" on page 252

Defining initial table spaces

When a database is created, three table spaces are defined:

- SYSCATSPACE for the system catalog tables
- TEMPSPACE1 for system temporary tables created during database processing
- USERSPACE1 for user-defined tables and indexes

Note: When you first create a database no user temporary table space is created.

If you do not specify any table space parameters with the `CREATE DATABASE` command, the database manager creates these table spaces using system managed storage (SMS) directory containers. These directory containers are created in the subdirectory created for the database. The extent size for these table spaces is set to the default.

Prerequisites:

The database must be created and you must have the authority to create table spaces.

Procedure:

To define initial table spaces using the Control Center:

1. Expand the object tree until you see the **Databases** folder.
2. Right-click the **Databases** folder, and select **Create —> Standard** or **Create —> With Automatic Maintenance** from the pop-up menu.
3. Follow the steps to complete this task.

To define initial table spaces using the command line, enter:

```
CREATE DATABASE <name>
  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 may 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
  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 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 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.

Related concepts:

- "Definition of system catalog tables" on page 136
- "Table space design" on page 111

Related tasks:

- "Creating a table space" on page 137

Related reference:

- "CREATE DATABASE" on page 252

Definition of 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 may wish to limit the access to the system catalog views.

Related concepts:

- "User-defined functions" in the *SQL Reference, Volume 1*
- "Catalog views" in the *SQL Reference, Volume 1*
- "Functions overview" in the *SQL Reference, Volume 1*

Related tasks:

- "Securing the system catalog view" on page 192

Related reference:

- "Functions" in the *SQL Reference, Volume 1*

Definition of the 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 may not have been physically written to disk, are redone. These actions ensure the integrity of the database.

Related concepts:

- "Understanding recovery logs" on page 804

Binding utilities to the database

When a database is created, the database manager attempts to bind the utilities in db2ubind.lst to the database. This file is stored in the bnd subdirectory of your sqllib directory.

Binding a utility creates a *package*, which is an object that includes all the information needed to process specific SQL statements from a single source file.

Note: If you wish to use these utilities from a client, you must bind them explicitly.

If for some reason you need to bind or rebind the utilities to a database, issue the following commands using the command line processor:

```
connect to sample
bind @db2ubind.lst
```

Note: 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. In this example, `sample` is the name of the database.

Related tasks:

- "Creating a database" on page 133

Related reference:

- "BIND" on page 232

Creating a table space

Table spaces establish the relationship between the physical storage devices used by your database system and the logical containers or tables used to store data.

Prerequisites:

You must know the device or file names of the containers that you will reference when creating your table spaces. In addition, you must know the space associated with each device or file name that you will allocate to the table space.

Procedure:

Creating a table space within a database assigns containers to the table space and records its definitions and attributes in the database system catalog. You can then create tables within this table space.

To create a table space using the Control Center:

1. Expand the object tree until you see the **Table spaces** folder.
2. Right-click the **Table spaces** folder, and select **Create —> Table Space Using Wizard** from the pop-up menu.
3. Follow the steps in the wizard to complete your task.

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>)
```

The following SQL statement creates an SMS table space on Windows using three directories on three separate drives:

```
CREATE TABLESPACE RESOURCE
   MANAGED BY SYSTEM
   USING ('d:\acc_tbsp', 'e:\acc_tbsp', 'f:\acc_tbsp')
```

The following SQL statement creates a DMS table space using 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.

In addition, if part of the path name specified does not exist, the database manager creates it. If a subdirectory is created by the database manager, it may also be deleted by the database manager when the table space is dropped.

The assumption in the above examples 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
```

The following SQL statement creates a DMS table space on a UNIX-based 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 12.67
   TRANSFERRATE 0.18
```

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.

The following example creates a DMS table space on a database partition group called ODDGROUP in a UNIX partitioned 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
```

UNIX devices are classified into two categories: character serial devices and block-structured devices. For all file-system devices, it is normal to have a corresponding character serial device (or *raw* device) for each block device (or *cooked* device). The block-structured devices are typically designated by names similar to "hd0" or "fd0". The character serial devices are typically designated by names similar to "rhd0", "rfd0", or "rmt0". These character serial devices have faster access than block devices. The character serial device names should be used on the CREATE TABLESPACE command and not block device names.

The overhead and transfer rate help to determine the best access path to use when the SQL statement is compiled. The current defaults are:

- OVERHEAD 12.67 ms
- TRANSFERRATE 0.18 ms

DB2 UDB can greatly improve the performance of sequential I/O using the sequential prefetch facility, which uses parallel I/O.

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 UNIX-based 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 SQL 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 to prevent system catalog contention.

Note: The PREFETCHSIZE should be a multiple of the EXTENTSIZE. For example if the EXTENTSIZE is 10, the PREFETCHSIZE should be 20 or 30. You should use the following equation to set your prefetch size manually when creating a table space:

```
prefetch size = (number of containers) X (number of physical spindles per
                    container) X extent size
```

You should also consider letting DB2 UDB automatically determine the prefetch size.

Related concepts:

- "Table space design" on page 111
- "System managed space" on page 114
- "Database managed space" on page 116
- "Sequential prefetching" in the *Administration Guide: Performance*

Related tasks:

- "Enabling large page support in a 64-bit environment (AIX)" in the *Administration Guide: Planning*

Related reference:

- "ALTER TABLESPACE" on page 557
- "CREATE TABLESPACE" on page 648

Creating a schema

While organizing your data into tables, it may also be beneficial to group 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 this schema.

Prerequisites:

The database tables and other related objects that are to be grouped together must exist.

Restrictions:

This statement must be issued by a user with DBADM authority.

Schemas may also be implicitly created when a user has 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.

If users do not have IMPLICIT_SCHEMA authority, the only schema they can create is one that has the same name as their own authorization ID.

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 new schema name cannot already exist in the system catalogs and it cannot begin with ″SYS″.

Procedure:

If a user has SYSADM or DBADM authority, then the 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).

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.

To allow another user to access a table without entering a schema name as part of the qualification on the table name requires that a view be established for that user. The definition of the view would define the fully-qualified table name including the user's schema; the user would simply need to query using the view name. The view would be fully-qualified by the user's schema as part of the view definition.

To create a schema using the Control Center:

1. Expand the object tree until you see the **Schema** folder within a database.
2. Right-click the **Schema** folder, and click **Create**.
3. Complete the information for the new schema, and click **OK**.

To create a schema using the command line, enter:

```
CREATE SCHEMA <name> AUTHORIZATION <name>
```

The following is an example of a CREATE SCHEMA statement that creates a schema for an individual user with the authorization ID ″joe″:

```
CREATE SCHEMA joeschma AUTHORIZATION joe
```

Related concepts:

- "Grouping objects by schema" on page 122
- "Implicit schema authority (IMPLICIT_SCHEMA) considerations" on page 26
- "Schema privileges" on page 27

Related tasks:

- "Setting a schema" on page 141

Related reference:

- "CREATE SCHEMA" on page 588

Setting a schema

Once you have several schemas in existence, you may want to designate one as the default for schema for use by unqualified object references in dynamic SQL statements issued from within a specific DB2 connection.

Procedure:

Establishing a default schema is done by setting the special register CURRENT SCHEMA to the schema you wish to use as the default. Any user can set this special register: no authorization is required.

The following is an example of how to set the CURRENT SCHEMA special register:

```
SET CURRENT SCHEMA = 'SCHEMA01'
```

This statement can be used from within an application program or issued interactively. Once set, the value of the CURRENT SCHEMA special register is used as the qualifier (schema) for unqualified object references in dynamic SQL statements, with the exception of the CREATE SCHEMA statement where an unqualified reference to a database object exists.

The initial value of the CURRENT SCHEMA special register is equal to the authorization ID of the current session user.

Related concepts:

- "Schemas" on page 20

Related reference:

- "SET SCHEMA" on page 902
- "Reserved schema names and reserved words" in the *SQL Reference, Volume 1*
- "CURRENT SCHEMA" on page 801

Creating and populating a table

Tables are the main repository of data within databases. Creating the tables and entering data to fill the tables will occur when you are creating an new database.

Prerequisites:

You must take the time to design and organize the tables that will hold your data.

Procedure:

After you determine how to organize your data into tables, the next step is to create those tables, by using the CREATE TABLE statement. The table descriptions are stored in the system catalog of the database to which you are connected.

The CREATE TABLE statement gives the table a name, which is a qualified or unqualified identifier, and a definition for each of its columns. You can store each table in a separate table space, so that a table space contains only one table. If a table will be dropped and created often, it is more efficient to store it in a separate table space and then drop the table space instead of the table. You can also store many tables within a single table space. In a partitioned database environment, the table space chosen also defines the database partition group and the database partitions on which table data is stored.

The table does not contain any data at first. To add rows of data to it, use one of the following:

- The INSERT statement
- The LOAD or IMPORT commands
- The autoloader utility if working in a partitioned database environment

Adding data to a table can be done without logging the change. The NOT LOGGED INITIALLY clause on the CREATE TABLE statement prevents logging the change to the table. Any changes made to the table by an INSERT, DELETE,

UPDATE, CREATE INDEX, DROP INDEX, or ALTER TABLE operation in the same unit of work in which the table is created are not logged. Logging begins in subsequent units of work.

A table consists of one or more column definitions. A maximum of 500 columns can be defined for a table. Columns represent the attributes of an entity. The values in any column are all the same type of information.

Note: The maximum of 500 columns is true when using a 4 KB page size. The maximum is 1012 columns when using an 8 KB, 16 KB, or 32 KB page size.

A column definition includes a *column name*, *data type*, and any necessary *null attribute*, or default value (optionally chosen by the user).

The column name describes the information contained in the column and should be something that will be easily recognizable. It must be unique within the table; however, the same name can be used in other tables.

The data type of a column indicates the length of the values in it and the kind of data that is valid for it. The database manager uses character string, numeric, date, time and large object data types. Graphic string data types are only available for database environments using multi-byte character sets. In addition, columns can be defined with user-defined distinct types.

The default attribute specification indicates what value is to be used if no value is provided. The default value can be specified, or a system-defined default value used. Default values may be specified for columns with, and without, the null attribute specification.

The null attribute specification indicates whether or not a column can contain null values.

To create a table using the Control Center:

1. Expand the object tree until you see the **Tables** folder.
2. Right-click the **Tables** folder, and click **Create**.
3. Follow the steps in the wizard to complete your tasks.

To create a table using the command line, enter:

```
CREATE TABLE <NAME>
   (<column_name>  <data_type>  <null_attribute>)
   IN <TABLE_SPACE_NAME)
```

The following is an example of a CREATE TABLE statement that creates the EMPLOYEE table in the RESOURCE table space. This table is defined in the sample database:

```
CREATE TABLE EMPLOYEE
   (EMPNO     CHAR(6)     NOT NULL PRIMARY KEY,
    FIRSTNME  VARCHAR(12) NOT NULL,
    MIDINIT   CHAR(1)     NOT NULL WITH DEFAULT,
    LASTNAME  VARCHAR(15) NOT NULL,
    WORKDEPT  CHAR(3),
    PHONENO   CHAR(4),
    PHOTO     BLOB(10M)   NOT NULL)
IN RESOURCE
```

When creating a table, you can choose to have the columns of the table based on the attributes of a structured type. Such a table is called a “typed table”.

A typed table can be defined to inherit some of its columns from another typed table. Such a table is called a “subtable”, and the table from which it inherits is called its “supertable”. The combination of a typed table and all its subtables is called a “table hierarchy”. The topmost table in the table hierarchy (the one with no supertable) is called the “root table” of the hierarchy.

To declare a global temporary table, use the DECLARE GLOBAL TEMPORARY TABLE statement.

You can also create a table that is defined based on the result of a query. This type of table is called a *materialized query table*.

Related concepts:

- “Import Overview” in the *Data Movement Utilities Guide and Reference*
- “Load Overview” in the *Data Movement Utilities Guide and Reference*
- “Moving data across platforms - file format considerations” in the *Data Movement Utilities Guide and Reference*
- “User-defined type (UDT)” in the *Administration Guide: Implementation*

Related tasks:

- “Creating a materialized query table” in the *Administration Guide: Implementation*

Related reference:

- “CREATE TABLE” on page 591
- “INSERT” on page 724
- “DECLARE GLOBAL TEMPORARY TABLE statement” in the *SQL Reference, Volume 2*
- “IMPORT” on page 285
- “LOAD” on page 304

Large object (LOB) column considerations

Before creating a table that contains large object columns, you need to make the following decisions:

1. Do you want to log changes to LOB columns?

   If you do not want to log these changes, you must turn logging off by specifying the NOT LOGGED clause when you create the table:

```
CREATE TABLE EMPLOYEE
   (EMPNO     CHAR(6)     NOT NULL PRIMARY KEY,
    FIRSTNME  VARCHAR(12) NOT NULL,
    MIDINIT   CHAR(1)     NOT NULL WITH DEFAULT,
    LASTNAME  VARCHAR(15) NOT NULL,
    WORKDEPT  CHAR(3),
    PHONENO   CHAR(4),
    PHOTO     BLOB(10M)   NOT NULL  NOT LOGGED)
IN RESOURCE
```

   If the LOB column is larger than 1 GB, logging must be turned off. (As a rule of thumb, you may not want to log LOB columns larger than 10 MB.) As with other options specified on a column definition, the only way to change the logging option is to re-create the table.

Even if you choose not to log changes, LOB columns are *shadowed* to allow changes to be rolled back, whether the roll back is the result of a system generated error, or an application request. Shadowing is a recovery technique where current storage page contents are never overwritten. That is, old, unmodified pages are kept as "shadow" copies. These copies are discarded when they are no longer needed to support a transaction rollback.

Note: When recovering a database using the RESTORE and ROLLFORWARD commands, LOB data that was "NOT LOGGED"and was written since the last backup will be *replaced by binary zeros*.

2. Do you want to minimize the space required for the LOB column?

   You can make the LOB column as small as possible using the COMPACT clause on the CREATE TABLE statement. For example:

```
CREATE TABLE EMPLOYEE
   (EMPNO     CHAR(6)     NOT NULL PRIMARY KEY,
    FIRSTNME  VARCHAR(12) NOT NULL,
    MIDINIT   CHAR(1)     NOT NULL WITH DEFAULT,
    LASTNAME  VARCHAR(15) NOT NULL,
    WORKDEPT  CHAR(3),
    PHONENO   CHAR(4),
    PHOTO     BLOB(10M)   NOT NULL  NOT LOGGED  COMPACT)
IN RESOURCE
```

   There is a *performance cost* when appending to a table with a compact LOB column, particularly if the size of LOB values are increased (because of storage adjustments that must be made).

   On platforms where sparse file allocation is not supported and where LOBs are placed in SMS table spaces, consider using the COMPACT clause. Sparse file allocation has to do with how physical disk space is used by an operating system. An operating system that supports sparse file allocation does not use as much physical disk space to store LOBs as compared to an operating system not supporting sparse file allocation. The COMPACT option allows for even greater physical disk space "savings" regardless of the support of sparse file allocation. Because you can get some physical disk space savings when using COMPACT, you should consider using COMPACT if your operating system does not support sparse file allocation.

   Note: DB2® system catalogs use LOB columns and may take up more space than in previous versions.

3. Do you want better performance for LOB columns, including those LOB columns in the DB2 system catalogs?

   There are large object (LOB) columns in the catalog tables. LOB data is not kept in the buffer pool with other data but is read from disk each time it is needed. Reading from disk slows down the performance of DB2 where the LOB columns of the catalogs are involved. Since a file system usually has its own place for storing (or caching) data, using a SMS table space, or a DMS table space built on file containers, make avoidance of I/O possible when the LOB has previously been referenced.

Related concepts:

- "Space requirements for large object data" on page 107

Related reference:

- "CREATE TABLE" on page 591
- "Large objects (LOBs)" in the *SQL Reference, Volume 1*

Creating a view

Views are derived from one or more base tables, nicknames, or views, and can be used interchangeably with base tables when retrieving data. When changes are made to the data shown in a view, the data is changed in the table itself.

A view can be created to limit access to sensitive data, while allowing more general access to other data.

When inserting into a view where the SELECT-list of the view definition directly or indirectly includes the name of an identity column of a base table, the same rules apply as if the INSERT statement directly referenced the identity column of the base table.

In addition to using views as described above, a view can also be used to:

- Alter a table without affecting application programs. This can happen by creating a view based on an underlying table. Applications that use the underlying table are not affected by the creation of the new view. New applications can use the created view for different purposes than those applications that use the underlying table.
- Sum the values in a column, select the maximum values, or average the values.
- Provide access to information in one or more data sources. You can reference nicknames within the CREATE VIEW statement and create multi-location/global views (the view could join information in multiple data sources located on different systems).

  When you create a view that references nicknames using standard CREATE VIEW syntax, you will see a warning alerting you to the fact that the authentication ID of view users will be used to access the underlying object or objects at data sources instead of the view creator authentication ID. Use the FEDERATED keyword to suppress this warning.

An alternative to creating a view is to use a nested or common table expression to reduce catalog lookup and improve performance.

Prerequisites:

The base table, nickname, or view on which the view is to be based must already exist before the view can be created.

Restrictions:

You can create a view that uses a UDF in its definition. However, to update this view so that it contains the latest functions, you must drop it and then re-create it. If a view is dependent on a UDF, that function cannot be dropped.

The following SQL statement creates a view with a function in its definition:

```
CREATE VIEW EMPLOYEE_PENSION (NAME, PENSION)
  AS SELECT NAME, PENSION(HIREDATE,BIRTHDATE,SALARY,BONUS)
  FROM EMPLOYEE
```

The UDF function PENSION calculates the current pension an employee is eligible to receive, based on a formula involving their HIREDATE, BIRTHDATE, SALARY, and BONUS.

Procedure:

To create a view using the Control Center:

1. Expand the object tree until you see the **Views** folder.
2. Right-click the **Views** folder, and select **Create** from the pop-up menu.
3. Complete the information, and click **Ok**.

To create a view using the command line, enter:

```
CREATE VIEW <name> (<column>, <column>, <column>)
   SELECT <column_names> FROM <table_name>
   WITH CHECK OPTION
```

For example, the EMPLOYEE table may have salary information in it, which should not be made available to everyone. The employee's phone number, however, should be generally accessible. In this case, a view could be created from the LASTNAME and PHONENO columns only. Access to the view could be granted to PUBLIC, while access to the entire EMPLOYEE table could be restricted to those who have the authorization to see salary information.

With a view, you can make a subset of table data available to an application program and validate data that is to be inserted or updated. A view can have column names that are different from the names of corresponding columns in the original tables.

The use of views provides flexibility in the way your programs and end-user queries can look at the table data.

The following SQL statement creates a view on the EMPLOYEE table that lists all employees in Department A00 with their employee and telephone numbers:

```
CREATE VIEW EMP_VIEW (DA00NAME, DA00NUM, PHONENO)
   AS SELECT LASTNAME, EMPNO, PHONENO FROM EMPLOYEE
   WHERE WORKDEPT = 'A00'
   WITH CHECK OPTION
```

The first line of this statement names the view and defines its columns. The name EMP_VIEW must be unique within its schema in SYSCAT.TABLES. The view name appears as a table name although it contains no data. The view will have three columns called DA00NAME, DA00NUM, and PHONENO, which correspond to the columns LASTNAME, EMPNO, and PHONENO from the EMPLOYEE table. The column names listed apply one-to-one to the select list of the SELECT statement. If column names are not specified, the view uses the same names as the columns of the result table of the SELECT statement.

The second line is a SELECT statement that describes which values are to be selected from the database. It may include the clauses ALL, DISTINCT, FROM, WHERE, GROUP BY, and HAVING. The name or names of the data objects from which to select columns for the view must follow the FROM clause.

The WITH CHECK OPTION clause indicates that any updated or inserted row to the view must be checked against the view definition, and rejected if it does not conform. This enhances data integrity but requires additional processing. If this clause is omitted, inserts and updates are not checked against the view definition.

The following SQL statement creates the same view on the EMPLOYEE table using the SELECT AS clause:

```
CREATE VIEW EMP_VIEW
   SELECT LASTNAME AS DA00NAME,
          EMPNO AS DA00NUM,
          PHONENO
   FROM EMPLOYEE
   WHERE WORKDEPT = 'A00'
   WITH CHECK OPTION
```

Related concepts:

- "Views" in the *SQL Reference, Volume 1*
- "Table and view privileges" on page 28
- "Controlling access to data with views" on page 49
- "Using triggers to update view contents" in the *Administration Guide: Implementation*

Related tasks:

- "Creating a typed view" in the *Administration Guide: Implementation*
- "Removing rows from a table or view" in the *Administration Guide: Implementation*
- "Altering or dropping a view" in the *Administration Guide: Implementation*
- "Recovering inoperative views" in the *Administration Guide: Implementation*

Related reference:

- "CREATE VIEW" on page 656
- "INSERT" on page 724

Creating an index

An *index* is a set of one or more keys, each pointing to rows in a table. An index allows more efficient access to rows in a table by creating a direct path to the data through pointers.

Procedure:

Performance Tip: If you are going to carry out the following series of tasks:

1. Create Table
2. Load Table
3. Create Index (without the COLLECT STATISTICS option)
4. Perform RUNSTATS

Or, if you are going to carry out the following series of tasks:

1. Create Table
2. Load Table
3. Create Index (with the COLLECT STATISTICS option)

then you should consider ordering the execution of tasks in the following way:

1. Create the table
2. Create the index
3. Load the table with the `statistics yes` option requested.

Indexes are maintained after they are created. Subsequently, when application programs use a key value to randomly access and process rows in a table, the index based on that key value can be used to access rows directly. This is important, because the physical storage of rows in a base table is not ordered.

When creating the table, you can choose to create a multi-dimensional clustering (MDC) table. By creating this type of table, block indexes are created. Regular indexes point to individual rows; block indexes point to blocks or extents of data, and are much smaller than regular indexes. Block indexes are stored, along with regular indexes, in the same table space.

When a row is inserted, unless there is a clustering index defined, the row is placed in the most convenient storage location that can accommodate it. When searching for rows of a table that meet a particular selection condition and the table has no indexes, the entire table is scanned. An index optimizes data retrieval without performing a lengthy sequential search.

The data for your indexes can be stored in the same table space as your table data, or in a separate table space containing index data. The table space used to store the index data is determined when the table is created.

To create an index using the Control Center:

1. Expand the object tree until you see the **Indexes** folder.
2. Right-click the **Indexes** folder, and select **Create —> Index Using Wizard** from the pop-up menu.
3. Follow the steps in the wizard to complete your task.

To create an index using the command line, enter:

```
CREATE INDEX <name> ON <table_name> (<column_name>)
```

Related concepts:

- "Optimizing load performance" in the *Data Movement Utilities Guide and Reference*
- "Using an index" on page 149
- "Options on the CREATE INDEX statement" on page 150
- "Index privileges" on page 31

Related tasks:

- "Renaming an existing table or index" in the *Administration Guide: Implementation*
- "Dropping an index, index extension, or an index specification" in the *Administration Guide: Implementation*

Related reference:

- "CREATE INDEX" on page 575

Using an index

An index is never directly used by an application program. The decision on whether to use an index and which of the potentially available indexes to use is the responsibility of the optimizer.

The best index on a table is one that:

- Uses high-speed disks
- Is highly-clustered
- Is made up of only a few narrow columns
- Uses columns with high cardinality

Related concepts:

- "Index planning tips" in the *Administration Guide: Performance*
- "Index performance tips" in the *Administration Guide: Performance*
- "Data access through index scans" in the *Administration Guide: Performance*
- "Table and index management for standard tables" in the *Administration Guide: Performance*
- "Table and index management for MDC tables" in the *Administration Guide: Performance*

Options on the CREATE INDEX statement

You can create an index that will allow duplicates (a non-unique index) to enable efficient retrieval by columns other than the primary key, and allow duplicate values to exist in the indexed column or columns.

The following SQL statement creates a non-unique index called LNAME from the LASTNAME column on the EMPLOYEE table, sorted in ascending order:

```
CREATE INDEX LNAME ON EMPLOYEE (LASTNAME ASC)
```

The following SQL statement creates a unique index on the phone number column:

```
CREATE UNIQUE INDEX PH ON EMPLOYEE (PHONENO DESC)
```

A unique index ensures that no duplicate values exist in the indexed column or columns. The constraint is enforced at the end of the SQL statement that updates rows or inserts new rows. This type of index cannot be created if the set of one or more columns already has duplicate values.

The keyword ASC puts the index entries in ascending order by column, while DESC puts them in descending order by column. The default is ascending order.

You can create a unique index on two columns, one of which is an include column. The primary key is defined on the column that is not the include column. Both of them are shown in the catalog as primary keys on the same table. Normally there is only one primary key per table.

The INCLUDE clause 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. The included columns may improve the performance of some queries through index-only access. The columns must be distinct from the columns used to enforce uniqueness (otherwise you will receive error message SQLSTATE 42711). The limits for the number of columns and sum of the length attributes apply to all of the columns in the unique key and in the index.

A check is performed to determine if an existing index matches the primary key definition (ignoring any INCLUDE columns in the index). An index definition matches if it identifies the same set of columns without regard to the order of the columns or the direction (either ascending or descending) specifications. If a matching index definition is found, the description of the index is changed to

indicate that it is the primary index, as required by the system, and it is changed to unique (after ensuring uniqueness) if it was a non-unique index.

This is why it is possible to have more than one primary key on the same table as indicated in the catalog.

When working with a structured type, it might be necessary to create user-defined index types. This requires a means of defining index maintenance, index search, and index exploitation functions.

The following SQL statement creates a clustering index called INDEX1 on the LASTNAME column of the EMPLOYEE table:

```
CREATE INDEX INDEX1 ON EMPLOYEE (LASTNAME) CLUSTER
```

To use the internal storage of the database effectively, use clustering indexes with the PCTFREE parameter associated with the ALTER TABLE statement so that new data can be inserted on the correct pages. When data is inserted on the correct pages, clustering order is maintained. Typically, the greater the INSERT activity on the table, the larger the PCTFREE value (on the table) that will be needed in order to maintain clustering. Since this index determines the order by which the data is laid out on physical pages, only one clustering index can be defined for any particular table.

If the index key values of these new rows are always new high key values for example, then the clustering attribute of the table will try to place them at the end of the table. Having free space in other pages will do little to preserve clustering. In this case, placing the table in append mode may be a better choice than a clustering index and altering the table to have a large PCTFREE value. You can place the table in append mode by issuing: ALTER TABLE APPEND ON.

The above discussion also applies to new ″overflow″ rows that result from UPDATEs that increase the size of a row.

A single index created using the ALLOW REVERSE SCANS parameter on the CREATE INDEX statement can be scanned in a forward or a backward direction. That is, such indexes support scans in the direction defined when the index was created and scans in the opposite or reverse direction. The statement could look something like:

```
CREATE INDEX iname ON tname (cname DESC) ALLOW REVERSE SCANS
```

In this case, the index (`iname`) is formed based on descending values (`DESC`) in the given column (`cname`). By allowing reverse scans, although the index on the column is defined for scans in descending order, a scan can be done in ascending order (reverse order). The actual use of the index in both directions is not controlled by you but by the optimizer when creating and considering access plans.

The MINPCTUSED clause of the CREATE INDEX statement specifies the threshold for the minimum amount of used space on an index leaf page. If this clause is used, online index defragmentation is enabled for this index. Once enabled, the following considerations are used to determine if an online index defragmentation takes place: After a key is physically removed from a leaf page of this index and a percentage of used space on the page is less than the specified threshold value, the neighboring index leaf pages are checked to determine if the keys on the two leaf pages can be merged into a single index leaf page.

For example, the following SQL statement creates an index with online index defragmentation enabled:

```
CREATE INDEX LASTN ON EMPLOYEE (LASTNAME) MINPCTUSED 20
```

When a key is physically removed from an index page of this index, if the remaining keys on the index page take up twenty percent or less space on the index page, then an attempt is made to delete an index page by merging the keys of this index page with those of a neighboring index page. If the combined keys can all fit on a single page, this merge is performed and one of the index pages is deleted.

The CREATE INDEX statement allows you to create the index while, at the same time, allowing read and write access to the underlying table and any previously existing indexes. To restrict access to the table while creating the index, use the LOCK TABLE statement to lock the table before creating the index. The new index is created by scanning the underlying table. Any changes made to the table while the index is being created are logged. Once the new index is created, the changes are applied to the index. To apply the logged changes more quickly during the index creation, a separate copy of the changes is maintained in memory buffer space, which is allocated on demand from the utility heap. This allows the index creation to process the changes by directly reading from memory first, and reading through the logs, if necessary, at a much later time. Once all the changes have been applied to the index, the table is quiesced while the new index is made visible.

When creating a unique index, ensure that there are no duplicate keys in the table and that the concurrent inserts during index creation are not going to introduce duplicate keys. Index creation uses a deferred unique scheme to detect duplicate keys, and therefore no duplicate keys will be detected until the very end of index creation, at which point the index creation will fail because of the duplicate keys.

The PCTFREE clause of the CREATE INDEX statement specifies the percentage of each index page to leave as free space when the index is built. Leaving more free space on the index pages will result in fewer page splits. This will reduce the need to reorganize the table in order to regain sequential index pages which increases prefetching. And prefetching is one important component that may improve performance. Again, if there are always high key values, then you will want to consider lowering the value of the PCTFREE clause of the CREATE INDEX statement. In this way there will be limited wasted space reserved on each index page.

The LEVEL2 PCTFREE clause directs the system to preserve a specified percentage of free space on each page in the second level of an index. You specify a percentage of free space when the index is created to accommodate future insertions and updates. The second level is the level immediately above the leaf level. The default is to preserve a minimum of 10 and the PCTFREE value in all non-leaf pages. The LEVEL2 PCTFREE parameter allows the default to be overwritten; if you use the LEVEL2 PCTFREE integer option in the CREATE INDEX statement, the integer percent of free space is left on level 2 intermediate pages. A minimum of 10 and the integer percent of free space is left on level 3 and higher intermediate pages. By leaving more free space on the second level, the number of page splits that occur at the second level of the index is reduced.

The PAGE SPLIT SYMMETRIC, PAGE SPLIT HIGH, and PAGE SPLIT LOW clauses allow a choice in the page split behavior when inserting into an index.

The PAGE SPLIT SYMMETRIC clause is a default page split behavior that splits roughly in the middle of an index page. Using this default behavior is best when the insertion into an index is random or does not follow one of the patterns that are addressed by the PAGE SPLIT HIGH and PAGE SPLIT LOW clauses.

The PAGE SPLIT HIGH behavior is useful when there are ever increasing ranges in the index. Increasing ranges in the index may occur when:

- There is an index with multiple key parts and there are many values (multiple index pages worth) where all except the last key part have the same value
- All inserts into the table would consist of a new value which has the same value as existing keys for all but the last key part
- The last key part of the inserted value is larger than that of the existing keys

For example, if we have the following key values in the index;

```
(1,1),(1,2),(1,3), ... (1,n),
(2,1),(2,2),(2,3), ... (2,n),
...
(m,1),(m,2),(m,3), ...(m,n)
```

then the next key to be inserted would have the value (x,y) where 1 <= x <= m and y > n. If the insertions follow such a pattern, the PAGE SPLIT HIGH clause can be used so that page splits do not result in many pages that are fifty percent empty.

Similarly, PAGE SPLIT LOW can be used when there are ever-decreasing ranges in the index, to avoid leaving pages 50 percent empty.

Note: If you want to add a primary or unique key, and you want the underlying index to use SPLIT HIGH, SPLIT LOW, PCTFREE, LEVEL2 PCTFREE, MINPCTUSED, CLUSTER, or ALLOW REVERSE SCANS you must first create an index specifying the desired keys and parameters. Then use an ALTER TABLE statement to add the primary or unique key. The ALTER TABLE statement will pick up and reuse the index that you have already created.

You can collect index statistics as part of the creation of the index. At the time when you use the CREATE INDEX statement, the key value statistics and the physical statistics are available for use. By collecting the index statistics as part of the CREATE INDEX statement, you will not need to run the RUNSTATS utility immediately following the completion of the CREATE INDEX statement.

For example, the following SQL statement will collect basic index statistics as part of the creation of an index:

```
CREATE INDEX IDX1 ON TABL1 (COL1) COLLECT STATISTICS
```

If you have a replicated summary table, its base table (or tables) must have a unique index, and the index key columns must be used in the query that defines the replicated summary table.

For intra-partition parallelism, create index performance is improved by using multiple processors for the scanning and sorting of data that is performed during index creation. The use of multiple processors is enabled by setting *intra_parallel* to YES(1) or ANY(-1). The number of processors used during index creation is

determined by the system and is not affected by the configuration parameters *dft_degree* or *max_querydegree*, by the application runtime degree, or by the SQL statement compilation degree.

In multiple partition databases, unique indexes must be defined as supersets of the partitioning key.

Related concepts:

- "Index performance tips" in the *Administration Guide: Performance*
- "Index reorganization" in the *Administration Guide: Performance*
- "Table and index management for standard tables" in the *Administration Guide: Performance*
- "Online index defragmentation" in the *Administration Guide: Performance*
- "Table and index management for MDC tables" in the *Administration Guide: Performance*

Related tasks:

- "Changing table attributes" in the *Administration Guide: Implementation*

Related reference:

- "max_querydegree - Maximum query degree of parallelism configuration parameter" in the *Administration Guide: Performance*
- "intra_parallel - Enable intra-partition parallelism configuration parameter" in the *Administration Guide: Performance*
- "dft_degree - Default degree configuration parameter" in the *Administration Guide: Performance*
- "CREATE INDEX" on page 575

Chapter 8. Concurrency, Isolation Levels, and Locking

Deadlocks between applications

With multiple applications working with data from the database there are opportunities for a deadlock to occur between two or more applications.

A deadlock is created when one application is waiting for another application to release a lock on data. Each of the waiting applications is locking data needed by another application. Mutual waiting for the other application to release a lock on held data leads to a deadlock. The applications can wait forever until one application releases the lock on the held data.

A deadlock is illustrated in the following figure.

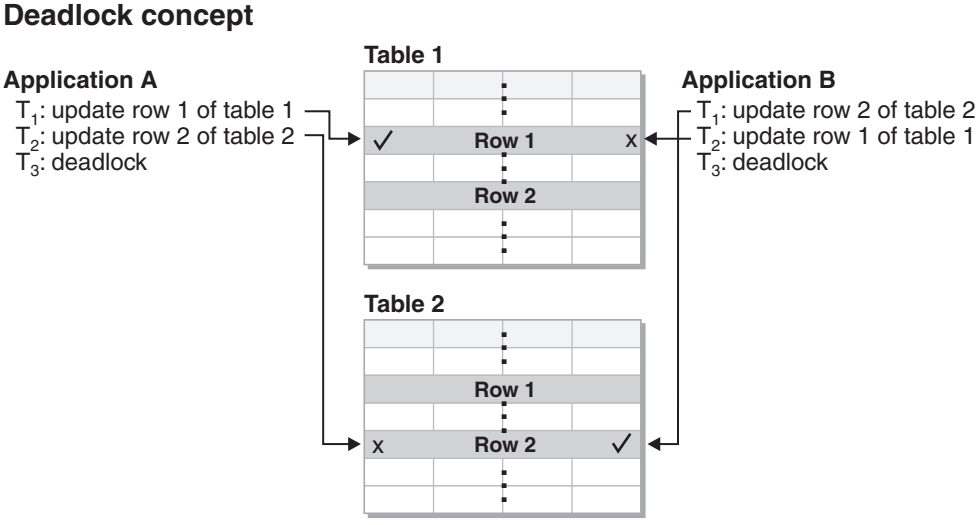


Figure 11. Deadlock detector

Because applications do not voluntarily release locks on data that they need, a deadlock detector process is required to break deadlocks and allow application processing to continue. As its name suggests, the deadlock detector monitors the information about agents waiting on locks. The deadlock detector arbitrarily selects one of the applications in the deadlock and releases the locks currently held by that "volunteered" application. By releasing the locks of that application, the data

© Copyright IBM Corp. 1993-2004

required by other waiting applications is made available for use. The waiting applications can then access the data required to complete transactions.

Related concepts:

- "Locks and performance" on page 166
- "DB2 architecture and process overview" on page 3

Concurrency Control and Isolation Levels

Concurrency issues

Because many users access and change data in a relational database, the database manager must be able both to allow users to make these changes and to ensure 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 from the database and both calculate new values for one of its columns based on the data these applications read. If A updates the row with its new value and B then also updates the row, the update performed by A is lost.
- **Access to uncommitted data**. Application A might update a value in the database, and application B might read that value before it was committed. Then, if the value of A is not later committed, but backed out, the calculations performed by B are based on uncommitted (and presumably invalid) data.
- **Nonrepeatable reads**. Some applications involve the following sequence of events: application A reads a row from the database, then goes on to process other SQL requests. In the meantime, application B either modifies or deletes the row and commits the change. Later, if application A attempts to read the original row again, it receives the modified row or discovers that the original row has been deleted.
- **Phantom Read Phenomenon**. The phantom read phenomenon occurs when:
  1. Your application executes a query that reads a set of rows based on some search criterion.
  2. Another application inserts new data or updates existing data that would satisfy your application's query.
  3. Your application repeats the query from step 1 (within the same unit of work).

     Some additional ("phantom") rows are returned as part of the result set that were not returned when the query was initially executed (step 1).

Note: Declared temporary tables have no concurrency issues because they are available only to the application that declared them. This type of table only exists from the time that the application declares it until the application completes or disconnects.

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) or databases in a single statement. To reference the data sources, which consist of a DBMS and data, DB2® uses *nicknames*. Nicknames are aliases for objects in other database managers. In a federated system, DB2 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, with location transparency if information about tables and views is moved, references to that information through nicknames can be updated without changing applications that request the information. When an application accesses data through nicknames, DB2 relies on the concurrency control protocols of data-source database managers to ensure isolation levels. Although DB2 tries to match the requested level of isolation at the data source with a logical equivalent, results may vary depending on data source capabilities.

Related concepts:

- "Performance impact of isolation levels" on page 157

Related tasks:

- "Specifying the isolation level" on page 160

Related reference:

- "locklist - Maximum storage for lock list configuration parameter" in the *Administration Guide: Performance*
- "maxlocks - Maximum percent of lock list before escalation" on page 794

Performance impact of isolation levels

An *isolation level* determines how data is locked or isolated from other processes while the data is being accessed. The isolation level will be in effect for the duration of the unit of work. Applications that use a cursor declared with a DECLARE CURSOR statement using the WITH HOLD clause will keep the chosen isolation level for the duration of the unit of work in which the OPEN CURSOR was performed. DB2® supports the following isolation levels:

- Repeatable Read
- Read Stability
- Cursor Stability
- Uncommitted Read.

Note: Some host database servers support the *no commit* isolation level. On other databases, this isolation level behaves like the uncommitted read isolation level.

Detailed explanations for each of the isolation levels follows in decreasing order of performance impact, but in increasing order of care required when accessing and updating data.

Repeatable Read

Repeatable Read (RR) locks all the rows an application references within a unit of work. Using Repeatable Read, a SELECT statement issued by an application twice within the same unit of work in which the cursor was opened, gives the same result each time. With Repeatable Read, lost updates, access to uncommitted data, and phantom rows are not possible.

The Repeatable Read application can retrieve and operate on the rows as many times as needed until the unit of work completes. However, no other applications can update, delete, or insert a row that would affect the result table, until the unit of work completes. Repeatable Read applications cannot see uncommitted changes of other applications.

With Repeatable Read, every row that is referenced is locked, not just the rows that are retrieved. Appropriate locking is performed so that another application cannot insert or update a row that would be added to the list of rows referenced by your query, if the query was re-executed. This prevents phantom rows from occurring. For example, if you scan 10 000 rows and apply predicates to them, locks are held on all 10 000 rows, even though only 10 rows qualify.

Note: The Repeatable Read isolation level ensures that all returned data remains unchanged until the time the application *sees* the data, even when temporary tables or row blocking are used.

Since Repeatable Read may acquire and hold a considerable number of locks, these locks may exceed the number of locks available as a result of the *locklist* and *maxlocks* configuration parameters. In order to avoid lock escalation, the optimizer may elect to acquire a single table-level lock immediately for an index scan, if it believes that lock escalation is very likely to occur. This functions as though the database manager has issued a LOCK TABLE statement on your behalf. If you do not want a table-level lock to be obtained ensure that enough locks are available to the transaction or use the Read Stability isolation level.

Read Stability

Read Stability (RS) locks only those rows that an application retrieves within a unit of work. It ensures that any qualifying row read during a unit of work is not changed by other application processes until the unit of work completes, and that any row changed by another application process is not read until the change is committed by that process. That is, "nonrepeatable read" behavior is **not** possible.

Unlike repeatable read, with Read Stability, if your application issues the same query more than once, you may see additional *phantom* rows (the *phantom read phenomenon*). Recalling the example of scanning 10 000 rows, Read Stability only locks the rows that qualify. Thus, with Read Stability, only 10 rows are retrieved, and a lock is held only on those ten rows. Contrast this with Repeatable Read, where in this example, locks would be held on all 10 000 rows. The locks that are held can be share, next share, update, or exclusive locks.

Note: The Read Stability isolation level ensures that all returned data remains unchanged until the time the application *sees* the data, even when temporary tables or row blocking are used.

One of the objectives of the Read Stability isolation level is to provide both a high degree of concurrency as well as a stable view of the data. To assist in achieving this objective, the optimizer ensures that table level locks are not obtained until lock escalation occurs.

The Read Stability isolation level is best for applications that include all of the following:
- Operate in a concurrent environment
- Require qualifying rows to remain stable for the duration of the unit of work
- Do not issue the same query more than once within the unit of work, or do not require that the query get the same answer when issued more than once in the same unit of work.

Cursor Stability

Cursor Stability (CS) locks any row accessed by a transaction of an application while the cursor is positioned on the row. This lock remains in effect until the next row is fetched or the transaction is terminated. However, if any data on a row is changed, the lock must be held until the change is committed to the database.

No other applications can update or delete a row that a Cursor Stability application has retrieved while any updatable cursor is positioned on the row. Cursor Stability applications cannot see uncommitted changes of other applications.

Recalling the example of scanning 10 000 rows, if you use Cursor Stability, you will only have a lock on the row under your current cursor position. The lock is removed when you move off that row (unless you update that row).

With Cursor Stability, both nonrepeatable read and the phantom read phenomenon are possible. Cursor Stability is the default isolation level and should be used when you want the maximum concurrency while seeing only committed rows from other applications.

Uncommitted Read

Uncommitted Read (UR) allows an application to access uncommitted changes of other transactions. The application also does not lock other applications out of the row it is reading, unless the other application attempts to drop or alter the table. Uncommitted Read works differently for read-only and updatable cursors.

Read-only cursors can access most uncommitted changes of other transactions. However, 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.

Note: Cursors that are updatable operating under the Uncommitted Read isolation level will behave as if the isolation level was cursor stability.

When it runs a program using isolation level UR, an application can use isolation level CS. This happens because the cursors used in the application program are ambiguous. The ambiguous cursors can be escalated to isolation level CS because of a BLOCKING option. The default for the BLOCKING option is UNAMBIG. This means that ambiguous cursors are treated as updatable and the escalation of the isolation level to CS occurs. To prevent this escalation, you have the following two choices:

- Modify the cursors in the application program so that they are unambiguous. Change the SELECT statements to include the FOR READ ONLY clause.
- Leave cursors ambiguous in the application program, but precompile the program or bind it with the BLOCKING ALL option to allow any ambiguous cursors to be treated as read-only when the program is run.

As in the example given for Repeatable Read, of scanning 10 000 rows, if you use Uncommitted Read, you do not acquire any row locks.

With Uncommitted Read, both nonrepeatable read behavior and the phantom read phenomenon are possible. The Uncommitted Read isolation level is most

commonly used for queries on read-only tables, or if you are executing only select statements and you do not care whether you see uncommitted data from other applications.

Summary of isolation levels

The following table summarizes the different isolation levels in terms of their undesirable effects.

Table 18. Summary of isolation levels

Isolation Level	Access to uncommitted data	Nonrepeatable reads	Phantom read phenomenon
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 table below provides a simple heuristic to help you choose an initial isolation level for your applications. Consider this table a starting point, and refer to the previous discussions of the various levels for factors that might make another isolation level more appropriate.

Table 19. 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

Choosing the appropriate isolation level for an application is very important to avoid the phenomena that are intolerable for that application. The isolation level affects not only the degree of isolation among applications but also the performance characteristics of an individual application since the CPU and memory resources that are required to obtain and free locks vary with the isolation level. The potential for deadlock situations also varies with the isolation level.

Related concepts:

- "Concurrency issues" on page 156

Related tasks:

- "Specifying the isolation level" on page 160

Specifying the isolation level

Because the isolation level determines how data is locked and 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. The isolation level that you specify is in effect for the duration of the unit of work.

The isolation level can be specified in several different ways. The following heuristics are used in determining which isolation level will be used in compiling an SQL statement:

Static SQL:

- If an isolation clause is specified in the statement, then the value of that clause is used.
- If no isolation clause is specifed in the statement, then the isolation level used is the one specified for the package at the time when the package was bound to the database.

Dynamic SQL:

- If an isolation clause is specified in the statement, then the value of that clause is used.
- If no isolation clause is specifed in the statement, and a SET CURRENT ISOLATION statement has been issued within the current session, then the value of the CURRENT ISOLATION special register is used.
- If no isolation clause is specifed in the statement, and no SET CURRENT ISOLATION statement has been issued within the current session, then the isolation level used is the one specified for the package at the time when the package was bound to the database.

Note: Many commercially written applications provide a method for choosing the isolation level. Refer to the application documentation for information.

Procedure:

To specify the isolation level:

1. **At precompile or bind time:**

   For an application written in a supported compiled language, use the ISOLATION option of the command line processor PREP or BIND commands. You can also use the PREP or BIND APIs 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 used during precompilation.
   - If you do not specify an isolation level, the default of cursor stability is used.

   Note: To determine the isolation level of a package, execute the following query:

   ```
   SELECT ISOLATION FROM SYSCAT.PACKAGES
     WHERE PKGNAME = 'XXXXXXXX'
     AND PKGSCHEMA = 'YYYYYYYY'
   ```

   where *XXXXXXXX* is the name of the package and *YYYYYYYY* is the schema name of the package. Both of these names must be in all capital letters.

2. **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 packages are also bound to the database when a database is created.

   REXX and the command line processor connect to a database using a default isolation level of cursor stability. Changing to a different isolation level does not change the connection state. It must be executed in the CONNECTABLE AND UNCONNECTED state or in the IMPLICITLY CONNECTABLE state.

   To verify the isolation level in use by a REXX application, check the value of the SQLISL REXX variable. The value is updated every time the CHANGE SQLISL command is executed.

3. **At the statement level:**

   Use the WITH clause. The statement-level isolation level overrides the isolation level specified for the package in which the statement appears.

   You can specify an isolation level for the following SQL statements:

   - SELECT
   - SELECT INTO
   - Searched DELETE
   - INSERT
   - Searched UPDATE
   - DECLARE CURSOR

   The following conditions apply to isolation levels specified for statements:

   - The WITH clause cannot be used on subqueries
   - The WITH UR option applies only to read-only operations. In other cases, the statement is automatically changed from UR to CS.

4. **From CLI or ODBC at runtime:**

   Use the CHANGE ISOLATION LEVEL command. For DB2 Call Level Interface (DB2 CLI), you can change the isolation level as part of the DB2 CLI configuration. At runtime, 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*. You can also use the TXNISOLATION keyword in the db2cli.ini file .

5. **When working with JDBC or SQLJ at run time:**

   Note: JDBC and SQLJ are implemented with CLI on DB2, which means the db2cli.ini settings might affect what is written and run using JDBC and SQLJ.

   Use the setTransactionIsolation method in the java.sql interface connection.

   In SQLJ, you run the db2profc SQLJ optimizer to create a package. The options that you can specify for this package include its isolation level.

6. **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 level of isolation for any dynamic SQL statements issued within the current session. Once set, the CURRENT ISOLATION special register provides the isolation level for any subsequent dynamic SQL statement compiled within the session, regardless of the package issuing the statement. This isolation level will apply until the session is ended or until a SET CURRENT ISOLATION statement is issued with the RESET option.

Related concepts:

- "Concurrency issues" on page 156

Related reference:

- "SQLSetConnectAttr function (CLI) - Set connection attributes" in the *CLI Guide and Reference, Volume 2*
- "CONNECT (Type 1)" on page 887
- "Statement attributes (CLI) list" in the *CLI Guide and Reference, Volume 2*

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, table blocks, or table rows. A *lock* associates a database manager resource with an application, called the *lock owner*, to control how other applications can access the same resource.

Although most locking occurs on tables, when a buffer pool is created, altered, or dropped, a buffer pool lock is set. The mode used with this lock is EXCLUSIVE (X). You may encounter this lock when a snapshot is taken using the Command Line Processor (CLP). When viewing the snapshot, you will see that the lock name used is the identifier (ID) of the buffer pool itself.

The database manager uses record-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. 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 used when the table is accessed. The choices are either ROW for row locks, or TABLE for table locks. Use ALTER TABLE... LOCKSIZE TABLE for read-only tables. This reduces the number of locks required by database activity.
- The amount of memory devoted to locking. The amount of memory devoted to locking is controlled by the `locklist` database configuration parameter. If the lock list fills, performance can degrade due to lock escalations and reduced concurrency on shared objects in the database. If lock escalations occur frequently, increase the value of either `locklist` or `maxlocks`, or both.

Ensure that all transactions COMMIT frequently to free held locks.

In general, record-level locking is used unless one of the following is the case:

- The isolation level chosen is uncommitted read (UR).
- The isolation level chosen is repeatable read (RR) and the access plan requires a scan with no predicates.
- The table LOCKSIZE attribute is “TABLE”.
- The lock list fills, causing escalation.
- There is an explicit table lock acquired via the LOCK TABLE statement. The LOCK TABLE statement prevents concurrent application processes from either changing a table or using a table.

A *lock escalation* occurs when the number of locks held on rows and tables in the database equals the percentage of the locklist specified by the `maxlocks` database configuration parameter. Lock escalation might not affect the table that acquires the lock that triggers escalation. To reduce the number of locks to about half the number held when lock escalation begins, the database manager begins converting many small row locks to table locks for all active tables, beginning with any locks on large object (LOB) or long VARCHAR elements. An *exclusive lock escalation* is a lock escalation in which the table lock acquired is an exclusive lock. Lock escalations reduce concurrency. Conditions that might cause lock escalations should be avoided.

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 only held while the cursor is positioned on the row.
- RS scans: Only qualifying row locks are held for the duration of the transaction.
- RR scans: All row locks are held for the duration of the transaction.

Related concepts:

- "Lock attributes" on page 164
- "Locks and performance" on page 166
- "Guidelines for locking" on page 170

Related reference:

- "locklist - Maximum storage for lock list configuration parameter" in the *Administration Guide: Performance*
- "dlchktime - Time interval for checking deadlock" on page 792
- "diaglevel - Diagnostic error capture level configuration parameter" in the *Administration Guide: Performance*
- "locktimeout - Lock timeout" on page 793
- "Lock type compatibility" on page 173
- "Lock modes and access paths for standard tables" on page 174
- "Lock modes for table and RID index scans of MDC tables" on page 177
- "Locking for block index scans for MDC tables" on page 180

Lock attributes

Database manager locks have the following basic attributes:

Mode The type of access allowed for the lock owner as well as the type of access permitted 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 imposes locks on other types of resources, such as rows, tables, and table spaces. For multidimensional clustering (MDC) tables, block locks can also be imposed. The object being locked determines the *granularity* of the lock.

Duration
The length of time a lock is held. The isolation level in which the query runs affects the lock duration.

The following table shows the modes and their effects in order of increasing control over resources. For detailed information about locks at various levels, refer to the lock-mode reference tables.

Table 20. Lock Mode Summary

Lock Mode	Applicable Object Type	Description
IN (Intent None)	Table spaces, blocks, tables	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	The lock owner can read data in the locked table, but cannot update this data. Other applications can read or update the table.
NS (Next Key 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. NS lock mode is not used for next-key locking. It is used instead of S mode during CS and RS scans to minimize the impact of next-key locking on these scans.
S (Share)	Rows, blocks, tables	The lock owner and all concurrent applications can read, but not update, the locked data.
IX (Intent Exclusive)	Table spaces, blocks, tables	The lock owner and concurrent applications can read and update data. Other concurrent applications can both read and update the table.
SIX (Share with Intent Exclusive)	Tables, blocks	The lock owner can read and update data. Other concurrent applications can read the table.
U (Update)	Rows, blocks, tables	The lock owner can update data. Other units of work can read the data in the locked object, but cannot attempt to update it.
NW (Next Key Weak Exclusive)	Rows	When a row is inserted into an index, an NW lock is acquired on the next row. For type 2 indexes, 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 W and NS locks.
X (Exclusive)	Rows, blocks, tables, buffer pools	The lock owner can both read and update data in the locked object. Only uncommitted read applications can access the locked object.
W (Weak Exclusive)	Rows	This lock is acquired on the row when a row is inserted into a table that does not have type-2 indexes defined. The lock owner can change the locked row. To determine if a duplicate value has been committed when a duplicate value is found, this lock is also used during insertion into a unique index. This lock is similar to an X lock except that it is compatible with the NW lock. Only uncommitted read applications can access the locked row.
Z (Super Exclusive)	Table spaces, tables	This lock is acquired on a table in 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.

Related concepts:

- "Locks and concurrency control" on page 163
- "Locks and performance" on page 166

Related reference:

- "maxlocks - Maximum percent of lock list before escalation" on page 794
- "Lock type compatibility" on page 173

• "Lock modes and access paths for standard tables" on page 174

Locks and performance

Several related factors affect the uses of locks and their effect on application performance. The following factors are discussed here:
- Concurrency and granularity
- Lock compatibility
- Lock conversion
- Lock escalation
- Lock waits and timeouts
- Deadlocks

Concurrency and 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 are better for maximum concurrency than 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 scope (granularity) of locks at either row or table level. By default, row locks are used. Only S (Shared) and X (Exclusive) locks are requested by these defined table locks. The ALTER TABLE statement LOCKSIZE ROW clause does not prevent normal lock escalation from occurring.

A permanent table lock defined by the ALTER TABLE statement might be preferable to a single-transaction table lock using LOCK TABLE statement in the following cases:
- 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 runs, the read-only applications are locked out, but in other circumstances, read-only applications can access the table concurrently with a minimum of locking overhead.

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.

Lock compatibility

Lock compatibility is another important factor in concurrent access of tables. Lock compatibility refers to the current lock on the object and the type of lock being requested on that object and determines whether the request can be granted.

Assume that application A holds a lock on a table that application B also wants to access. The database manager requests, on behalf of application B, a lock of some particular mode. If the mode of the lock held by A permits the lock requested by B, the two locks (or modes) are said to be compatible.

If the lock mode requested for application B is not compatible with the lock held by application A, application B cannot continue. Instead, it must wait until application A releases its lock, and all other existing incompatible locks are released.

Lock conversion

Changing the mode of a lock already held is called a *conversion*. Lock conversion occurs when a process accesses a data object on which it already holds a lock, and the access mode requires a more restrictive lock than the one already held. A process can hold only one lock on a data object at any time, although it can request a lock many times on the same data object indirectly through a query.

Some lock modes apply only to tables, others only to rows or blocks. For rows or blocks, conversion usually occurs if an X is needed and an S or U (Update) lock is held.

IX (Intent Exclusive) and S (Shared) locks are special cases with regard to lock conversion, however. Neither S nor IX is considered to be more restrictive than the other, so if one of these is held and the other is required, the resulting conversion is to a SIX (Share with Intent Exclusive) lock. All other conversions result in the requested lock mode becoming the mode of the lock held if the requested mode is more restrictive.

A dual conversion might also occur when a query updates a row. If the row is read through an index access and locked as S, the table that contains the row has a covering intention lock. But if the lock type is IS instead of IX, if the row is subsequently changed the table lock is converted to an IX and the row to an X.

Lock conversion usually takes place implicitly as a query is executed. Understanding the kinds of locks obtained for different queries and table and index combinations can assist you in designing and tuning your application.

Lock Escalation

Lock escalation is an internal mechanism that reduces the number of locks held. In a single table, locks are escalated to a table lock from many row locks, or for multi-dimensional clustering (MDC) tables, from many row or block locks. Lock escalation occurs when applications hold too many locks of any type. Lock escalation can occur for a specific database agent if the agent exceeds its allocation of the lock list. Such escalation is handled internally; the only externally detectable result might be a reduction in concurrent access on one or more tables.

In an appropriately configured database, lock escalation occurs infrequently. For example, lock escalation might occur when an application designer creates an index on a large table to increase performance and concurrency but a transaction accesses most of the records in the table. In this case, because the database manager cannot predict how much of the table will be locked, it locks each record individually instead of locking only the table either S or X. In this case, the database designer might consult with the application designer, and recommend a LOCK TABLE statement for this transaction.

Occasionally, the process receiving the internal escalation request holds few or no row locks on any table, but locks are escalated because one or more processes hold

many locks. The process might not request another lock or access the database again except to end the transaction. Then another process might request the lock or locks that trigger the escalation request.

Note: Lock escalation might also cause deadlocks. For example, suppose a read-only application and an update application are both accessing the same table. If the update application has exclusive locks on many rows on the table, the database manager might try to escalate the locks on this table to an exclusive table lock. However, the table lock held by the read-only application will cause the exclusive lock escalation request to wait. If the read-only application requires a row lock on a row already locked by the update application, this creates a deadlock. To avoid this kind of problem, either code the update application to lock the table exclusively when it starts or increase the size of the lock list.

Lock waits and timeouts

Lock timeout detection is a database manager feature that prevents applications from waiting indefinitely for a lock to be released in an abnormal situation. For example, a transaction might be waiting for a lock held by another user's application, but the other user has left the workstation without allowing the application to commit the transaction that would release the lock. To avoid stalling an application in such a case, set the *locktimeout* configuration parameter to the maximum time that any application should wait to obtain a lock.

Setting this parameter helps avoid global deadlocks, especially in distributed unit of work (DUOW) applications. If the time that the lock request is pending is greater than the *locktimeout* value, the requesting application receives an error and its transaction is rolled back. For example, if *program1* tries to acquire a lock which is already held by *program2*, *program1* returns SQLCODE -911 (SQLSTATE 40001) with reason code 68 if the timeout period expires. The default value for *locktimeout* is -1, which turns off lock timeout detection.

Note: For table, row, and MDC block locks, an application can override the database level *locktimeout* setting by using SET CURRENT LOCK TIMEOUT.

To log more information about lock-request timeouts in the administration notification log, set the database manager configuration parameter *notifylevel* to four. The logged information includes the locked object, the lock mode, and the application holding the lock. The current dynamic SQL statement or static package name might also be logged. A dynamic SQL statement is logged only at *notifylevel* four.

Deadlocks

Contention for locks can result in deadlocks. For example, suppose that Process 1 locks table A in X (exclusive) mode and Process 2 locks table B in X mode. If Process 1 then tries to lock table B in X mode and Process 2 tries to lock table A in X mode, the processes are in a deadlock. In a deadlock, both processes are suspended until their second lock request is granted, but neither request is granted until one of the processes performs a commit or rollback. This state continues indefinitely until an external agent activates one of the processes and forces it to perform a rollback.

To handle deadlocks, the database manager uses an asynchronous system background process called the deadlock detector. The deadlock detector becomes

active at intervals specified by the *dlchktime* configuration parameter. When activated, the deadlock detector examines the lock system for deadlocks. In a partitioned database, each partition sends *lock graphs* to the database partition that contains the system catalog views. Global deadlock detection takes place on this partition.

If it finds a deadlock, the deadlock detector 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 the selected process automatically. When the rollback is complete, the locks that belonged to the victim process are released, and the other processes involved in the deadlock can continue.

To ensure good performance, select the proper interval for the deadlock detector. An interval that is too short causes unnecessary overhead, and an interval that is too long allows a deadlock to delay a process for an unacceptable amount of time. For example, a wake-up interval of 5 minutes allows a deadlock to exist for almost 5 minutes, which can seem like a long time for short transaction processing. Balance® the possible delays in resolving deadlocks with the overhead of detecting them.

In a partitioned database, the *dlchktime* configuration parameter interval is applied only at the catalog node. If a large number of deadlocks are detected in a partitioned database, increase the value of the *dlchktime* parameter to account for lock waits and communication waits.

A different problem occurs when an application with more than one independent process that accesses the database is structured to make deadlocks likely. An example is an application in which several processes access the same table for reads and then writes. If the processes do read-only SQL queries at first and then do SQL updates on the same table, the chance of deadlocks increases because of potential contention between the processes for the same data. For instance, if two processes read the table, and then update the table, process A might try to get an X lock on a row on which process B has an S lock, and vice versa. To avoid such deadlocks, applications that access data with the intention of modifying it should do one of the following:

- Use the FOR UPDATE OF clause when performing a select. This clause ensures that a U lock is imposed when process A attempts to read the data. Row blocking, however, is disabled.
- Use the WITH RR USE AND KEEP UPDATE LOCKS or the WITH RS USE AND KEEP UPDATE LOCKS clause when performing the query. Either clause ensures that a U lock is imposed when process A attempts to read the data, and allows row blocking.

Note: You might consider defining a monitor that records when deadlocks occur. Use the SQL statement CREATE EVENT to create a 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.

In a federated system environment in which an application accesses nicknames, the data requested by the application might not be available because of a deadlock at a data source. When this happens, DB2® relies on the deadlock handling facilities at the data source. If deadlocks occur across more than one data source, DB2 relies on data source timeout mechanisms to break the deadlock.

To log more information about deadlocks, set the database manager configuration parameter *notifylevel* to four. The administration notification log stores information that includes the object, the lock mode, and the application holding the lock on the object. The current dynamic SQL statement or static package name might also be logged. The dynamic SQL statement is logged only if *notifylevel* is four.

Related concepts:

- "Locks and concurrency control" on page 163
- "Deadlocks between applications" on page 155

Related tasks:

- "Correcting lock escalation problems" on page 172

Related reference:

- "diaglevel - Diagnostic error capture level configuration parameter" in the *Administration Guide: Performance*
- "locktimeout - Lock timeout" on page 793

Guidelines for locking

Consider the following guidelines when you tune locking for 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 is issued, locks are released except for table locks associated with cursors declared WITH HOLD.

- Specify an appropriate isolation level.

  Locks are acquired even if your application merely reads rows, so it is still important to commit read-only units of work. This is because shared locks are acquired by repeatable read, read stability, and cursor stability isolation levels in read-only applications. With repeatable read and read stability, all locks are held until a COMMIT is issued, preventing other processes from updating the locked data, unless you close your cursor using the WITH RELEASE clause. In addition, catalog locks are acquired even in uncommitted read applications using dynamic SQL.

  The database manager ensures that your application does not retrieve uncommitted data (rows that have been updated by other applications but are not yet committed) unless you are using the uncommitted read isolation level.

- Use the LOCK TABLE statement appropriately.

  The statement locks an entire table. Only the table specified in the LOCK TABLE statement is locked. 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 the desired result in terms of concurrency and performance. The lock is not released until the unit of work is committed or rolled back.

LOCK TABLE IN SHARE MODE

You want to access data that is *consistent in time*; that is, data current for a table at a specific point in time. If the table experiences frequent activity, the only way to ensure that the entire table remains stable is to lock it. For example, your application wants to take a snapshot of a table. However, during the time your application needs to process some rows of a table, other applications are updating rows you have not yet processed. This is allowed with repeatable read, but this action is not what you want.

As an alternative, your application can issue the LOCK TABLE IN SHARE MODE statement: no rows can be changed, regardless of whether you have retrieved them or not. You can then retrieve as many rows as you need, knowing that the rows you have retrieved have not been changed just before you retrieved them.

With LOCK TABLE IN SHARE MODE, other users can retrieve data from the table, but they cannot update, delete, or insert rows into the table.

LOCK TABLE IN EXCLUSIVE MODE

You want to update a large part of the table. It is less expensive and more efficient to prevent all other users from accessing the table than it is to lock each row as it is updated, and then unlock the row later when all changes are committed.

With LOCK TABLE IN EXCLUSIVE MODE, all other users are locked out; no other applications can access the table unless they are uncommitted read applications.

- Use ALTER TABLE statements in applications.

  The ALTER TABLE statement with the LOCKSIZE parameter is an alternative to the LOCK TABLE statement. The LOCKSIZE parameter lets you specify a lock granularity of either ROW locks or TABLE locks for the next table access.

  The selection of ROW locks is no different from selecting the default lock size when a table is created. The selection of TABLE locks may improve the performance of queries by limiting the number of locks that need to be acquired. However, concurrency might be reduced because all locks are on the complete table. Neither choice prevents normal lock escalation.

- Close cursors to release the locks that they hold.

  When you close a cursor with the CLOSE CURSOR statement that includes the WITH RELEASE clause, the database manager attempts to release all read locks that have been held for the cursor. Table read locks are IS, S, and U table locks. Row-read locks are S, NS, and U row locks. Block-read locks are IS, S, and U block locks.

  The WITH RELEASE clause has no effect on cursors that are operating under the CS or UR isolation levels. When specified for cursors that are operating under the RS or RR isolation levels, the WITH RELEASE clause ends some of the guarantees of those isolation levels. Specifically, a RS cursor may experience the *nonrepeatable read* phenomenon, and a RR cursor may experience either the *nonrepeatable read* or *phantom read* phenomenon.

  If a cursor that is originally RR or RS is reopened after being closed using the WITH RELEASE clause, then 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 CURSOR WITH RELEASE.

- In a partitioned database, when you changing the configuration parameters that affecting locking, ensure that the changes are made to all of the partitions.

Related concepts:

- "Locks and concurrency control" on page 163
- "Lock attributes" on page 164
- "Locks and performance" on page 166
- "Factors that affect locking" on page 182

Correcting lock escalation problems

The database manager can automatically escalate locks from row or block level to table level. The *maxlocks* database configuration parameter specifies when lock escalation is 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 long object (LOBs) and long VARCHAR descriptors are locked, then 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 rarely occurs. If lock escalation reduces concurrency to an unacceptable level, however, you need to analyze the problem and decide how to solve it.

Prerequisites:

Ensure that lock escalation information is recorded. Set the database manager configuration parameter *notifylevel* to 3, which is the default, or to 4. At *notifylevel* of 2, only the error SQLCODE is reported. At *notifylevel* of 3 or 4, when lock escalation fails, information is recorded for the error SQLCODE and the table for which the escalation failed. The current SQL statement is logged only if it is a currently executing, dynamic SQL statement and *notifylevel*is set to 4.

Procedure:

Follow these general steps to diagnose the cause of unacceptable lock escalations and apply a remedy:

1. Analyze in the administration notification log on all tables for which locks are 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 information 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 Share lock, or an "X," or eXclusive lock is acquired.
   - The internal return code of the result of the acquisition of the new table lock level.
2. Use the information in administration notification log to decide how to resolve the escalation problem. Consider the following possibilities:

- Increase the number of locks allowed globally by increasing the value of the *maxlocks* or the *locklist* parameters, or both, in the database configuration file. In a partitioned database, make this change on all partitions.

  You might choose this method if concurrent access to the table by other processes is most important. However, the overhead of obtaining record level locks can induce more delay to other processes than is saved by concurrent access to a table.
- Adjust the process or processes that caused the escalation. For these processes, you might issue LOCK TABLE statements explicitly.
- Change the degree of isolation. Note that this may lead to decreased concurrency, however.
- Increase the frequency of commits to reduce the number of locks held at a given time.
- Consider frequent COMMIT statements for transactions that require long VARCHAR or various kinds of long 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 for rows that contain more ordinary kinds of data.

Related reference:

- "maxlocks - Maximum percent of lock list before escalation" on page 794
- "diaglevel - Diagnostic error capture level configuration parameter" in the *Administration Guide: Performance*

Lock type compatibility

The following table displays information about the circumstances in which a lock request can be granted when another process holds or is requesting a lock on the same resource in a given state. A **no** indicates that the requestor must wait until all incompatible locks are released by other processes. Note that a timeout can occur when a requestor is waiting for a lock. A **yes** indicates that the lock is granted unless an earlier requestor is waiting for the resource.

Table 21. Lock Type Compatibility

	State of Held Resource											
State Being Requested	**none**	**IN**	**IS**	**NS**	**S**	**IX**	**SIX**	**U**	**X**	**Z**	**NW**	**W**
none	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes
IN	yes	yes	yes	yes	yes	yes	yes	yes	yes	no	yes	yes
IS	yes	yes	yes	yes	yes	yes	yes	yes	no	no	no	no
NS	yes	yes	yes	yes	yes	no	no	yes	no	no	yes	no
S	yes	yes	yes	yes	yes	no	no	yes	no	no	no	no
IX	yes	yes	yes	no	no	yes	no	no	no	no	no	no
SIX	yes	yes	yes	no	no	no	no	no	no	no	no	no
U	yes	yes	yes	yes	yes	no	no	no	no	no	no	no
X	yes	yes	no	no	no	no	no	no	no	no	no	no
Z	yes	no	no	no	no	no	no	no	no	no	no	no
NW	yes	yes	no	yes	no	no	no	no	no	no	no	yes
W	yes	yes	no	no	no	no	no	no	no	no	yes	no

Note:

I	Intent
N	None
NS	Next Key Share
S	Share
X	Exclusive
U	Update
Z	Super Exclusive
NW	Next Key Weak Exclusive
W	Weak Exclusive

Note:

- yes - **grant** lock requested immediately
- no - **wait** for held lock to be released or timeout to occur

Related concepts:

- "Locks and concurrency control" on page 163
- "Lock attributes" on page 164
- "Locks and performance" on page 166

Related reference:

- "Lock modes and access paths for standard tables" on page 174
- "Locking for block index scans for MDC tables" on page 180

Lock modes and access paths for standard tables

This topic includes reference information about locking methods for standard tables for different data-access plans.

The following tables list the types of locks obtained for standard tables at each level for different access plans. Each entry is made up of two parts: table lock and row lock. A dash indicates that a particular level of locking is not done.

Notes:

1. In a multi-dimensional clustering (MDC) environment, an additional lock level, BLOCK, is used.
2. Lock modes can be changed explicitly with the lock-request-clause of a select statement.

Table 22. Lock Modes for Table Scans

Isolation Level	Read-only and ambiguous scans	Cursored operation		Searched update or delete	
		Scan	Where current of	Scan	Update or delete
Access Method: ***Table scan with no predicates***					
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
Access Method: ***Table Scan with predicates***					
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: At UR isolation level with IN lock for type-1 indexes or if there are predicates on include columns in the index, the isolation level is upgraded to CS and the locks to an IS table lock and NS row locks.

Table 23. Lock Modes for RID Index Scans

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan	Update or Delete
Access Method: ***RID index scan with no predicates***					
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
Access Method: ***RID index scan with a single qualifying row***					
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
Access Method: ***Index scan with start and stop predicates only***					
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 23. Lock Modes for RID Index Scans (continued)

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan	Update or Delete
Access Method: ***Index Scan with index and other predicates (sargs, resids) only***					
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

The following table shows the lock modes for cases in which reading of the data pages is deferred to allow the list of rows to be:

- Further qualified using multiple indexes
- Sorted for efficient prefetching

Table 24. Lock modes for index scans used for deferred data page access

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan	Update or delete
Access Method: ***RID index scan with no predicates***					
RR	IS/S	IX/S		X/-	
RS	IN/-	IN/-		IN/-	
CS	IN/-	IN/-		IN/-	
UR	IN/-	IN/-		IN/-	
Access Method: ***Deferred Data Page Access, after a RID index scan with no predicates***					
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
Access Method: ***RID index scan with predicates (sargs, resids)***					
RR	IS/S	IX/S		IX/S	
RS	IN/-	IN/-		IN/-	
CS	IN/-	IN/-		IN/-	
UR	IN/-	IN/-		IN/-	
Access Method: ***RID index scan with start and stop predicates only***					
RR	IS/S	IX/S		IX/X	
RS	IN/-	IN/-		IN/-	
CS	IN/-	IN/-		IN/-	
UR	IN/-	IN/-		IN/-	
Access Method: ***Deferred data-page access after a RID index scan with start and stop predicates only***					

Table 24. Lock modes for index scans used for deferred data page access (continued)

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
Access Method: ***Deferred data-page Access, after a RID index scan with predicates***					
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

Related concepts:

- "Lock attributes" on page 164
- "Locks and performance" on page 166

Related reference:

- "Lock type compatibility" on page 173
- "Lock modes for table and RID index scans of MDC tables" on page 177
- "Locking for block index scans for MDC tables" on page 180

Lock modes for table and RID index scans of MDC tables

In a multi-dimensional clustering (MDC) environment, an additional lock level, BLOCK, is used. The following tables list the types of locks obtained at each level for different access plans. Each entry is made up of three parts: table lock, block lock, and row lock. A dash indicates that a particular level of locking is not used.

Note: Lock modes can be changed explicitly with the lock-request-clause of a select statement.

Table 25. Lock Modes for Table Scans

Isolation Level	Read-only and ambiguous scans	Cursored operation		Searched update or delete	
		Scan	Where current of	Scan or delete	Update
Access Method: ***Table scan with no predicates***					
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/-
Access Method: ***Table Scan with predicates on dimension columns only***					
RR	S/-/-	U/-/-	SIX/IX/X	U/-/-	SIX/X/-

Table 25. Lock Modes for Table Scans (continued)

Isolation Level	Read-only and ambiguous scans	Cursored operation		Searched update or delete	
		Scan	Where current of	Scan or delete	Update
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/-
Access Method: *Table Scan with other predicates (sargs, resids)*					
RR	S/-/-	U/-/-	SIX/IX/X	U/-/-	SIX/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

The following two tables show lock modes for RID indexes on MDC tables.

Table 26. Lock Modes for RID Index Scans

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan Delete	Update
Access Method: *RID index scan with no predicates*					
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
Access Method: *RID index scan with single qualifying row*					
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
Access Method: *RID index scan with start and stop predicates only*					
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
Access Method: *Index scan with index predicates only*					
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
Access Method: *Index scan with other predicates (sargs, resids)*					
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 26. Lock Modes for RID Index Scans (continued)

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan Delete	Update
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

Note: In the following table, which shows lock modes for RID index scans used for deferred data-page access, at UR isolation level with IN lock for type-1 indexes or 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, and NS row locks.

Table 27. Lock modes for RID index scans used for deferred data-page access

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan Delete	Update
Access Method: ***RID index scan with no predicates***					
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/-	
Access Method: ***Deferred data-page access after a RID index scan with no predicates***					
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
Access Method: ***RID index scan with predicates (sargs, resids)***					
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/-	
Access Method: ***Deferred data-page access after a RID index scan with predicates (sargs, resids)***					
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
Access Method: ***RID index scan with start and stop predicates only***					
RR	IS/IS/S	IX/IX/S		IX/IX/X	
RS	IN/IN/-	IN/IN/-		IN/IN/-	
CS	IN/IN/-	IN/IN/-		IN/IN/-	

Table 27. Lock modes for RID index scans used for deferred data-page access (continued)

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan Delete	Update
UR	IN/IN/-	IN/IN/-		IN/IN/-	
Access Method: *Deferred data-page access after a RID index scan with start and stop predicates only*					
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

Related concepts:

- "Locks and concurrency control" on page 163
- "Lock attributes" on page 164
- "Locks and performance" on page 166

Related reference:

- "Lock type compatibility" on page 173
- "Lock modes and access paths for standard tables" on page 174
- "Locking for block index scans for MDC tables" on page 180

Locking for block index scans for MDC tables

The following tables list the types of locks obtained at each level for different access plans. Each entry is made up of three parts: table lock, block lock, and row lock. A dash indicates that a particular level of locking is not done.

Note: Lock modes can be changed explicitly with the lock-request-clause of a select statement.

Table 28. Lock Modes for Index Scans

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan Delete	Update
Access Method: *With no predicates*					
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/--
Access Method: *With dimension predicates only*					
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 28. Lock Modes for Index Scans (continued)

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan Delete	Update
Access Method: ***With dimension start and stop predicates only***					
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/-
Access Method: ***Index Scan with predicates***					
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

The following table lists lock modes for block index scans used for deferred data-page access:

Table 29. Lock modes for block index scans used for deferred data-page access

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan Delete	Update
Access Method: ***Block index scan with no predicates***					
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/--	
Access Method: ***Deferred data-page access after a block index scan with no predicates***					
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/--
Access Method: ***Block index scan with dimension predicates only***					
RR	IS/S/--	IX/IX/--		IX/S/--	
RS	IS/IS/NS	IX/--/--		IX/--/--	
CS	IS/IS/NS	IX/--/--		IX/--/--	
UR	IN/IN/--	IX/--/--		IX/--/--	
Access Method: ***Deferred data-page access after a block index scan with dimension predicates only***					
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/--

Table 29. Lock modes for block index scans used for deferred data-page access (continued)

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan Delete	Update
UR	IN/IN/--	IX/IX/U	IX/IX/X	IX/U/--	IX/X/--
Access Method: ***Block index scan with start and stop predicates only***					
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/--	
Access Method: ***Deferred data-page access after a block index scan with start and stop predicates only***					
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/--	
Access Method: ***Block index scan other predicates (sargs, resids)***					
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/--	
Access Method: ***Deferred data-page access after a block index scan with other predicates (sargs, resids***					
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

Related concepts:

- "Locks and performance" on page 166

Related reference:

- "Lock type compatibility" on page 173
- "Lock modes and access paths for standard tables" on page 174
- "Lock modes for table and RID index scans of MDC tables" on page 177

Factors that affect locking

The following factors affect the mode and granularity of database manager locks:

- The type of processing that the application performs
- The data access method
- Whether indexes are type-2 or type-1
- Various configuration parameters

Related concepts:

- "Locks and concurrency control" on page 163
- "Lock attributes" on page 164
- "Locks and performance" on page 166
- "Guidelines for locking" on page 170
- "Index cleanup and maintenance" in the *Administration Guide: Performance*
- "Locks and types of application processing" on page 183
- "Locks and data-access methods" on page 184
- "Index types and next-key locking" on page 185

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

  This type includes all select statements that are intrinsically read-only, have an explicit FOR READ ONLY clause, or are ambiguous but which the SQL compiler assumes to be read-only because of the value of the BLOCKING option that the PREP or BIND command specifies. This processing type requires only Share locks (S, NS, or IS).
- Intent to change

  This type includes all select statements with the FOR UPDATE clause, with the USE AND KEEP UPDATE LOCKS clause, with the USE AND KEEP EXCLUSIVE LOCKS clause, or for which the SQL compiler interprets an ambiguous statement to imply that change is intended. This type uses Share and Update locks (S, U, and X for rows; IX, U, X, and S for blocks; IX, U, and X for tables).
- Change

  This type includes UPDATE, INSERT, and DELETE, but not UPDATE WHERE CURRENT OF or DELETE WHERE CURRENT OF. This type requires Exclusive locks (X or IX).
- Cursor controlled

  This type includes UPDATE WHERE CURRENT OF and DELETE WHERE CURRENT OF. It also requires Exclusive locks (X or IX).

A statement that inserts, updates or deletes data in a target table, based on the result from a sub-select statement, does two types of processing. The rules for read-only processing determine the locks for the tables returned in the sub-select statement. The rules for change processing determine the locks for the target table.

Related concepts:

- "Locks and concurrency control" on page 163
- "Lock attributes" on page 164
- "Locks and performance" on page 166
- "Guidelines for locking" on page 170
- "Deadlocks between applications" on page 155
- "Locks and data-access methods" on page 184
- "Index types and next-key locking" on page 185

Related tasks:

- "Correcting lock escalation problems" on page 172

Related reference:

- "Lock type compatibility" on page 173

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. For example, when an index scan is used to locate a specific row, the optimizer will probably choose row-level locking (IS) for the table. For example, if the EMPLOYEE table that has an index on employee number (EMPNO), access through an index might be used to select information for a single employee with a statement that contains the following SELECT clause:

```
SELECT *
  FROM EMPLOYEE
  WHERE EMPNO = '000310';
```

If an index is not used, the entire table must be scanned in sequence to find the selected rows, and may thus acquire 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 with a a statement that contains the following SELECT clause:

```
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 a cursor, an exclusive lock is always obtained to perform the update or delete.

Locking in range-clustered tables works slighly differently from standard key or next-key locking. In 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 or next key locking, only rows with existing records are locked.

Reference tables provide detailed information about which locks are obtained for what kind of access plan.

Deferred access of the data pages implies that access to the row occurs in two steps, which results in more complex locking scenarios. The timing of lock aquisition and the persistence of the locks depend on the isolation level. Because the Repeatable Read isolation level retains all locks until the end of the transaction, the locks acquired in the first step are held and there is no need to acquire further locks in the second step. For the Read Stability and Cursor Stability isolation levels, locks must be acquired during the second step. To maximize concurrency, locks are not acquired during the first step and rely on the reapplication of all predicates to ensure that only qualifying rows are returned.

Related concepts:

- "Locks and concurrency control" on page 163
- "Lock attributes" on page 164
- "Locks and performance" on page 166

- "Guidelines for locking" on page 170
- "Locks and types of application processing" on page 183
- "Index types and next-key locking" on page 185

Related tasks:

- "Correcting lock escalation problems" on page 172

Related reference:

- "Lock type compatibility" on page 173
- "Lock modes and access paths for standard tables" on page 174
- "Lock modes for table and RID index scans of MDC tables" on page 177
- "Locking for block index scans for MDC tables" on page 180

Index types and next-key locking

As transactions cause changes to type-1 indexes, some next-key locking occurs. For type-2 indexes, minimal next-key locking occurs.

- Next-key locking for type 2 indexes

  Next-key locking occurs when a key is inserted into an index.

  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 an RR index scan. The lock mode used for the next-key lock is NW. This next-key lock is released before the key insertion is actually performed. Key insertion occurs when a row is inserted into a table.

  When updates to a row result in a change to the value of the index key for that row, key insertion also occurs 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. If the key that follows the end of the scan range is marked deleted, the scan continues to lock the corresponding rows until it finds a key that is not marked deleted, when it locks the corresponding row for that key, or until the end of the index is locked.

- Next-key locking for type-1 indexes:

  Next-key locks occur during deletes and inserts to indexes and during index scans. When a row is updated in, deleted from, or inserted into a table, an X lock is obtained on that row. For insertions this might be downgraded to a W lock.

  When the key is deleted from the table index or inserted into it, the table row that corresponds to the key that follows the deleted or inserted key in the index is locked. For updates that affect the value of the key, the original key value is first deleted and the new value is inserted, so two next-key locks are acquired. The duration of these locks is determined as follows:

  – During index key deletion, the lock mode on the next key is X and the lock is held until commit time.
  – During index key insertion, the lock mode on the next key is NW. This lock is acquired only if there is contention for the lock, in which case the lock is released before the key is actually inserted into the index.

– During RR scans, the table row that corresponds to the key just beyond the end of the index scan range is locked in S mode and is held until commit time.
– During CS/RS scans, the row corresponding to the key just beyond the end of the index scan range is locked in NS mode if there is contention for the lock. This lock is released once the end of the scan range is verified.

The next-key locking for type-1 indexes during key insertions and key deletion might result in deadlocks. The following example shows how two transactions could deadlock. With type 2 indexes, such deadlocks do not occur.

Consider the following example of an index that contains 6 rows with the following values: `1 5 6 7 8 12`.

1. Transaction 1 deletes the row with key value 8. The row with value 8 is locked in X mode. When the corresponding key from the index is deleted, the row with value 12 is locked in X mode.
2. Transaction 2 deletes the row with key value 5. The row with value 5 is locked in X mode. When the corresponding key from the index is deleted, the row with value 6 is locked in X mode.
3. Transaction 1 inserts a row with key value 4. This row is locked in W mode. When inserting the new key into the index is attempted, the row with value 6 is locked in NW mode. This lock attempt will wait on the X lock that transaction 2 has on this row.
4. Transaction 2 inserts a row with key value 9. This row is locked in W mode. When inserting the new key into the index is attempted, the row with key value 12 is locked in NW mode. This lock attempt will wait on the X lock that transaction 1 has on this row.

When type-1 indexes are used, this scenario will result in a deadlock and one of these transactions will be rolled back.

Related concepts:

- "Advantages and disadvantages of indexes" in the *Administration Guide: Performance*
- "Index performance tips" in the *Administration Guide: Performance*
- "Index structure" in the *Administration Guide: Performance*
- "Index reorganization" in the *Administration Guide: Performance*
- "Online index defragmentation" in the *Administration Guide: Performance*
- "Index cleanup and maintenance" in the *Administration Guide: Performance*

Chapter 9. Configuring DB2 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 DB2 to be Common Criteria compliant

Immediately *after* installing DB2, you 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.

Prerequisites:

To update configuration parameter values, you require the SYSADM authority level.

Procedure:

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
   ```

2. Update the database manager configuration to specify the TCP/IP port that DB2 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 DB2. For additional information about the *svcename* database manager configuration parameter, see ″svcename - TCP/IP service name″

3. Stop DB2. Issue the following command:

   ```
   db2stop
   ```

4. Start DB2. Issue the following command:

   ```
   db2start
   ```

The updated database manager configuration parameters take effect when DB2 is restarted.

Related concepts:

- “System administration authority (SYSADM)” on page 21

Related reference:

- “svcename - TCP/IP service name” on page 786
- “authentication - Authentication type” on page 783

© Copyright IBM Corp. 1993-2004

- "UPDATE DATABASE MANAGER CONFIGURATION" on page 384
- "ATTACH" on page 839

Chapter 10. System catalogs and security maintenance

Using the system catalog for security issues

Information about each database is automatically maintained in a set of views called the system catalog, which is created when the database is generated. This system catalog describes tables, columns, indexes, programs, privileges, and other objects.

These views list the privileges held by users and the identity of the user granting each privilege:

SYSCAT.DBAUTH	Lists the database privileges
SYSCAT.TABAUTH	Lists the table and view privileges
SYSCAT.COLAUTH	Lists the column privileges
SYSCAT.PACKAGEAUTH	Lists the package privileges
SYSCAT.INDEXAUTH	Lists the index privileges
SYSCAT.SCHEMAAUTH	Lists the schema privileges
SYSCAT.PASSTHRUAUTH	Lists the server privilege
SYSCAT.ROUTINEAUTH	Lists the routine (functions, methods, and stored procedures) privileges

Privileges granted to users by the system will have SYSIBM as the grantor. SYSADM, SYSMAINT and SYSCTRL are not listed in the system catalog.

The CREATE and GRANT statements place privileges in the system catalog. Users with SYSADM and DBADM authorities can grant and revoke SELECT privilege on the system catalog views.

Related tasks:

- "Retrieving authorization names with granted privileges" on page 190
- "Retrieving all names with DBADM authority" on page 190
- "Retrieving names authorized to access a table" on page 191
- "Retrieving all privileges granted to users" on page 191
- "Securing the system catalog view" on page 192

© Copyright IBM Corp. 1993-2004

Related reference:

- "SYSCAT.COLAUTH" on page 193
- "SYSCAT.DBAUTH" on page 194
- "SYSCAT.INDEXAUTH" on page 195
- "SYSCAT.PACKAGEAUTH" on page 195
- "SYSCAT.SCHEMAAUTH" on page 198
- "SYSCAT.TABAUTH" on page 206
- "SYSCAT.PASSTHRUAUTH" on page 197
- "SYSCAT.ROUTINEAUTH catalog view" in the *SQL Reference, Volume 1*

Details on using the system catalog for security issues

This section reviews some of the ways to determine who has what privileges within the database.

Retrieving authorization names with granted privileges

Procedure:

No single system catalog view contains information about all privileges. 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.

Related concepts:

- "Using the system catalog for security issues" on page 189

Retrieving all names with DBADM authority

Procedure:

The following statement retrieves all authorization names that have been directly granted DBADM authority:

```
SELECT DISTINCT GRANTEE FROM SYSCAT.DBAUTH
   WHERE DBADMAUTH = 'Y'
```

Related concepts:

- "Database administration authority (DBADM)" on page 25
- "Using the system catalog for security issues" on page 189

Retrieving names authorized to access a table

Procedure:

The following statement retrieves all authorization names that are directly authorized to access the table EMPLOYEE with the qualifier JAMES:

```
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   = 'Y' OR  UPDATEAUTH   = 'G')
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. However, it will not return the authorization names of users who only hold SYSADM authority.

Remember that some of the authorization names may be groups, not just individual users.

Related concepts:

- "Table and view privileges" on page 28
- "Using the system catalog for security issues" on page 189

Retrieving all privileges granted to users

Procedure:

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. For example, the following statement retrieves a list of the database privileges that have been directly granted to an individual authorization name:

```
SELECT * FROM SYSCAT.DBAUTH
   WHERE GRANTEE = USER AND GRANTEETYPE = 'U'
```

The following statement retrieves a list of the table privileges that were directly granted by a specific user:

```
SELECT * FROM SYSCAT.TABAUTH
   WHERE GRANTOR  = USER
```

The following statement retrieves a list of the individual column privileges that were directly granted by a specific user:

```
SELECT * FROM SYSCAT.COLAUTH
   WHERE GRANTOR  = USER
```

The keyword USER in these statements is always equal to the value of a user's authorization name. USER is a read-only special register.

Related concepts:

- "Privileges, authority levels, and database authorities" on page 15
- "Database authorities" on page 24
- "Using the system catalog for security issues" on page 189

Related tasks:

- "Granting privileges" on page 44
- "Revoking privileges" on page 45

Securing the system catalog view

Procedure:

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 SYSADM or DBADM authority to do this.

At a minimum, you should consider restricting access to the following catalog views:

- SYSCAT.DBAUTH
- SYSCAT.TABAUTH
- SYSCAT.PACKAGEAUTH
- SYSCAT.INDEXAUTH
- SYSCAT.COLAUTH
- SYSCAT.PASSTHRUAUTH
- SYSCAT.SCHEMAAUTH

This would prevent information on user privileges from becoming available to everyone with access to the database. With this information, an unethical user could gain unauthorized 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 always equal to the value of the 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 base table:

```
REVOKE SELECT ON TABLE SYSCAT.TABAUTH FROM PUBLIC
```

Related concepts:

- "Catalog statistics" in the *Administration Guide: Performance*
- "Database authorities" on page 24
- "Using the system catalog for security issues" on page 189

Related tasks:

- "Granting privileges" on page 44
- "Revoking privileges" on page 45

System Catalog Views

SYSCAT.COLAUTH

Contains one or more rows for each user or group who is granted a column level privilege, indicating the type of privilege and whether or not it is grantable.

Table 30. SYSCAT.COLAUTH Catalog View

Column Name	Data Type	Nullable	Description
GRANTOR	VARCHAR(128)		Authorization ID of the user who granted the privileges or SYSIBM.
GRANTEE	VARCHAR(128)		Authorization ID of the user or group who holds the privileges.
GRANTEETYPE	CHAR(1)		U = Grantee is an individual user. G = Grantee is a group.
TABSCHEMA	VARCHAR(128)		Qualified name of the table or view.
TABNAME	VARCHAR(128)		
COLNAME	VARCHAR(128)		Name of the column to which this privilege applies.
COLNO	SMALLINT		Number of this column in the table or view.

Table 30. SYSCAT.COLAUTH Catalog View (continued)

Column Name	Data Type	Nullable	Description
PRIVTYPE	CHAR(1)		Indicates the type of privilege held on the table or view: U = Update privilege R = Reference privilege
GRANTABLE	CHAR(1)		Indicates if the privilege is grantable. G = Grantable N = Not grantable

SYSCAT.DBAUTH

Records the database authorities held by users.

Table 31. SYSCAT.DBAUTH Catalog View

Column Name	Data Type	Nullable	Description
GRANTOR	VARCHAR(128)		SYSIBM or authorization ID of the user who granted the privileges.
GRANTEE	VARCHAR(128)		Authorization ID of the user or group who holds the privileges.
GRANTEETYPE	CHAR(1)		U = Grantee is an individual user. G = Grantee is a group.
DBADMAUTH	CHAR(1)		Whether grantee holds DBADM authority over the database: Y = Authority is held. N = Authority is not held.
CREATETABAUTH	CHAR(1)		Whether grantee can create tables in the database (CREATETAB): Y = Privilege is held. N = Privilege is not held.
BINDADDAUTH	CHAR(1)		Whether grantee can create new packages in the database (BINDADD): Y = Privilege is held. N = Privilege is not held.
CONNECTAUTH	CHAR(1)		Whether grantee can connect to the database (CONNECT): Y = Privilege is held. N = Privilege is not held.
NOFENCEAUTH	CHAR(1)		Whether grantee holds privilege to create non-fenced functions. Y = Privilege is held. N = Privilege is not held.
IMPLSCHEMAAUTH	CHAR(1)		Whether grantee can implicitly create schemas in the database (IMPLICIT_SCHEMA): Y = Privilege is held. N = Privilege is not held.

Table 31. SYSCAT.DBAUTH Catalog View (continued)

Column Name	Data Type	Nullable	Description
LOADAUTH	CHAR(1)		Whether grantee holds LOAD authority over the database: Y = Authority is held. N = Authority is not held.
EXTERNALROUTINEAUTH	CHAR(1)		Whether grantee can create external routines (CREATE_EXTERNAL_ROUTINE): Y = Authority is held. N = Authority is not held.
QUIESCECONNECTAUTH	CHAR(1)		Whether grantee can connect to a database (QUIESCE_CONNECT): Y = Authority is held. N = Authority is not held.

SYSCAT.INDEXAUTH

Contains a row for every privilege held on an index.

Table 32. SYSCAT.INDEXAUTH Catalog View

Column Name	Data Type	Nullable	Description
GRANTOR	VARCHAR(128)		Authorization ID of the user who granted the privileges.
GRANTEE	VARCHAR(128)		Authorization ID of the user or group who holds the privileges.
GRANTEETYPE	CHAR(1)		U = Grantee is an individual user. G = Grantee is a group.
INDSCHEMA	VARCHAR(128)		Qualified name of the index.
INDNAME	VARCHAR(18)		
CONTROLAUTH	CHAR(1)		Whether grantee holds CONTROL privilege over the index: Y = Privilege is held. N = Privilege is not held.

SYSCAT.PACKAGEAUTH

Contains a row for every privilege held on a package.

Table 33. SYSCAT.PACKAGEAUTH Catalog View

Column Name	Data Type	Nullable	Description
GRANTOR	VARCHAR(128)		Authorization ID of the user who granted the privileges.
GRANTEE	VARCHAR(128)		Authorization ID of the user or group who holds the privileges.
GRANTEETYPE	CHAR(1)		U = Grantee is an individual user. G = Grantee is a group.

Table 33. SYSCAT.PACKAGEAUTH Catalog View (continued)

Column Name	Data Type	Nullable	Description
PKGSCHEMA	VARCHAR(128)		Name of the package on which the privileges are held.
PKGNAME	CHAR(8)		
CONTROLAUTH	CHAR(1)		Indicates whether grantee holds CONTROL privilege on the package: Y = Privilege is held. N = Privilege is not held.
BINDAUTH	CHAR(1)		Indicates whether grantee holds BIND privilege on the package: Y = Privilege is held. N = Privilege is not held. G = Privilege is held and grantable.
EXECUTEAUTH	CHAR(1)		Indicates whether grantee holds EXECUTE privilege on the package: Y = Privilege is held. N = Privilege is not held. G = Privilege is held and grantable.

SYSCAT.PACKAGEDEP

Contains a row for each dependency that packages have on indexes, tables, views, triggers, functions, aliases, types, and hierarchies.

Table 34. SYSCAT.PACKAGEDEP Catalog View

Column Name	Data Type	Nullable	Description
PKGSCHEMA	VARCHAR(128)		Name of the package.
PKGNAME	CHAR(8)		
UNIQUEID	CHAR(8)		Internal date and time information indicating when the package was first created. Useful for identifying a specific package when multiple packages having the same name exist.
PKGVERSION	VARCHAR(64)		Version identifier of the package.
BINDER	VARCHAR(128)	Yes	Binder of the package.

Table 34. SYSCAT.PACKAGEDEP Catalog View (continued)

Column Name	Data Type	Nullable	Description
BTYPE	CHAR(1)		Type of object BNAME: A = Alias B = Trigger D = Server definition F = Function instance I = Index M = Function mapping N = Nickname O = Privilege dependency on all subtables or subviews in a table or view hierarchy P = Page size R = Structured type S = Materialized query table T = Table U = Typed table V = View W = Typed view
BSCHEMA	VARCHAR(128)		Qualified name of an object on which the package depends.
BNAME	VARCHAR(128)		
TABAUTH	SMALLINT	Yes	If BTYPE is O, S, T, U, V or W then it encodes the privileges that are required by this package (Select, Insert, Delete, Update).

Note: 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

This catalog view contains information about authorizations to query data sources in pass-through sessions. A constraint on the base table requires that the values in SERVER correspond to the values in the SERVER column of SYSCAT.SERVERS. None of the fields in SYSCAT.PASSTHRUAUTH are nullable.

Table 35. Columns in SYSCAT.PASSTHRUAUTHCatalog View

Column Name	Data Type	Nullable	Description
GRANTOR	VARCHAR(128)		Authorization ID of the user who granted the privilege.
GRANTEE	VARCHAR(128)		Authorization ID of the user or group who holds the privilege.
GRANTEETYPE	CHAR(1)		A letter that specifies the type of grantee: U = Grantee is an individual user. G = Grantee is a group.
SERVERNAME	VARCHAR(128)		Name of the data source that the user or group is being granted authorization to.

SYSCAT.SCHEMAAUTH

Contains one or more rows for each user or group who is granted a privilege on a particular schema in the database. All schema privileges for a single schema granted by a specific grantor to a specific grantee appear in a single row.

Table 36. SYSCAT.SCHEMAAUTH Catalog View

Column Name	Data Type	Nullable	Description
GRANTOR	VARCHAR(128)		Authorization ID of the user who granted the privileges or SYSIBM.
GRANTEE	VARCHAR(128)		Authorization ID of the user or group who holds the privileges.
GRANTEETYPE	CHAR(1)		U = Grantee is an individual user. G = Grantee is a group.
SCHEMANAME	VARCHAR(128)		Name of the schema.
ALTERINAUTH	CHAR(1)		Indicates whether grantee holds ALTERIN privilege on the schema: Y = Privilege is held. G = Privilege is held and grantable. N = Privilege is not held.
CREATEINAUTH	CHAR(1)		Indicates whether grantee holds CREATEIN privilege on the schema: Y = Privilege is held. G = Privilege is held and grantable. N = Privilege is not held.
DROPINAUTH	CHAR(1)		Indicates whether grantee holds DROPIN privilege on the schema: Y = Privilege is held. G = Privilege is held and grantable. N = Privilege is not held.

SYSCAT.SCHEMATA

Contains a row for each schema.

Table 37. SYSCAT.SCHEMATA Catalog View

Column Name	Data Type	Nullable	Description
SCHEMANAME	VARCHAR(128)		Name of the schema.
OWNER	VARCHAR(128)		Authorization id of the schema. The value for implicitly created schemas is SYSIBM.
DEFINER	VARCHAR(128)		User who created the schema.
CREATE_TIME	TIMESTAMP		Timestamp indicating when the object was created.
REMARKS	VARCHAR(254)	Yes	User-provided comment.

SYSCAT.SEQUENCEAUTH

Contains a row for each authorization ID that can be used to use or to alter a sequence.

Table 38. SYSCAT.SEQUENCEAUTH Catalog View

Column Name	Data Type	Nullable	Description
GRANTOR	VARCHAR(128)		SYSIBM or authorization ID that granted the privilege.
GRANTEE	VARCHAR(128)		Authorization ID that holds the privilege.
GRANTEETYPE	CHAR(1)		Type of authorization ID that holds the privilege. U = grantee is an individual user
SEQSCHEMA	VARCHAR(128)		Qualified name of the sequence.
SEQNAME	VARCHAR(128)		
USAGEAUTH	CHAR(1)		Y = privilege is held N = privilege is not held G = privilege is held and is grantable
ALTERAUTH	CHAR(1)		Y = privilege is held N = privilege is not held G = privilege is held and is grantable

SYSCAT.SEQUENCES

Contains a row for each sequence or identity column defined in the database.

Table 39. Columns in SYSCAT.SEQUENCES Catalog View

Column Name	Data Type	Nullable	Description
SEQSCHEMA	VARCHAR(128)		Qualified name of the sequence (generated by DB2 for an identity column).
SEQNAME	VARCHAR(128)		
DEFINER	VARCHAR(128)		Definer of the sequence.
OWNER	VARCHAR(128)		Owner of the sequence.
SEQID	INTEGER		Internal ID of the sequence.
SEQTYPE	CHAR(1)		Sequence type S = Regular sequence I = Identity sequence
INCREMENT	DECIMAL(31,0)		Increment value.
START	DECIMAL(31,0)		Starting value.
MAXVALUE	DECIMAL(31,0)		Maximal value.
MINVALUE	DECIMAL(31,0)		Minimum value.
CYCLE	CHAR(1)		Whether cycling will occur when a boundary is reached: Y - cycling will occur N - cycling will not occur
CACHE	INTEGER		Number of sequence values to preallocate in memory for faster access. 0 indicates that values are not preallocated.

Table 39. Columns in SYSCAT.SEQUENCES Catalog View (continued)

Column Name	Data Type	Nullable	Description
ORDER	CHAR(1)		Whether or not the sequence numbers must be generated in order of request: Y - sequence numbers must be generated in order of request N - sequence numbers are not required to be generated in order of request
DATATYPEID	INTEGER		For built-in types, the internal ID of the built-in type. For distinct types, the internal ID of the distinct type.
SOURCETYPEID	INTEGER		For a built-in type, this has a value of 0. For a distinct type, this is the internal ID of the built-in type that is the source type for the distinct type.
CREATE_TIME	TIMESTAMP		Time when the sequence was created.
ALTER_TIME	TIMESTAMP		Time when the last ALTER SEQUENCE statement was executed for this sequence.
PRECISION	SMALLINT		The precision of the data type of the sequence. Values are: 5 for a SMALLINT, 10 for INTEGER, and 19 for BIGINT. For DECIMAL, it is the precision of the specified DECIMAL data type.
ORIGIN	CHAR(1)		Sequence Origin U - User generated sequence S - System generated sequence
REMARKS	VARCHAR(254)	Yes	User supplied comments, or null.

SYSCAT.TABCONST

Each row represents a table constraint of type CHECK, UNIQUE, PRIMARY KEY, or FOREIGN KEY.

Table 40. SYSCAT.TABCONST Catalog View

Column Name	Data Type	Nullable	Description
CONSTNAME	VARCHAR(18)		Name of the constraint (unique within a table).
TABSCHEMA	VARCHAR(128)		Qualified name of the table to which this constraint applies.
TABNAME	VARCHAR(128)		
DEFINER	VARCHAR(128)		Authorization ID under which the constraint was defined.
TYPE	CHAR(1)		Indicates the constraint type: F = Foreign key I = Functional dependency K = Check P = Primary key U = Unique
REMARKS	VARCHAR(254)	Yes	User-supplied comment, or null.
ENFORCED	CHAR(1)		Y = Enforce constraint N = Do not enforce constraint

Table 40. SYSCAT.TABCONST Catalog View (continued)

Column Name	Data Type	Nullable	Description
CHECKEXISTINGDATA	CHAR(1)		D = Defer checking of existing data I = Immediately check existing data N = Never check existing data
ENABLEQUERYOPT	CHAR(1)		Y = Query optimization is enabled N = Query optimization is disabled

SYSCAT.TABLES

Contains one row for each table, view, nickname or alias that is created. All of the catalog tables and views have entries in the SYSCAT.TABLES catalog view.

Table 41. SYSCAT.TABLES Catalog View

Column Name	Data Type	Nullable	Description
TABSCHEMA	VARCHAR(128)		Qualified name of the table, view, nickname, or alias.
TABNAME	VARCHAR(128)		
DEFINER	VARCHAR(128)		User who created the table, view, nickname or alias.
TYPE	CHAR(1)		The type of object: A = Alias H = Hierarchy table N = Nickname S = Materialized query table T = Table U = Typed table V = View W = Typed view
STATUS	CHAR(1)		The check pending status of the object: N = Normal table, view, alias or nickname C = Check pending on table or nickname X = Inoperative view or nickname
DROPRULE	CHAR(1)		N = No rule R = Restrict rule applies on drop
BASE_TABSCHEMA	VARCHAR(128)	Yes	If TYPE = A, these columns identify the table, view, alias, or nickname that is referenced by this alias; otherwise they are null.
BASE_TABNAME	VARCHAR(128)	Yes	
ROWTYPESCHEMA	VARCHAR(128)	Yes	Contains the qualified name of the rowtype of this table, where applicable. Null otherwise.
ROWTYPENAME	VARCHAR(18)		
CREATE_TIME	TIMESTAMP		The timestamp indicating when the object was created.
STATS_TIME	TIMESTAMP	Yes	Last time when any change was made to recorded statistics for this table. Null if no statistics available.

Table 41. SYSCAT.TABLES Catalog View (continued)

Column Name	Data Type	Nullable	Description
COLCOUNT	SMALLINT		Number of columns in the table.
TABLEID	SMALLINT		Internal table identifier.
TBSPACEID	SMALLINT		Internal identifier of primary table space for this table.
CARD	BIGINT		Total number of rows in the table. For tables in a table hierarchy, the number of rows at the given level of the hierarchy; –1 if statistics are not gathered, or the row describes a view or alias; –2 for hierarchy tables (H-tables).
NPAGES	INTEGER		Total number of pages on which the rows of the table exist; –1 if statistics are not gathered, or the row describes a view or alias; –2 for subtables or H-tables.
FPAGES	INTEGER		Total number of pages; –1 if statistics are not gathered, or the row describes a view or alias; –2 for subtables or H-tables.
OVERFLOW	INTEGER		Total number of overflow records in the table; –1 if statistics are not gathered, or the row describes a view or alias; –2 for subtables or H-tables.
TBSPACE	VARCHAR(18)	Yes	Name of primary table space for the table. If no other table space is specified, all parts of the table are stored in this table space. Null for aliases and views.
INDEX_TBSPACE	VARCHAR(18)	Yes	Name of table space that holds all indexes created on this table. Null for aliases and views, 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(18)	Yes	Name of table space that holds all long data (LONG or LOB column types) for this table. Null for aliases and views, 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	Yes	Number of parent tables of this table (the number of referential constraints in which this table is a dependent).
CHILDREN	SMALLINT	Yes	Number of dependent tables of this table (the number of referential constraints in which this table is a parent).
SELFREFS	SMALLINT	Yes	Number of self-referencing referential constraints for this table (the number of referential constraints in which this table is both a parent and a dependent).
KEYCOLUMNS	SMALLINT	Yes	Number of columns in the primary key of the table.
KEYINDEXID	SMALLINT	Yes	Index ID of the primary index. This field is null or 0 if there is no primary key.

Table 41. SYSCAT.TABLES Catalog View (continued)

Column Name	Data Type	Nullable	Description
KEYUNIQUE	SMALLINT		Number of unique constraints (other than primary key) defined on this table.
CHECKCOUNT	SMALLINT		Number of check constraints defined on this table.
DATACAPTURE	CHAR(1)		Y = Table participates in data replication N = Does not participate L = Table participates in data replication, including replication of LONG VARCHAR and LONG VARGRAPHIC columns
CONST_CHECKED	CHAR(32)		Byte 1 represents foreign key constraints. Byte 2 represents check constraints. Byte 5 represents materialized query table. Byte 6 represents generated columns. Byte 7 represents staging table. Other bytes are reserved. Encodes constraint information on checking. Values: Y = Checked by system U = Checked by user N = Not checked (pending) W = Was in a 'U' state when the table was placed in check pending (pending) 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.
PMAP_ID	SMALLINT	Yes	Identifier of the partitioning map used by this table. Null for aliases and views.
PARTITION_MODE	CHAR(1)		Mode used for tables in a partitioned database. H = Hash on the partitioning key R = Table replicated across database partitions Blank for aliases, views and tables in single partition database partition groups with no partitioning key defined. Also blank for nicknames.
LOG_ATTRIBUTE	CHAR(1)		0 = Default logging 1 = Table created not logged initially
PCTFREE	SMALLINT		Percentage of each page to be reserved for future inserts. Can be changed by ALTER TABLE.

Table 41. SYSCAT.TABLES Catalog View (continued)

Column Name	Data Type	Nullable	Description
APPEND_MODE	CHAR(1)		Controls how rows are inserted on pages: N = New rows are inserted into existing spaces if available Y = New rows are appended at end of data Initial value is N.
REFRESH	CHAR(1)		Refresh mode: D = Deferred I = Immediate O = Once Blank if not a materialized query table
REFRESH_TIME	TIMESTAMP	Yes	For REFRESH = D or O, timestamp of the REFRESH TABLE statement that last refreshed the data. Otherwise null.
LOCKSIZE	CHAR(1)		Indicates preferred lock granularity for tables when accessed by DML statements. Only applies to tables. Possible values are: R = Row T = Table Blank if not applicable Initial value is R.
VOLATILE	CHAR(1)		C = Cardinality of the table is volatile Blank if not applicable
ROW_FORMAT	CHAR(1)		Not used.
PROPERTY	VARCHAR(32)		Properties for the table. A single blank indicates that the table has no properties.
STATISTICS_PROFILE	CLOB(32K)	Yes	RUNSTATS command used to register a statistical profile of the table.
COMPRESSION	CHAR(1)		V = Value compression is activated, and a row format that supports compression is used N = No compression. A row format that does not support compression is used
ACCESS_MODE	CHAR(1)		Access mode of the object. This access mode is used in conjunction with the STATUS field to represent one of four states. Possible values are: N = No access (corresponds to a status value of C) R = Read-only (corresponds to a status value of C) D = No data movement (corresponds to a status value of N) F = Full access (corresponds to a status value of N)

Table 41. SYSCAT.TABLES Catalog View (continued)

Column Name	Data Type	Nullable	Description
CLUSTERED	CHAR(1)	Yes	Y = multi-dimensional clustering (MDC) table Null for a non-MDC table
ACTIVE_BLOCKS	INTEGER	Yes	Total number of in-use blocks in an MDC table; –1 if statistics are not gathered.
MAXFREESPACESEARCH	SMALLINT		Reserved for future use.
REMARKS	VARCHAR(254)	Yes	User-provided comment.

SYSCAT.TABLESPACES

Contains a row for each table space.

Table 42. SYSCAT.TABLESPACES Catalog View

Column Name	Data Type	Nullable	Description
TBSPACE	VARCHAR(18)		Name of the table space.
DEFINER	VARCHAR(128)		Authorization ID of the table space definer.
CREATE_TIME	TIMESTAMP		Creation time of the table space.
TBSPACEID	INTEGER		Internal table space identifier.
TBSPACETYPE	CHAR(1)		The type of table space: S = System managed space D = Database managed space
DATATYPE	CHAR(1)		The type of data that can be stored: A = All types of permanent data L = Large data - long data or index data T = System temporary tables only U = Declared temporary tables only
EXTENTSIZE	INTEGER		Size of 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 prefetch is performed; -1 if prefetch size is AUTOMATIC.
OVERHEAD	DOUBLE		Controller overhead and disk seek and latency time, in milliseconds.
TRANSFERRATE	DOUBLE		Time to read one page of size PAGESIZE into the buffer.
PAGESIZE	INTEGER		Size (in bytes) of pages in the table space.
DBPGNAME	VARCHAR(18)		Name of the database partition group for the table space.
BUFFERPOOLID	INTEGER		ID of buffer pool used by this table space; 1 indicates the default buffer pool.
DROP_RECOVERY	CHAR(1)		N = table is not recoverable after a DROP TABLE statement Y = table is recoverable after a DROP TABLE statement

Table 42. SYSCAT.TABLESPACES Catalog View (continued)

Column Name	Data Type	Nullable	Description
REMARKS	VARCHAR(254)	Yes	User-provided comment.

SYSCAT.TBSPACEAUTH

Contains one row for each user or group who is granted USE privilege on a particular table space in the database.

Table 43. SYSCAT.TBSPACEAUTH Catalog View

Column Name	Data Type	Nullable	Description
GRANTOR	CHAR(128)		Authorization ID of the user who granted the privileges or SYSIBM.
GRANTEE	CHAR(128)		Authorization ID of the user or group who holds the privileges.
GRANTEETYPE	CHAR(1)		U = Grantee is an individual user. G = Grantee is a group.
TBSPACE	VARCHAR(18)		Name of the table space.
USEAUTH	CHAR(1)		Indicates whether grantee holds USE privilege on the table space: G = Privilege is held and grantable. N = Privilege is not held. Y = Privilege is held.

SYSCAT.USEROPTIONS

Each row contains server specific option values.

Table 44. SYSCAT.USEROPTIONS Catalog View

Column Name	Data Type	Nullable	Description
AUTHID	VARCHAR(128)		Local authorization ID (always uppercase)
SERVERNAME	VARCHAR(128)		Name of the server for which the user is defined.
OPTION	VARCHAR(128)		Name of the user options.
SETTING	VARCHAR(255)		Value.

SYSCAT.TABAUTH

Contains one or more rows for each user or group who is granted a privilege on a particular table or view in the database. All the table privileges for a single table or view granted by a specific grantor to a specific grantee appear in a single row.

Table 45. SYSCAT.TABAUTH Catalog View

Column Name	Data Type	Nullable	Description
GRANTOR	VARCHAR(128)		Authorization ID of the user who granted the privileges or SYSIBM.
GRANTEE	VARCHAR(128)		Authorization ID of the user or group who holds the privileges.

Table 45. SYSCAT.TABAUTH Catalog View (continued)

Column Name	Data Type	Nullable	Description
GRANTEETYPE	CHAR(1)		U = Grantee is an individual user. G = Grantee is a group.
TABSCHEMA	VARCHAR(128)		Qualified name of the table or view.
TABNAME	VARCHAR(128)		
CONTROLAUTH	CHAR(1)		Indicates whether grantee holds CONTROL privilege on the table or view: Y = Privilege is held. N = Privilege is not held.
ALTERAUTH	CHAR(1)		Indicates whether grantee holds ALTER privilege on the table: Y = Privilege is held. N = Privilege is not held. G = Privilege is held and grantable.
DELETEAUTH	CHAR(1)		Indicates whether grantee holds DELETE privilege on the table or view: Y = Privilege is held. N = Privilege is not held. G = Privilege is held and grantable.
INDEXAUTH	CHAR(1)		Indicates whether grantee holds INDEX privilege on the table: Y = Privilege is held. N = Privilege is not held. G = Privilege is held and grantable.
INSERTAUTH	CHAR(1)		Indicates whether grantee holds INSERT privilege on the table or view: Y = Privilege is held. N = Privilege is not held. G = Privilege is held and grantable.
SELECTAUTH	CHAR(1)		Indicates whether grantee holds SELECT privilege on the table or view: Y = Privilege is held. N = Privilege is not held. G = Privilege is held and grantable.
REFAUTH	CHAR(1)		Indicates whether grantee holds REFERENCE privilege on the table or view: Y = Privilege is held. N = Privilege is not held. G = Privilege is held and grantable.
UPDATEAUTH	CHAR(1)		Indicates whether grantee holds UPDATE privilege on the table or view: Y = Privilege is held. N = Privilege is not held. G = Privilege is held and grantable.

Chapter 11. Other security considerations

Introduction to 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.

Related concepts:

- "Screening router firewalls" on page 209
- "Application proxy firewalls" on page 210
- "Circuit level firewalls" on page 210
- "Stateful multi-layer inspection (SMLI) firewalls" on page 210

Screening router firewalls

This type of 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® Universal Database (DB2 UDB) are open for incoming and outgoing packets. DB2 UDB uses port 523 for the DB2 Administration Server (DAS), which is used by the DB2 UDB 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.

Related concepts:

- "Introduction to firewall support" on page 209

© Copyright IBM Corp. 1993-2004

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 Universal Database™ (DB2 UDB) server on the firewall, acting as a hop server to the final destination server, acts like an application proxy.

Related concepts:

- "Introduction to firewall support" on page 209

Circuit level firewalls

This type of 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.

DB2® Universal Database (DB2 UDB) supports SOCKS Version 4.

Related concepts:

- "Introduction to firewall support" on page 209

Stateful multi-layer inspection (SMLI) firewalls

This type of firewall is 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.

Related concepts:

- "Introduction to firewall support" on page 209

Guidelines for stored procedures

Stored procedures permit one call to a remote database to execute a preprogrammed procedure in a database application environment in which many situations are repetitive. For example, for receiving a fixed set of data, performing the same set of multiple requests against a database, or returning a fixed set of data might represent several accesses to the database.

Processing a single SQL statement for a remote database requires sending two transmissions: one request and one receive. Because an application contains many SQL statements it requires many transmissions to complete its work.

However, when a database client uses a stored procedure that encapsulates many SQL statements, it requires only two transmissions for the entire process.

Stored procedures usually run in processes separate from the database agents. This separation requires the stored procedure and agent processes to communicate through a router. However, a special kind of stored procedure that runs in the agent process might improve performance, although it carries significant risks of corrupting data and databases.

These risky stored procedures are those created as *not fenced*. For a not-fenced stored procedure, nothing separates the stored procedure from the database control structures that the database agent uses. If a DBA wants to ensure that the stored procedure operations will not accidentally or maliciously damage the database control structures, the *not fenced* option is omitted.

Because of the risk of damaging your database, use *not fenced* stored procedures **only** when you need the maximum possible performance benefits. In addition, make absolutely sure that the procedure is well coded and has been thoroughly tested before allowing it to run as a not-fenced stored procedure. If a fatal error occurs while running a not-fenced stored procedure, the database manager determines whether the error occurred in the application or database manager code and performs the appropriate recovery.

A not-fenced stored procedure can corrupt the database manager beyond recovery, possibly resulting in lost data and the possibility of a corrupt database. Exercise extreme caution when you run not-fenced trusted stored procedures. In almost all cases, the proper performance analysis of an application results in the good performance without using not-fenced stored procedures. For example, triggers might improve performance.

Related concepts:

- "Query tuning guidelines" in the *Administration Guide: Performance*

Chapter 12. 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.

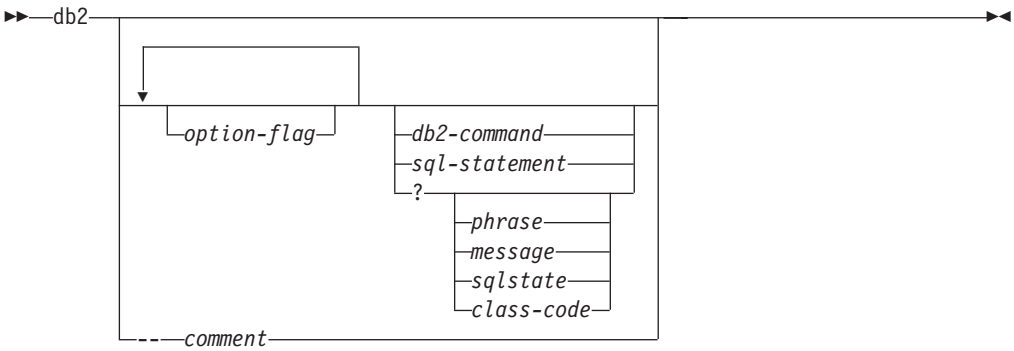
Note: On Windows, db2cmd opens the CLP-enabled DB2 window, and initializes the DB2 command line environment. Issuing this command is equivalent to clicking on the *DB2 Command Window* icon.

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.

Note: 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.

© Copyright IBM Corp. 1993-2004

? 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.

Note: In each case, a blank space must separate the question mark (?) from the variable name.

Related concepts:

- "Command Line Processor (CLP)" on page 221

Related reference:

- "Command line processor options" on page 214
- "Command Line Processor Return Codes" on page 220

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 46 on page 215 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 46. 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
-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
-l*filename*	This option tells the command line processor to log commands in a history file.	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
-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*	This option tells the command line processor to define and to use *x* as the statement termination character.	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 SQL statement warning messages.	ON
-x	This option tells the command line processor to return data without any headers, including column names.	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.

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 other options are combined with option -f, option -f must be specified last. For example:

```
db2 -tvf filename
```

Note: 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 QUIT or TERMINATE 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.

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.

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.

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.

Note: 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 the command 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.

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 47. 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 Option (-t):

The -t option tells the command line processor to use a semicolon (;) as the statement termination character, and disables the backslash (\) line continuation character.

Note: This option cannot be changed from within the interactive mode.

The default setting for this command option is OFF (+t or -t-).

To define a termination character, use -td followed by the chosen termination character. For example, -tdx sets x as the statement termination character.

The termination character cannot be used to concatenate multiple statements from the command line, since only the last non-blank character on each input line is checked for a termination symbol.

The statement termination character option does not affect any other command 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 tells the command line processor to show SQL statement warning messages.

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.

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.

Related reference:

- "db2 - Command Line Processor Invocation" on page 213
- "Command Line Processor Return Codes" on page 220

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 a DB2 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.

Related reference:

- "db2 - Command Line Processor Invocation" on page 213
- "Command line processor options" on page 214

Command Line Processor (CLP)

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.

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 Universal Database™ (UDB) 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 48. 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
%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 run-time, the empty string is substituted for that token.
- The interactive input prompt will always present the authorization IDs, database names, and instance names in upper case, 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.

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 ″tyronnem″. DB2INSTANCE set to ″DB2″. DB2DBDFT set to ″SAMPLE″.
  ```
  (TYRONNEM@DB2, @SAMPLE)
  ```
- Database connection to database ″sample″ with authorization ID ″horman″. DB2INSTANCE set to ″DB2″. DB2DBDFT set to ″SAMPLE″.
  ```
  (@DB2, HORMAN@SAMPLE)
  ```
- Instance attachment to instance ″DB2″ with authorization ID ″tyronnem″. Database connection to database ″sample″ with authorization ID ″horman″. DB2INSTANCE set to ″DB2″. DB2DBDFT not set.
  ```
  (TYRONNEM@DB2, HORMAN@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 49. Environment Variables

Variable	Minimum	Maximum	Default
DB2BQTIME	1 second	5294967295	1 second
DB2BQTRY	0 tries	5294967295	60 tries
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 upper case 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.

Chapter 13. DB2 UDB Commands for Administrators

BACKUP DATABASE

Creates a backup copy of a database or a table space.

Scope:

This command only affects the database partition on which it is executed.

Authorization:

One of the following:

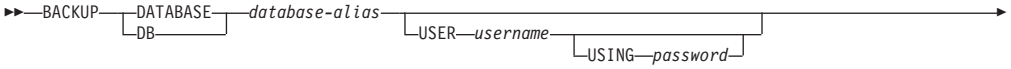
- *sysadm*
- *sysctrl*
- *sysmaint*

Required connection:

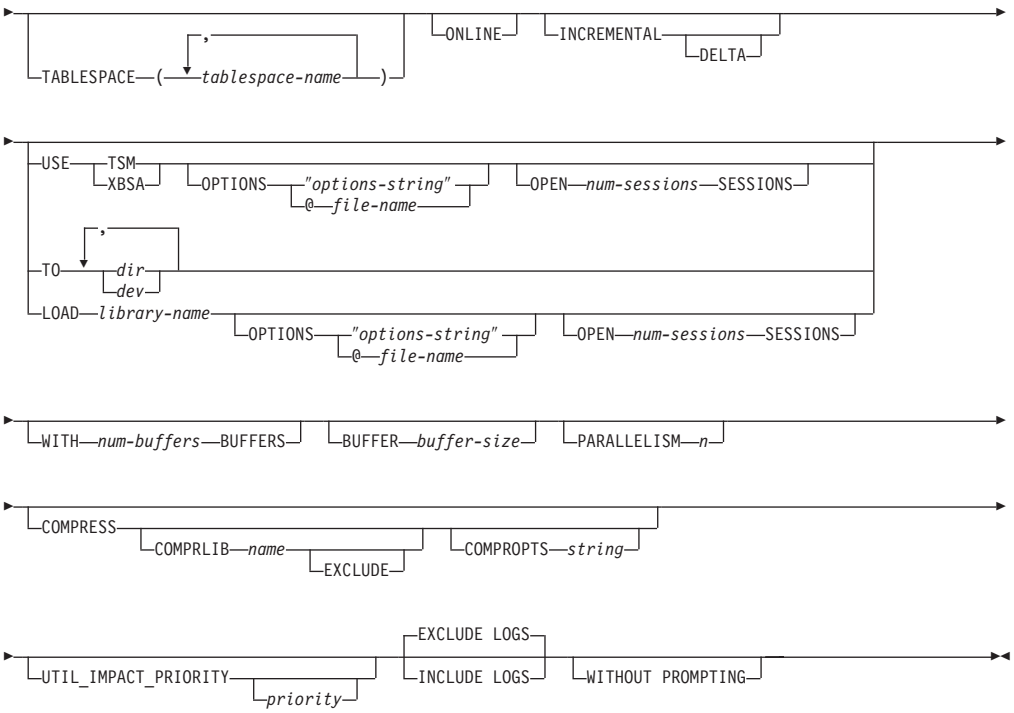
Database. This command automatically establishes a connection to the specified database.

Note: If a connection to the specified database already exists, that connection will be terminated and a new connection established specifically for the backup operation. The connection is terminated at the completion of the backup operation.

Command syntax:



© Copyright IBM Corp. 1993-2004



Command parameters:

DATABASE database-alias
Specifies the alias of the database to back up.

USER username
Identifies the user name under which to back up the database.

USING password
The password used to authenticate the user name. If the password is omitted, the user is prompted to enter it.

TABLESPACE tablespace-name
A list of names used to specify the table spaces to be backed up.

ONLINE
Specifies online backup. The default is offline backup. Online backups are only available for databases configured with *logretain* or *userexit* enabled. During an online backup, DB2 obtains IN (Intent None) locks on all tables existing in SMS table spaces as they are processed and S (Share) locks on LOB data in SMS table spaces.

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 output.

USE 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.

OPTIONS

"options-string"
Specifies options to be used for the backup operation.The string will be passed to the vendor support library, for example TSM, exactly as it was entered, without the quotes.

Note: Specifying this option overrides the value specified by the VENDOROPT database configuration parameter.

@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, for example TSM. 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.

Note: 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. If USE TSM, TO, and LOAD are omitted, the default target directory for the backup image is the current working directory of the client computer. This target directory or device must exist on the database server. 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. To back up to the FAT file system on Windows operating systems, users must conform to the 8.3 naming restriction.

Use of tape devices or floppy disks might generate messages and prompts for user input. Valid response options are:

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 *only* the device that generated the warning message (for example, when there are no more tapes)

t Terminate. Abort the backup operation.

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.

When using tape devices on SCO UnixWare 7, specify a buffer size of 16.

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. 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.

Note: When performing an offline backup operation, logs are excluded whether or not this option is specified.

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.

WITHOUT PROMPTING
Specifies that the backup will run unattended, and that any actions which normally require user intervention will return an error message.

Examples:

1. In the following example, the database WSDB is defined on all 4 partitions, numbered 0 through 3. The path /dev3/backup is accessible from all partitions. Partition 0 is the catalog partition, and needs to be backed-up separately since this is an offline backup. To perform an offline backup of all the WSDB database partitions to /dev3/backup, issue the following commands from one of the database partitions:

   ```
   db2_all '<<+0< db2 BACKUP DATABASE wsdb TO /dev3/backup'
   db2_all '|<<-0< db2 BACKUP DATABASE wsdb TO /dev3/backup'
   ```

   In the second command, the db2_all utility will issue the same backup command to each database partition in turn (except partition 0). All four database partition backup images will be stored in the /dev3/backup directory.

2. In the following example database SAMPLE is backed up to a TSM server using two concurrent TSM client sessions. DB2 calculates the optimal buffer size for this environment.

   ```
   db2 backup database sample use tsm open 2 sessions with 4 buffers
   ```

3. In the following example, a table space-level backup of table spaces (syscatspace, userspace1) of database `payroll` is done to tapes.

   ```
   db2 backup database payroll tablespace (syscatspace, userspace1) to
     /dev/rmt0, /dev/rmt1 with 8 buffers without prompting
   ```

4. The USE TSM OPTIONS keywords can be used to specify the TSM information to use for the backup operation. The following example shows how to use the USE TSM OPTIONS keywords to specify a fully qualified file name:

   ```
   db2 backup db sample use TSM options @/u/dmcinnis/myoptions.txt
   ```

   The file myoptions.txt contains the following information: -fromnode=bar -fromowner=dmcinnis

5. 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) db2 backup db sample use tsm
(Mon) db2 backup db sample online incremental delta use tsm
(Tue) db2 backup db sample online incremental delta use tsm
(Wed) db2 backup db sample online incremental use tsm
(Thu) db2 backup db sample online incremental delta use tsm
(Fri) db2 backup db sample online incremental delta use tsm
(Sat) db2 backup db sample online incremental use tsm
```

6. In the following example, three identical target directories are specified for a backup operation on database SAMPLE. You might want to do this if the target file system is made up of multiple physical disks.

```
db2 backup database sample to /dev3/backup, /dev3/backup, /dev3/backup
```

   The data will be concurrently backed up to the three target directories, and three backup images will be generated with extensions `.001`, `.002`, and `.003`.

Related reference:

- "RESTORE DATABASE" on page 354
- "ROLLFORWARD DATABASE" on page 363

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:

- *sysadm* or *dbadm* authority
- BINDADD privilege if a package does not exist and one of:
  - IMPLICIT_SCHEMA authority on the database if the schema name of the package does not exist
  - CREATEIN privilege on the schema if the schema name of the package exists
- ALTERIN privilege on the schema if the package exists
- BIND privilege on the package if it exists.

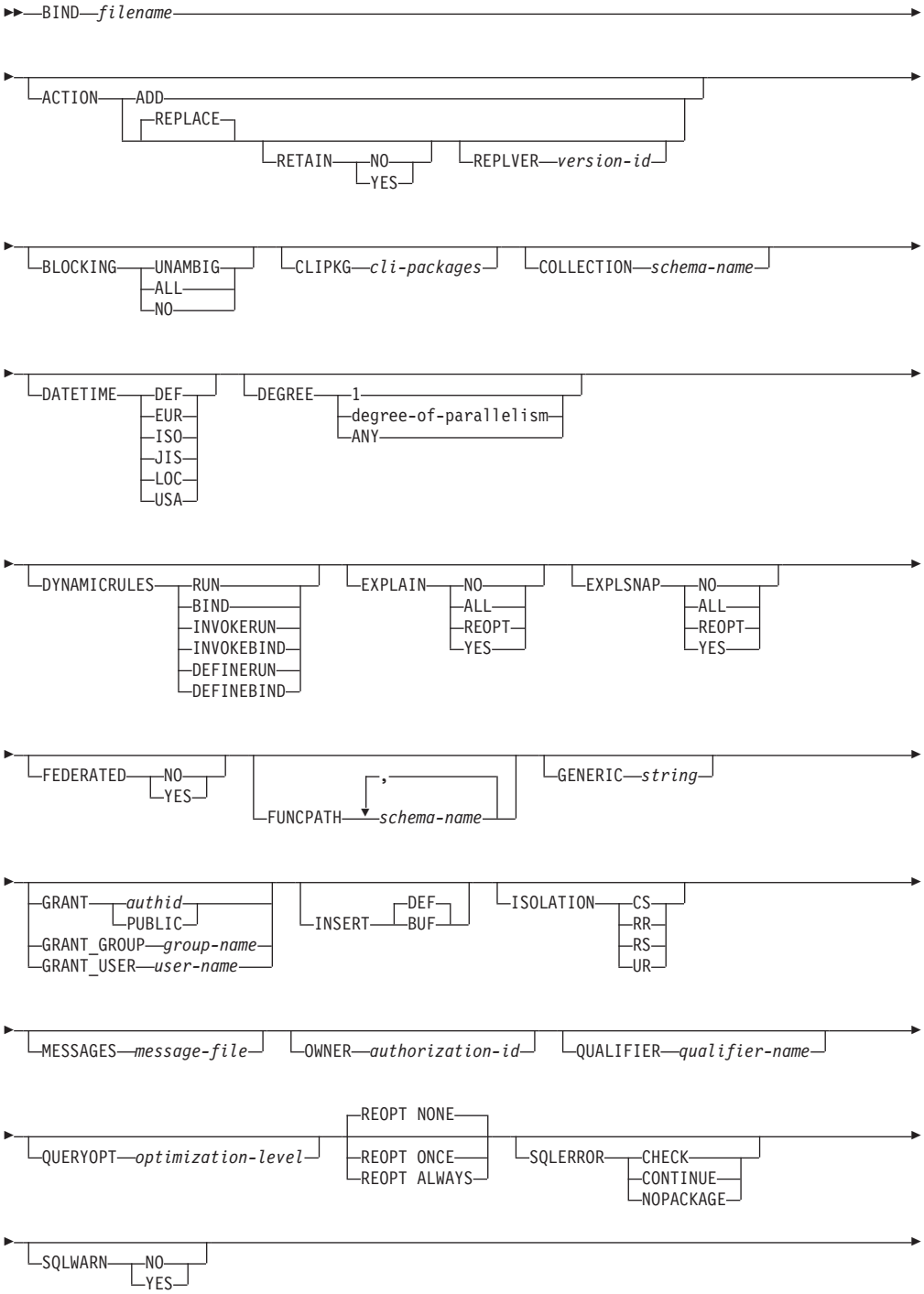
The user also needs all privileges required to compile any static SQL statements in the application. Privileges granted to groups are not used for authorization checking of static statements. If the user has *sysadm* authority, but not explicit privileges to complete the bind, the database manager grants explicit *dbadm* authority automatically.

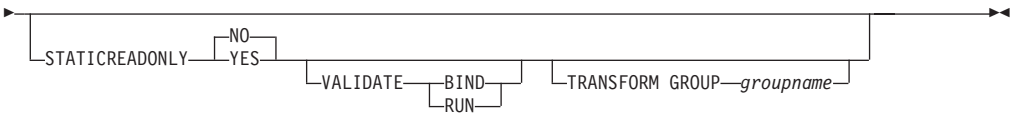
Required connection:

Database. If implicit connect is enabled, a connection to the default database is established.

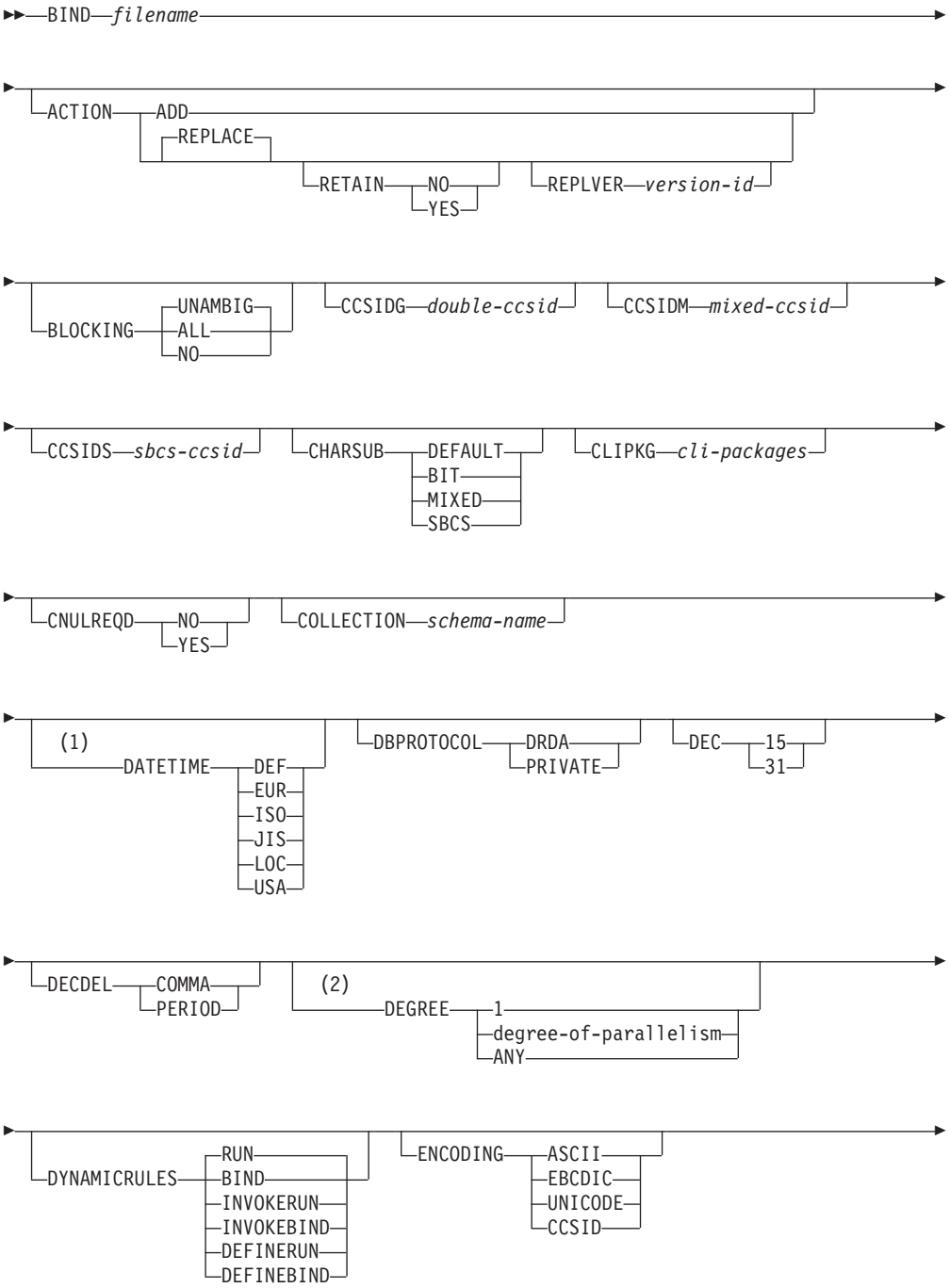
Command syntax:

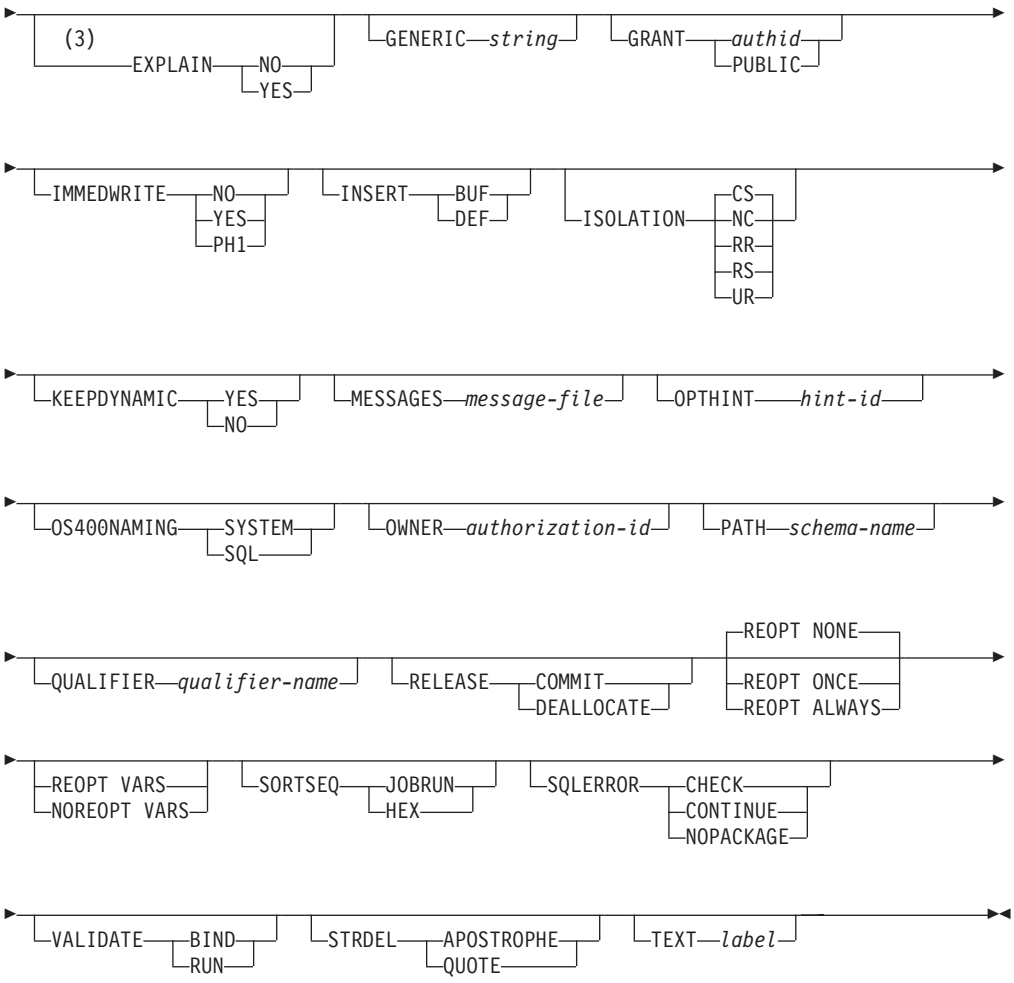
For DB2 for Windows and UNIX





For DB2 on servers other than Windows and UNIX





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.

BLOCKING
Specifies the type of row blocking for cursors.

ALL Specifies to block for:

- Read-only cursors
- Cursors not specified as FOR UPDATE OF

Ambiguous cursors are treated as read-only.

NO Specifies not to block any cursors. Ambiguous cursors are treated as updatable.

UNAMBIG
Specifies to block for:

- Read-only cursors
- Cursors not specified as FOR UPDATE OF

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 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.

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 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.

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 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.

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 for Windows and UNIX.

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 for Windows and UNIX.

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 30-character collection identifier for the package. If not specified, the authorization identifier for the user processing the package is used.

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 for Windows and UNIX. 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 for Windows and UNIX. 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.

Note: 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).

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).

Note: 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).

Note: This value is not supported by DRDA

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.

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.

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 number of schemas that can be specified is limited by the length of the resulting function path, which cannot exceed 254 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 1023 bytes.

GRANT

authid Grants EXECUTE and BIND privileges to a specified user name or group ID.

PUBLIC
Grants EXECUTE and BIND privileges to PUBLIC.

GRANT_GROUP group-name
Grants EXECUTE and BIND privileges to a specified group ID.

GRANT_USER user-name
Grants EXECUTE and BIND privileges to a specified user name.

Note: If more than one of the GRANT, GRANT_GROUP, and GRANT_USER options are specified, only the last option specified is executed.

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 for Windows and UNIX.

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 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.

OS400NAMING
Specifies which naming option is to be used when accessing DB2 UDB for iSeries data. Supported by DB2 UDB for iSeries only. For a list of supported option values, refer to the documentation for DB2 for iSeries.

Please note that 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 iSeries 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 iSeries system naming convention is used, whether or not it has been precompiled or bound using the OS400NAMING SYSTEM option.

OWNER authorization-id
Designates a 30-character 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 SYSADM or 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.

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 30-character 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 for Windows and UNIX.

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 the iSeries system. Supported by DB2 UDB for iSeries only. For a list of supported option values, refer to the documentation for DB2 for iSeries.

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, and special registers. Valid values are:

NONE
The access path for a given SQL statement containing host variables, parameter markers, 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, 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, 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 back-level compatibility. Specifies whether to have DB2 determine an access path at run time using values for host 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). This DB2 precompile/bind option is not supported by DRDA.

NO Warnings will not be returned from the SQL compiler.

YES Warnings will be returned from the SQL compiler.

Note: 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.

YES Any static cursor that does not contain the FOR UPDATE

or FOR READ ONLY clause will be considered READ ONLY. This is the default value.

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 for Windows and UNIX. 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 for Windows and UNIX.

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 characters 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.

Related concepts:

- "Isolation levels" in the *SQL Reference, Volume 1*
- "Authorization Considerations for Dynamic SQL" on page 951
- "Effect of DYNAMICRULES bind option on dynamic SQL" on page 952
- "Effects of REOPT on static SQL" in the *Application Development Guide: Programming Client Applications*
- "Effects of REOPT on dynamic SQL" in the *Application Development Guide: Programming Client Applications*

Related reference:

- "SET CURRENT QUERY OPTIMIZATION statement" in the *SQL Reference, Volume 2*
- "PRECOMPILE" on page 842
- "DB2 CLI bind files and package names" in the *CLI Guide and Reference, Volume 1*
- "Special registers" on page 797
- "Datetime values" in the *SQL Reference, Volume 1*

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 node.

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:

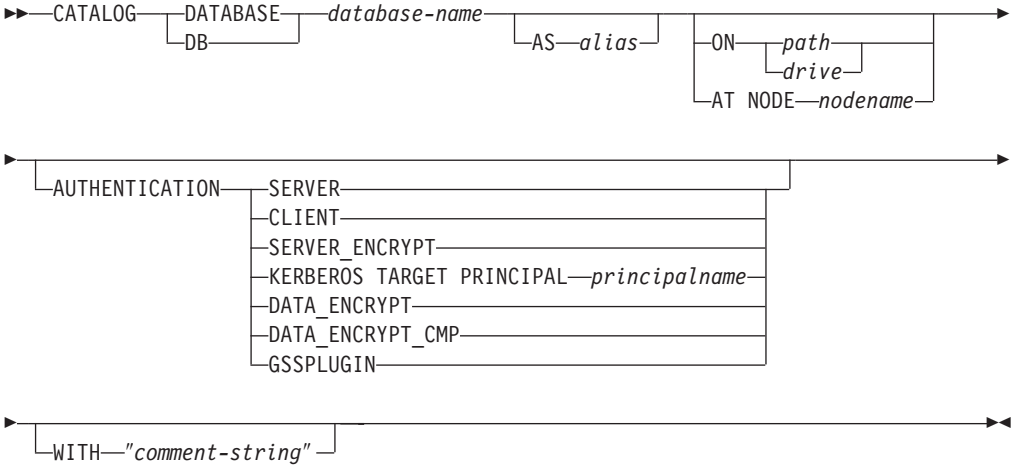
One of the following:

- *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
On UNIX based systems, specifies the path on which the database being cataloged resides. On Windows operating systems, specifies the letter of the drive on which the database being cataloged resides.

AT NODE nodename
Specifies the name of the node 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 node containing the target database.

CLIENT
Specifies that authentication takes place on the node where the application is invoked.

SERVER_ENCRYPT
Specifies that authentication takes place on the node containing the target database, and that passwords are encrypted at the source. 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. When authentication is Kerberos, and an APPC connection is used for access, only SECURITY=NONE is supported.

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 node containing the target database, and that connections must use data encryption.

DATA_ENCRYPT_CMP
Specifies that authentication takes place on the node containing the target database, and that connections must use data encryption. This option is provided for compatibility with downlevel products that do not support data encryption, in which case they will be allowed to connect with SERVER_ENCRYPT and not encrypt user data. Any product that does support data encryption will be forced to use it.

GSSPLUGIN
Specifies that authentication takes place using an external GSS API-based plug-in security mechanism. When

authentication is GSSPLUGIN, and an APPC connection is used for access, only SECURITY=NONE is supported.

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 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.

If neither path nor node 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 node as the database manager instance are cataloged as *indirect* entries. Databases on other nodes 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.

Note: If directory caching is enabled, database and node 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 might not be effective until the application has restarted.

To refresh the CLP's directory cache, use the TERMINATE command. To refresh DB2'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.

Related reference:

• "GET DATABASE MANAGER CONFIGURATION" on page 281

- "LIST DATABASE DIRECTORY Command" in the *Command Reference*
- "TERMINATE Command" in the *Command Reference*
- "UNCATALOG DATABASE Command" in the *Command Reference*

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. When you initialize a new database you can specify the AUTOCONFIGURE option to display and optionally apply the initial values for the buffer pool size, database and database manager parameters. The AUTOCONFIGURE option is not available in a partitioned database environment.

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:

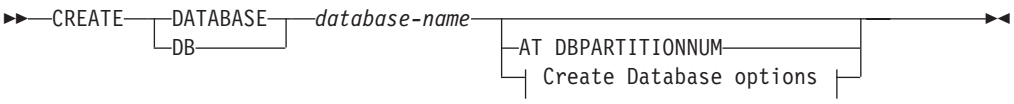
One of the following:

- *sysadm*
- *sysctrl*

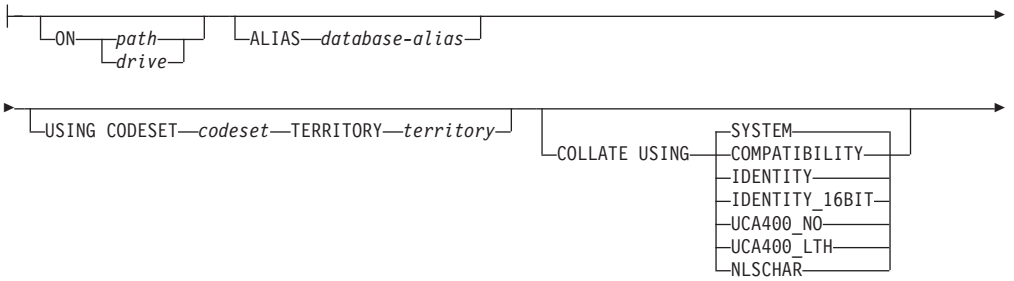
Required connection:

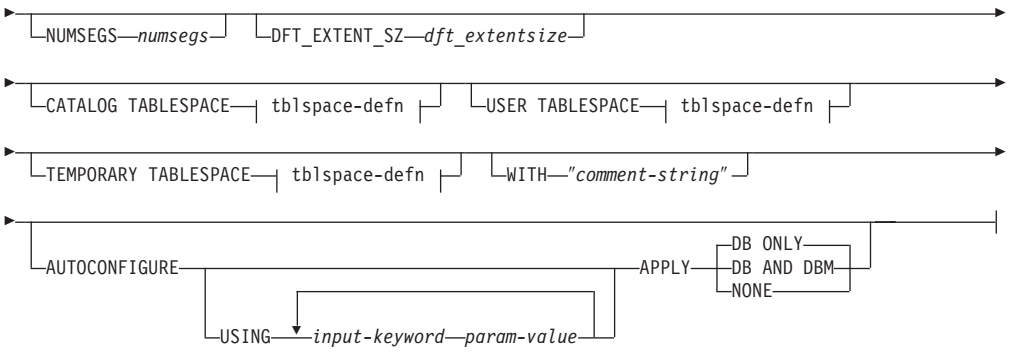
Instance. To create a database at another (remote) node, it is necessary to first attach to that node. A database connection is temporarily established by this command during processing.

Command syntax:

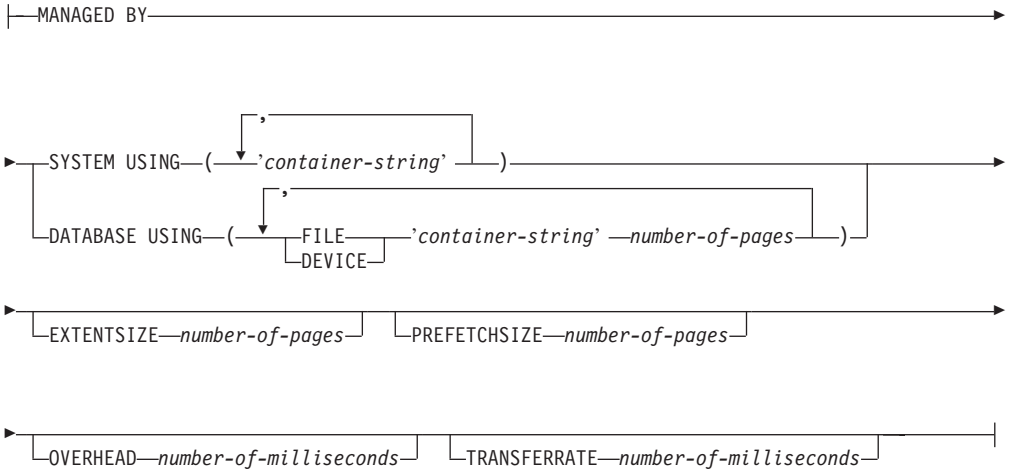


Create Database options:





tblspace-defn:



Notes:

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. In a partitioned database environment, use of the AUTOCONFIGURE option will result in failure of the CREATE DATABASE command. If you want to use the AUTOCONFIGURE option in a partitioned database environment, first create the database without specifying the AUTOCONFIGURE option, then run the AUTOCONFIGURE command on each partition.
5. 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.

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 DBPARITIONNUM option, the database at this partition is in the restore-pending state. You must immediately restore the database on this node. This parameter is not intended for general use. For example, it should be used with RESTORE DATABASE command if the database partition at a node was damaged and must be re-created. Improper use of this parameter can cause inconsistencies in the system, so it should only be used with caution.

ON path/drive
On UNIX based 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). Maximum length is 205 characters. On the Windows operating system, specifies the letter of the drive on which to create the database.

Note: 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.

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 to be used for data entered into this database. After you create the database, you cannot change the specified territory.

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.

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.

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 Corsortium 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 based on the Unicode Standard version 4.00 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_LTH
The UCA (Unicode Collation Algorithm) collation sequence based on the Unicode Standard version 4.00, 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. Note that this collator might order Thai data differently from the NLSCHAR collator option.

NLSCHAR
System-defined collating sequence using the unique collation rules for the specific code set/territory.

Note: 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
Collating sequence based on the database territory. This option cannot be specified when creating a Unicode database.

NUMSEGS numsegs
Specifies the number of segment directories that will be created and used to store DAT, IDX, LF, LB, and LBA files for any default SMS table spaces. This parameter does not affect 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.

CATALOG TABLESPACE tblspace-defn
Specifies the definition of the table space which will hold the catalog tables, SYSCATSPACE. If not specified, SYSCATSPACE will be 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
```

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, USERSPACE1 will be 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/SQLT0001.1
/u/smith/smith/NODE0000/SQL00001/SQLT0001.2
/u/smith/smith/NODE0000/SQL00001/SQLT0001.3
/u/smith/smith/NODE0000/SQL00001/SQLT0001.4
```

TEMPORARY TABLESPACE tblspace-defn
Specifies the definition of the initial system temporary table space, TEMPSPACE1. If not specified, TEMPSPACE1 will be 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/SQLT0002.1
/u/smith/smith/NODE0000/SQL00001/SQLT0002.2
/u/smith/smith/NODE0000/SQL00001/SQLT0002.3
/u/smith/smith/NODE0000/SQL00001/SQLT0002.4
```

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.

USING input-keyword param-value

Table 50. Valid input keywords and parameter values

Keyword	Valid values	Default value	Explanation
mem_percent	1–100	25	Percentage of memory to dedicate. If other applications (other than the operating system) are running on this server, set this to less than 100.
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.

Table 50. Valid input keywords and parameter values (continued)

Keyword	Valid values	Default value	Explanation
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
Displays the recommended changes, but does not apply them.

Usage notes:

The CREATE DATABASE command:

- Creates a database in the specified subdirectory. In partitioned database environment, creates the database on all database partitions listed in `db2nodes.cfg`, and creates a `$DB2INSTANCE/NODE`*xxxx* directory under the specified subdirectory at each database partition. In a non-partitioned environment, creates a `$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.
- 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.

  Note: The utilities bind file list contains two bind files that cannot be bound against down-level servers:
  - `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 down-level server, 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 authority, and CONNECT, CREATETAB, BINDADD, CREATE_NOT_FENCED, IMPLICIT_SCHEMA and LOAD privileges to the database creator
  - CONNECT, CREATETAB, BINDADD, and IMPLICIT_SCHEMA privileges 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.

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.

Compatibilities:

For compatibility with versions earlier than Version 8:

- The keyword NODE can be substituted for DBPARTITIONNUM.

Related concepts:

- "Isolation levels" in the *SQL Reference, Volume 1*
- "Unicode implementation in DB2 Universal Database" in the *Administration Guide: Planning*

Related tasks:

- "Collating Thai characters" in the *Administration Guide: Planning*
- "Creating a database" on page 133

Related reference:

- "CREATE TABLESPACE" on page 648
- "sqlecrea - Create Database" on page 500
- "BIND" on page 232
- "CATALOG DATABASE" on page 249
- "DROP DATABASE" on page 268
- "RESTORE DATABASE" on page 354
- "CATALOG LDAP DATABASE Command" in the *Command Reference*

- "AUTOCONFIGURE Command" in the *Command Reference*

db2audit - Audit Facility Administrator Tool

DB2 provides 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 an audit log file. 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 an instance level, recording all instance level activities and database level activities.

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.
- Stop recording auditable events within the DB2 instance.
- Configure the behavior of the audit facility.
- Select the categories of the auditable events to be recorded.
- Request a description of the current audit configuration.
- Flush any pending audit records from the instance and write them to the audit log.
- Extract audit records by formatting and copying them from the audit log to a flat file or ASCII delimited files. Extraction is done for one of two reasons: In preparation for analysis of log records, or in preparation for pruning of log records.
- Prune audit records from the current audit log.

db2icrt - Create Instance

Creates DB2 instances.

On Windows operating systems, the db2icrt utility is located in the `\sqllib\bin` subdirectory.

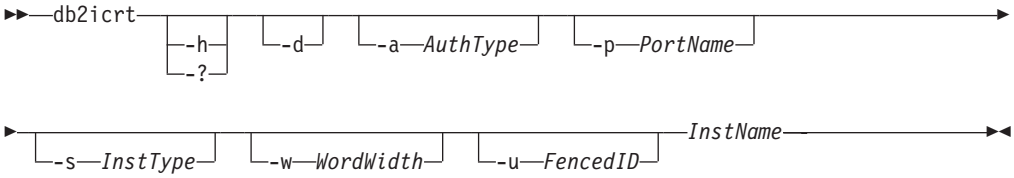
On UNIX-based systems, the db2icrt utility is located in the `DB2DIR/instance` directory, where `DB2DIR` represents `/usr/opt/db2_08_01` on AIX, and `/opt/IBM/db2/V8.1` on all other UNIX-based systems. If you have a FixPak or modification level installed in an alternate path, the `DB2DIR` directory is `usr/opt/db2_08_FPn` on AIX and `opt/IBM/db2/V8.FPn` on all other UNIX-based systems, where `n` represents the number of the FixPak or modification level. The db2icrt utility creates an instance on the directory from which you invoke it.

Authorization:

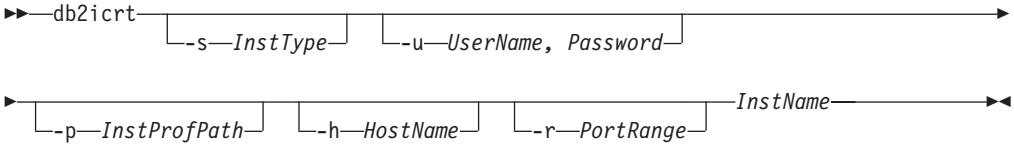
Root access on UNIX-based systems or Local Administrator authority on Windows operating systems.

Command syntax:

For UNIX-based systems



For Windows operating systems



Command parameters:
For UNIX-based systems

-h or -?
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 for your system. Valid values are:

CLIENT
Used to create an instance for a client.

ESE Used to create an instance for a database server with local and remote clients.

Note: Specify this option if you are creating an instance for a PE database system, a single-partition ESE database system, or DB2 Connect.

WSE Used to create an instance for a Workgroup Server Edition server.

-w *WordWidth*
Specifies the width, in bits, of the instance to be created (31, 32 or 64). You must have the requisite version of DB2 installed (31-bit, 32-bit, or 64-bit) to be able to select the appropriate width. The default value is the lowest bit width supported, and depends on the installed version of DB2 UDB, the platform it is operating on, and the instance type. This parameter is only valid on AIX 5L, HP-UX, and the Solaris Operating Environment.

-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 requred if you are creating a server instance.

InstName
Specifies the name of the instance.

For Windows operating systems

-s *InstType*
Specifies the type of instance to create. Valid values are:

Client Used to create an instance for a client.

Note: Use this value if you are using DB2 Connect Personal Edition.

Standalone
Used to create an instance for a database server with local clients.

ESE Used to create an instance for a database server with local and remote clients.

Note: Specify this option if you are creating an instance for a PE database system, a single partition ESE database system, or DB2 Connect.

WSE Used to create an instance for a Workgroup Server Edition server.

-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. 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
```

InstName
Specifies the name of the instance.

Examples:

Example 1:

On an AIX machine, to create an instance called ″db2inst1″ on the directory `/u/db2inst1/sqllib/bin`, issue the following command from that directory:

On a client machine:

```
usr/opt/db2_08_01/instance/db2icrt db2inst1
```

On a server machine:

```
usr/opt/db2_08_01/instance/db2icrt -u db2fenc1 db2inst1
```

where db2fenc1 is the user ID under which fenced user-defined functions and fenced stored procedures will run.

Example 2:

On an AIX machine, if you have Alternate FixPak 1 installed, run the following command to create an instance running FixPak 1 code from the Alternate FixPak 1 install path:

```
/usr/opt/db2_08_FP1/instance/db2icrt -u db2fenc1 db2inst1
```

Usage notes:

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), 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 `db2iclus` command to migrate it to run in a MSCS instance.

Related reference:

- "db2iclus - Microsoft Cluster Server Command" in the *Command Reference*

db2rbind - Rebind all Packages

Rebinds packages in a database.

Authorization:

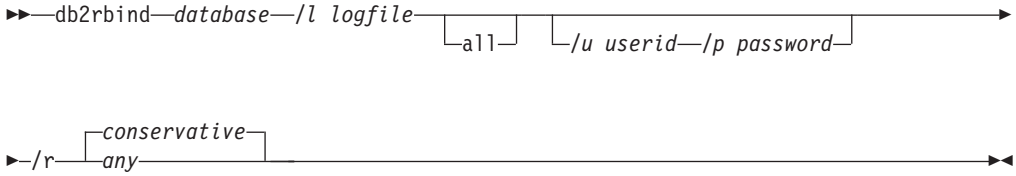
One of the following:

- *sysadm*

Required connection:

None

Command syntax:



Command parameters:

database
Specifies an alias name for the database whose packages are to be revalidated.

/l Specifies the (optional) path and the (mandatory) file name to be used for recording errors that result from the package revalidation procedure.

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 User ID. This parameter must be specified if a password is specified.

/p 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 functions and data types are considered during function resolution and type resolution on static DML statements in the package. This option is not supported by DRDA. Valid values are:

conservative
Only functions and types in the SQL path that were defined before the last explicit bind time stamp are considered for function and type resolution. Conservative binding semantics are used. This is the default. This option is not supported for an inoperative package.

any Any of the functions and types in the SQL path are considered for function and type resolution. Conservative binding semantics are not used.

Usage notes:

- This command uses the rebind API (sqlarbnd) to attempt the revalidation 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.

Related reference:

- "BIND" on page 232
- "PRECOMPILE" on page 842
- "REBIND" on page 866

db2secv82 - Set permissions for DB2 objects

Sets the permissions for DB2 objects (for example, files, directories, network shares, registry keys and services) on updated DB2 Universal Database (UDB) installations.

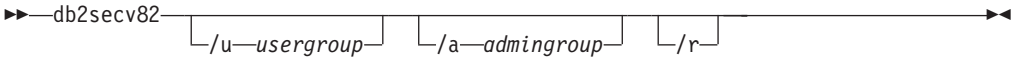
Authorization:

- *sysadm*

Required connection:

none

Command syntax:



Command parameters:

/u *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.

/a *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.

/r Specifies that the changes made by previously running db2secv82.exe should be reversed. If you specify this option, all other options are ignored.

Note: This option will only work if no other DB2 commands have been issued since the db2secv82.exe command was issued.

db2set - DB2 Profile Registry

Displays, sets, or removes DB2 profile variables. An external environment registry command that supports local and remote administration, via the DB2 Administration Server, of DB2's environment variables stored in the DB2 profile registry.

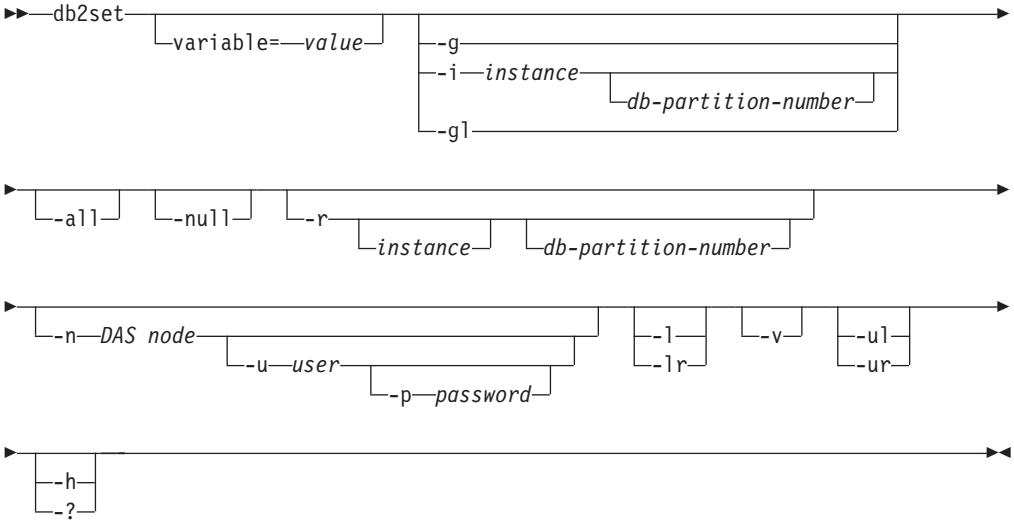
Authorization:

sysadm

Required connection:

None

Command syntax:



Command parameters:

variable= value
Sets a specified variable to a specified value. To delete a variable, do not specify a value for the specified variable. Changes to settings take effect after the instance has been restarted.

-g Accesses the global profile variables.

-i Specifies the instance profile to use instead of the current, or default.

db-partition-number
Specifies a number listed in the `db2nodes.cfg` file.

-gl Accesses the global profile variables stored in LDAP. This option is only effective if the registry variable DB2_ENABLE_LDAP has been set to YES.

-all Displays all occurrences of the local environment variables as defined in:

- The environment, denoted by [e]
- The node level registry, denoted by [n]
- The instance level registry, denoted by [i]
- The global level registry, denoted by [g].

-null Sets the value of the variable at the specified registry level to NULL. This avoids having to look up the value in the next registry level, as defined by the search order.

-r instance
Resets the profile registry for the given instance. If no instance is specified, and an instance attachment exists, resets the profile for the current instance. If no instance is specified, and no attachment exists, resets the profile for the instance specified by the DB2INSTANCE environment variable.

-n DAS node
Specifies the remote DB2 administration server node name.

-u user
Specifies the user ID to use for the administration server attachment.

-p password
Specifies the password to use for the administration server attachment.

-l Lists all instance profiles.

-lr Lists all supported registry variables.

-v Specifies verbose mode.

-ul Accesses the user profile variables.

Note: This parameter is supported on Windows operating systems only.

-ur Refreshes the user profile variables.

Note: This parameter is supported on Windows operating systems only.

-h/-? Displays help information. When this option is specified, all other options are ignored, and only the help information is displayed.

Examples:

- Display all defined profiles (DB2 instances):

```
db2set -l
```

- Display all supported registry variables:

```
db2set -lr
```

- Display all defined global variables:
  ```
  db2set -g
  ```
- Display all defined variables for the current instance:
  ```
  db2set
  ```
- Display all defined values for the current instance:
  ```
  db2set -all
  ```
- Display all defined values for DB2COMM for the current instance:
  ```
  db2set -all DB2COMM
  ```
- Reset all defined variables for the instance INST on node 3:
  ```
  db2set -r -i INST 3
  ```
- Unset the variable DB2CHKPTR on the remote instance RMTINST through the DAS node RMTDAS using user ID MYID and password MYPASSWD:
  ```
  db2set -i RMTINST -n RMTDAS -u MYID -p MYPASSWD DB2CHKPTR=
  ```
- Set the variable DB2COMM to be TCPIP,IPXSPX,NETBIOS globally:
  ```
  db2set -g DB2COMM=TCPIP,IPXSPX,NETBIOS
  ```
- Set the variable DB2COMM to be only TCPIP for instance MYINST:
  ```
  db2set -i MYINST DB2COMM=TCPIP
  ```
- Set the variable DB2COMM to null at the given instance level:
  ```
  db2set -null DB2COMM
  ```

Usage notes:

If no variable name is specified, the values of all defined variables are displayed. If a variable name *is* specified, only the value of that variable is displayed. To display all the defined values of a variable, specify *variable* `-all`. To display all the defined variables in all registries, specify `-all`.

To modify the value of a variable, specify *variable*`=`, followed by its new value. To set the value of a variable to NULL, specify *variable* `-null`.

Note: Changes to settings take effect after the instance has been restarted.

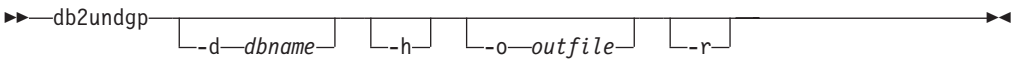
To delete a variable, specify *variable*`=`, followed by no value.

db2undgp - Revoke Execute Privilege

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 procedure 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*
database name (maximum of 8 characters)

-h Displays help for the command.

-o *outfile*
output the revoke statements in the specified file File name length <= 80

-r perform the revoke

Usage notes:

Notes:

1. 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 node 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 with DB2 ESE, 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 back/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.

This command unlinks all files that are linked through any DATALINK columns. Since the unlink operation is performed asynchronously on the DB2 Data Links Manager, its effects might not be seen immediately on the DB2 Data Links Manager, and the unlinked files might not be immediately available for other operations. When the command is issued, all the DB2 Data Links Managers configured to that database must be available; otherwise, the drop database operation will fail.

Compatibilities:

For compatibility with versions earlier than Version 8:

- The keyword NODE can be substituted for DBPARTITIONNUM.

Related reference:

- "CATALOG DATABASE" on page 249
- "CREATE DATABASE" on page 252
- "UNCATALOG DATABASE Command" in the *Command Reference*

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.

Authorization:

One of the following:

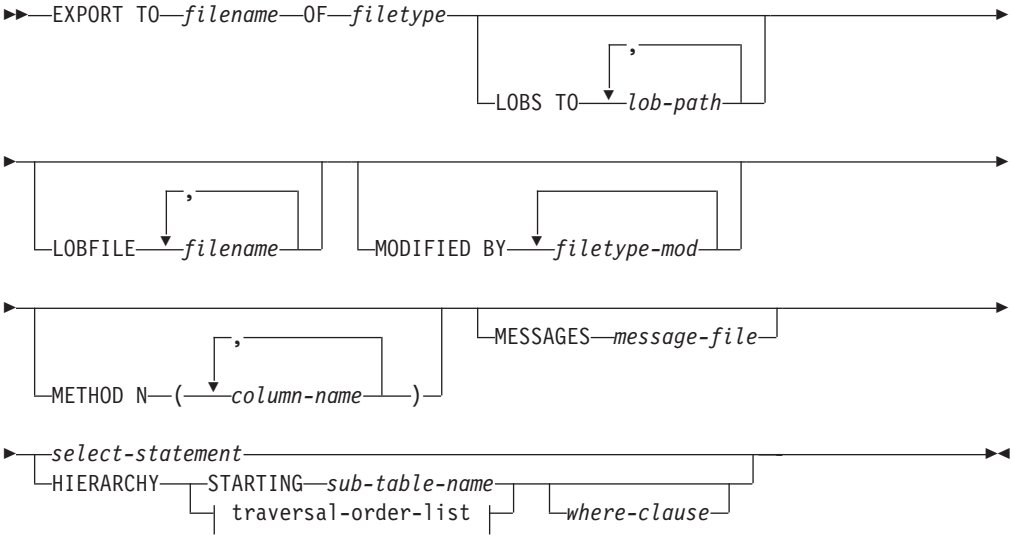
- *sysadm*
- *dbadm*

or CONTROL or SELECT privilege on each participating table or view.

Required connection:

Database. If implicit connect is enabled, a connection to the default database is established.

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.

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. 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`, `/u/foo/lob/path/bar.002`, 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.

MESSAGES message-file
Specifies the destination for warning and error messages that occur during an export operation. If the file already exists, the export utility appends the information. If *message-file* is omitted, the messages are written to standard output.

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 export.

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

  Note: 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.
- IXF (integrated exchange format, PC version), in which most of the table attributes, as well as any existing indexes, are saved in the IXF file, except when columns are specified in the SELECT statement. With this format, the table can be recreated, while with the other file formats, the table must already exist before data can be imported into it.

select-statement
Specifies the SELECT statement that will return the data to be exported. If the SELECT 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
Specifies the name of the file to which data is to be exported. If the complete path to the file is not specified, the export utility uses the current directory and the default drive as the destination.

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.

Examples:

The following example shows how to export information from the STAFF table in the SAMPLE database to the file `myfile.ixf`. The output will be in IXF format.

Note that you must be connected to the SAMPLE database before issuing the command. The index definitions (if any) will be stored in the output file except when the database connection is made through DB2 Connect.

```
db2 export to myfile.ixf of ixf messages msgs.txt select * from staff
```

The following example shows how to export the information about employees in Department 20 from the STAFF table in the SAMPLE database. The output will be in IXF format and will go into the awards.ixf file. Note that you must first connect to the SAMPLE database before issuing the command. Also note that the actual column name in the table is 'dept' instead of 'department'.

```
db2 export to awards.ixf of ixf messages msgs.txt select * from staff
   where dept = 20
```

The following example shows how to export LOBs to a DEL file:

```
db2 export to myfile.del of del lobs to mylobs/
   lobfile lobs1, lobs2 modified by lobsinfile
   select * from emp_photo
```

The following example shows how to export LOBs to a DEL file, specifying a second directory for files that might not fit into the first directory:

```
db2 export to myfile.del of del
   lobs to /db2exp1/, /db2exp2/ modified by lobsinfile
   select * from emp_photo
```

The following example shows how to export data to a DEL file, using a single quotation mark as the string delimiter, a semicolon as the column delimiter, and a comma as the decimal point. The same convention should be used when importing data back into the database:

```
db2 export to myfile.del of del
   modified by chardel'' coldel; decpt,
   select * from staff
```

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.

The export utility will not create multiple-part PC/IXF files when invoked from an AIX system.

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.

DB2 Data Links Manager considerations:

To ensure that a consistent copy of the table and the corresponding files referenced by the DATALINK columns are copied for export, do the following:

1. Issue the command: QUIESCE TABLESPACES FOR TABLE tablename SHARE.
   This ensures that no update transactions are in progress when EXPORT is run.
2. Issue the EXPORT command.
3. Run the **dlfm_export** utility at each Data Links server. Input to the **dlfm_export** utility is the control file name, which is generated by the export utility. This produces a tar (or equivalent) archive of the files listed within the control file.
4. Issue the command: QUIESCE TABLESPACES FOR TABLE tablename RESET.
   This makes the table available for updates.

EXPORT is executed as an SQL application. The rows and columns satisfying the SELECT statement conditions are extracted from the database. For the DATALINK columns, the SELECT statement should not specify any scalar function.

Successful execution of EXPORT results in generation of the following files:

- An export data file as specified in the EXPORT command. A DATALINK column value in this file has the same format as that used by the IMPORT and LOAD utilities. When the DATALINK column value is the SQL NULL value, handling is the same as that for other data types.
- Control files *server_name*, which are generated for each Data Links server. On Windows operating systems, a single control file, `ctrlfile.lst`, is used by all Data Links servers. These control files are placed in the directory <data-file path>/dlfm/YYYYMMDD/HHMMSS (on the Windows NT operating system, `ctrlfile.lst` is placed in the directory <data-file path>\dlfm\YYYYMMDD\HHMMSS). YYYYMMDD represents the date (year month day), and HHMMSS represents the time (hour minute second).

The **dlfm_export** utility is provided to export files from a Data Links server. This utility generates an archive file, which can be used to restore files in the target Data Links server.

Related concepts:

- "Export Overview" in the *Data Movement Utilities Guide and Reference*
- "Privileges, authorities and authorization required to use export" on page 837

Related tasks:

- "Using Export" in the *Data Movement Utilities Guide and Reference*

Related reference:

- "db2Export - Export" on page 405
- "Export Sessions - CLP Examples" in the *Data Movement Utilities Guide and Reference*
- "File type modifiers for export" in the *Command Reference*
- "Delimiter restrictions for moving data" in the *Command Reference*

GET AUTHORIZATIONS

Reports the authorities of the current user from values found in the database configuration file and the authorization system catalog view (`SYSCAT.DBAUTH`).

Authorization:

None

Required connection:

Database. If implicit connect is enabled, a connection to the default database is established.

Command syntax:

Command parameters:
None
Examples:
The following is sample output from GET AUTHORIZATIONS:

```
 Administrative Authorizations for Current User

 Direct SYSADM authority                     = NO
 Direct SYSCTRL authority                    = NO
 Direct SYSMAINT authority                   = NO
 Direct DBADM authority                      = YES
 Direct CREATETAB authority                  = YES
 Direct BINDADD authority                    = YES
 Direct CONNECT authority                    = YES
 Direct CREATE_NOT_FENC authority            = YES
 Direct IMPLICIT_SCHEMA authority            = YES
 Direct LOAD authority                       = YES
 Direct QUIESCE_CONNECT authority            = YES
 Direct CREATE_EXTERNAL_ROUTINE authority    = YES

 Indirect SYSADM authority                   = YES
 Indirect SYSCTRL authority                  = NO
 Indirect SYSMAINT authority                 = NO
 Indirect DBADM authority                    = NO
 Indirect CREATETAB authority                = YES
 Indirect BINDADD authority                  = YES
 Indirect CONNECT authority                  = YES
 Indirect CREATE_NOT_FENC authority          = NO
 Indirect IMPLICIT_SCHEMA authority          = YES
 Indirect LOAD authority                     = NO
 Indirect QUIESCE_CONNECT authority          = NO
 Indirect CREATE_EXTERNAL_ROUTINE authority = NO
```

Usage notes:

Direct authorities are acquired by explicit commands that grant the authorities to a user ID. Indirect authorities are based on authorities acquired by the groups to which a user belongs.

Note: PUBLIC is a special group to which all users belong.

GET DATABASE CONFIGURATION

Returns the values of individual entries in a specific database configuration file.

Scope:

This command returns information only for the partition on which it is executed.

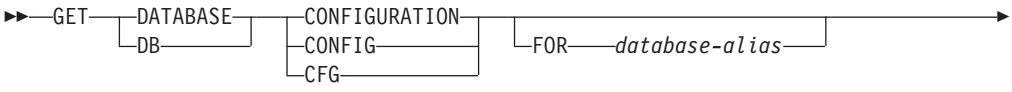
Authorization:

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.

Examples:

Notes:

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 AIX):

```
                    Database Configuration for Database mick

 Database configuration release level                    = 0x0a00
 Database release level                                  = 0x0a00

 Database territory                                      = en_US
 Database code page                                      = 819
 Database code set                                       = ISO8859-1
 Database country/region code                            = 1
 Database collating sequence                             = UNIQUE
 Alternate collating sequence              (ALT_COLLATE) =

 Dynamic SQL Query management           (DYN_QUERY_MGMT) = DISABLE

 Discovery support for this database       (DISCOVER_DB) = ENABLE

 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

 Backup pending                                          = NO

 Database is consistent                                  = YES
 Rollforward pending                                     = NO
 Restore pending                                         = NO

 Multi-page file allocation enabled                      = YES

 Log retain for recovery status                          = NO
 User exit for logging status                            = NO
```

```
Data Links Token Expiry Interval (sec)      (DL_EXPINT) = 60
Data Links Write Token Init Expiry Intvl(DL_WT_IEXPINT) = 60
Data Links Number of Copies             (DL_NUM_COPIES) = 1
Data Links Time after Drop (days)        (DL_TIME_DROP) = 1
Data Links Token in Uppercase                (DL_UPPER) = NO
Data Links Token Algorithm                   (DL_TOKEN) = MAC0

Database heap (4KB)                            (DBHEAP) = 1200
Size of database shared memory (4KB)  (DATABASE_MEMORY) = AUTOMATIC
Catalog cache size (4KB)              (CATALOGCACHE_SZ) = 64
Log buffer size (4KB)                        (LOGBUFSZ) = 8
Utilities heap size (4KB)                (UTIL_HEAP_SZ) = 5000
Buffer pool size (pages)                     (BUFFPAGE) = 1000
Extended storage segments size (4KB)    (ESTORE_SEG_SZ) = 16000
Number of extended storage segments   (NUM_ESTORE_SEGS) = 0
Max storage for lock list (4KB)              (LOCKLIST) = 128

Max size of appl. group mem set (4KB) (APPGROUP_MEM_SZ) = 30000
Percent of mem for appl. group heap   (GROUPHEAP_RATIO) = 70
Max appl. control heap size (4KB)     (APP_CTL_HEAP_SZ) = 128

Sort heap thres for shared sorts (4KB) (SHEAPTHRES_SHR) = (SHEAPTHRES)
Sort list heap (4KB)                         (SORTHEAP) = 256
SQL statement heap (4KB)                     (STMTHEAP) = 2048
Default application heap (4KB)             (APPLHEAPSZ) = 128
Package cache size (4KB)                   (PCKCACHESZ) = (MAXAPPLS*8)
Statistics heap size (4KB)                (STAT_HEAP_SZ) = 4384

Interval for checking deadlock (ms)          (DLCHKTIME) = 10000
Percent. of lock lists per application        (MAXLOCKS) = 10
Lock timeout (sec)                          (LOCKTIMEOUT) = -1

Changed pages threshold                 (CHNGPGS_THRESH) = 60
Number of asynchronous page cleaners    (NUM_IOCLEANERS) = 1
Number of I/O servers                    (NUM_IOSERVERS) = 3
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) = 1
Max DB files open per application             (MAXFILOP) = 64

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                                        = /home/db2inst/db2inst
                                                           /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 of max active 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

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

TSM management class                    (TSM_MGMTCLASS) =
TSM node name                            (TSM_NODENAME) =
TSM owner                                   (TSM_OWNER) =
TSM password                             (TSM_PASSWORD) =

Automatic maintenance                       (AUTO_MAINT) = OFF
  Automatic database backup            (AUTO_DB_BACKUP) = OFF
  Automatic table maintenance          (AUTO_TBL_MAINT) = OFF
    Automatic runstats                  (AUTO_RUNSTATS) = OFF
    Automatic statistics profiling    (AUTO_STATS_PROF) = OFF
      Automatic profile updates         (AUTO_PROF_UPD) = OFF
    Automatic reorganization               (AUTO_REORG) = OFF
```

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 mick
Description                                Parameter    Current Value  Delayed
                                                                       Value

Database configuration release level                    = 0x0a00
Database release level                                  = 0x0a00

Database territory                                      = en_US
Database code page                                      = 819
Database code set                                       = ISO8859-1
Database country/region code                            = 1
Database collating sequence                             = UNIQUE      UNIQUE
Alternate collating sequence                (ALT_COLLATE) =
Dynamic SQL Query management             (DYN_QUERY_MGMT) = DISABLE     DISABLE
Discovery support for this database         (DISCOVER_DB) = ENABLE      ENABLE
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

Backup pending                                          = NO
```

```
Database is consistent                                = YES
Rollforward pending                                   = NO
Restore pending                                       = NO

Multi-page file allocation enabled                    = YES

Log retain for recovery status                        = NO
User exit for logging status                          = NO

Data Links Token Expiry Interval (sec)      (DL_EXPINT) = 60          60
Data Links Write Token Init Expiry Intvl(DL_WT_IEXPINT) = 60          60
Data Links Number of Copies             (DL_NUM_COPIES) = 1           1
Data Links Time after Drop (days)        (DL_TIME_DROP) = 1           1
Data Links Token in Uppercase                (DL_UPPER) = NO          NO
Data Links Token Algorithm                   (DL_TOKEN) = MAC0        MAC0

Database heap (4KB)                           (DBHEAP) = 1200        1200
Size of database shared memory (4KB)  (DATABASE_MEMORY) = AUTOMATIC   AUTOMATIC
                                                          (11516)      (11516)
Catalog cache size (4KB)              (CATALOGCACHE_SZ) = 64          64
Log buffer size (4KB)                        (LOGBUFSZ) = 8           8
Utilities heap size (4KB)                (UTIL_HEAP_SZ) = 5000        5000
Buffer pool size (pages)                     (BUFFPAGE) = 1000        1000
Extended storage segments size (4KB)    (ESTORE_SEG_SZ) = 16000       16000
Number of extended storage segments   (NUM_ESTORE_SEGS) = 0           0
Max storage for lock list (4KB)              (LOCKLIST) = 128         128

Max size of appl. group mem set (4KB) (APPGROUP_MEM_SZ) = 30000       30000
Percent of mem for appl. group heap   (GROUPHEAP_RATIO) = 70          70
Max appl. control heap size (4KB)     (APP_CTL_HEAP_SZ) = 128         128
Sort heap thres for shared sorts (4KB) (SHEAPTHRES_SHR) = (SHEAPTHRES) (SHEAPTHRES)
Sort list heap (4KB)                         (SORTHEAP) = 256         256
SQL statement heap (4KB)                     (STMTHEAP) = 2048        2048
Default application heap (4KB)             (APPLHEAPSZ) = 128         128
Package cache size (4KB)                   (PCKCACHESZ) = (MAXAPPLS*8) (MAXAPPLS*8)
Statistics heap size (4KB)                (STAT_HEAP_SZ) = 4384        4384

Interval for checking deadlock (ms)          (DLCHKTIME) = 10000       10000
Percent. of lock lists per application        (MAXLOCKS) = 10          10
Lock timeout (sec)                         (LOCKTIMEOUT) = -1          -1

Changed pages threshold                  (CHNGPGS_THRESH) = 60          60
Number of asynchronous page cleaners    (NUM_IOCLEANERS) = 1           1
Number of I/O servers                    (NUM_IOSERVERS) = 3           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   AUTOMATIC
                                                            (40)        (40)
Average number of active applications        (AVG_APPLS) = 1           1
Max DB files open per application             (MAXFILOP) = 64          64

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                                         = home/db2inst /home
                                                            /db2inst      /db2inst
                                                            /NODE0000     /db2inst
                                                            /SQL00001     /NODE0000
```

```
                                                        /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 of max active 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

 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)  (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

 TSM management class                    (TSM_MGMTCLASS) =
 TSM node name                            (TSM_NODENAME) =
 TSM owner                                   (TSM_OWNER) =
 TSM password                             (TSM_PASSWORD) =

 Automatic maintenance                       (AUTO_MAINT) = OFF         OFF
   Automatic database backup              (AUTO_DB_BACKUP) = OFF         OFF
   Automatic table maintenance            (AUTO_TBL_MAINT) = OFF         OFF
     Automatic runstats                    (AUTO_RUNSTATS) = OFF         OFF
     Automatic statistics profiling       (AUTO_STATS_PROF) = OFF         OFF
     Automatic profile updates             (AUTO_PROF_UPD) = OFF         OFF
     Automatic reorganization                 (AUTO_REORG) = OFF          OFF
```

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.

Related tasks:

- "Changing node and database configuration files" in the *Administration Guide: Implementation*
- "Configuring DB2 with configuration parameters" on page 779

Related reference:

- "RESET DATABASE CONFIGURATION Command" in the *Command Reference*
- "UPDATE DATABASE CONFIGURATION" on page 381
- "Configuration parameters summary" in the *Administration Guide: Performance*

GET DATABASE MANAGER CONFIGURATION

Returns the values of individual entries in the database manager configuration file.

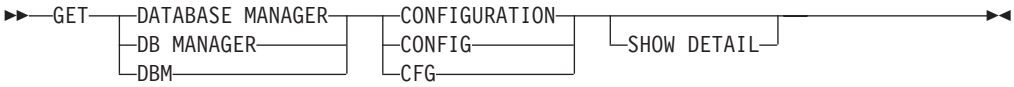
Authorization:

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.

Examples:

Note: Both node type and platform determine which configuration parameters are listed.

The following is sample output from GET DATABASE MANAGER CONFIGURATION (issued on AIX):

```
          Database Manager Configuration

     Node type = Database Server with local clients

 Database manager configuration release level            = 0x0a00

 CPU speed (millisec/instruction)             (CPUSPEED) = 4.000000e-05

 Max number of concurrently active databases     (NUMDB) = 8
 Data Links support                          (DATALINKS) = NO
 Federated Database System Support           (FEDERATED) = NO
 Transaction processor monitor name        (TP_MON_NAME) =

 Default charge-back account           (DFT_ACCOUNT_STR) =
```

GET DATABASE MANAGER CONFIGURATION

```
Java Development Kit installation path       (JDK_PATH) = /usr/java131

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) =
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
Database manager authentication         (AUTHENTICATION) = SERVER
Cataloging allowed without authority    (CATALOG_NOAUTH) = YES
Trust all clients                       (TRUST_ALLCLNTS) = YES
Trusted client authentication           (TRUST_CLNTAUTH) = CLIENT
Bypass federated authentication             (FED_NOAUTH) = NO

Default database path                        (DFTDBPATH) = /home/db2inst

Database monitor heap size (4KB)           (MON_HEAP_SZ) = 90
Java Virtual Machine heap size (4KB)      (JAVA_HEAP_SZ) = 512
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

Sort heap threshold (4KB)                   (SHEAPTHRES) = 20000

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
Max number of existing agents                (MAXAGENTS) = 200
Agent pool size                          (NUM_POOLAGENTS) = 100(calculated)
Initial number of agents in pool         (NUM_INITAGENTS) = 0
Max number of coordinating agents       (MAX_COORDAGENTS) = MAXAGENTS
Max no. of concurrent coordinating agents  (MAXCAGENTS) = MAX_COORDAGENTS
Max number of client connections        (MAX_CONNECTIONS) = MAX_COORDAGENTS

Keep fenced process                          (KEEPFENCED) = YES
Number of pooled fenced processes           (FENCED_POOL) = MAX_COORDAGENTS
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) =
 SPM log size                            (SPM_LOG_FILE_SZ) = 256
 SPM resync agent limit                   (SPM_MAX_RESYNC) = 20
 SPM log path                               (SPM_LOG_PATH) =

 TCP/IP Service name                          (SVCENAME) =
 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

 No. of int. communication buffers(4KB)(FCM_NUM_BUFFERS) = 512
 Number of FCM request blocks              (FCM_NUM_RQB) = AUTOMATIC
 Number of FCM connection entries      (FCM_NUM_CONNECT) = AUTOMATIC
 Number of FCM message anchors         (FCM_NUM_ANCHORS) = AUTOMATIC
```

The following output sample shows the information displayed when you specify the WITH DETAIL option. The value that appears in the **Delayed Value** is the value that will be in effect the next time you start the database manager instance.

```
                        Database Manager Configuration
 Node type = Database Server with local clients
 Description                                   Parameter   Current Value  Delayed
                                                                           Value

Database manager configuration release level            = 0x0a00

 CPU speed (millisec/instruction)             (CPUSPEED) = 4.000000e     4.000000e
                                                           -05           -05
 Max number of concurrently active databases     (NUMDB) = 8             8
 Data Links support                          (DATALINKS) = NO            NO
 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) = /wsdb/v81     /usr
                                                            /bldsupp      /java131
                                                            /AIX/jdk1.3.1

 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) = BUILD
 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) =
```

```
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_           NOT_
                                                          SPECIFIED    SPECIFIED
Database manager authentication        (AUTHENTICATION) = SERVER      SERVER
Cataloging allowed without authority   (CATALOG_NOAUTH) = YES         YES
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) = /home       /home
                                                           /db2inst    /db2inst

Database monitor heap size (4KB)          (MON_HEAP_SZ) = 90          90
Java Virtual Machine heap size (4KB)     (JAVA_HEAP_SZ) = 512         512
Audit buffer size (4KB)                  (AUDIT_BUF_SZ) = 0           0
Size of instance shared memory (4KB)  (INSTANCE_MEMORY) = AUTOMATIC   AUTOMATIC
                                                          (5386)       (20)
Backup buffer default size (4KB)            (BACKBUFSZ) = 1024        1024
Restore buffer default size (4KB)           (RESTBUFSZ) = 1024        1024

Sort heap threshold (4KB)                  (SHEAPTHRES) = 20000       20000

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
Max number of existing agents               (MAXAGENTS) = 200         200
Agent pool size                        (NUM_POOLAGENTS) = 100         100
                                                                      (calculated)
Initial number of agents in pool       (NUM_INITAGENTS) = 0           0
Max number of coordinating agents     (MAX_COORDAGENTS) = 200         MAXAGENTS
Max no. of concurrent coordinating agents  (MAXCAGENTS) = 200         MAX_COORDAGENTS
Max number of client connections      (MAX_CONNECTIONS) = 200         MAX_COORDAGENTS

Keep fenced process                         (KEEPFENCED) = YES         YES
Number of pooled fenced processes           (FENCED_POOL) = MAX_         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) =
SPM log size                             (SPM_LOG_FILE_SZ) = 256         256
SPM resync agent limit                    (SPM_MAX_RESYNC) = 20          20
SPM log path                                (SPM_LOG_PATH) =

TCP/IP Service name                           (SVCENAME) =
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

 No. of int. communication buffers(4KB)(FCM_NUM_BUFFERS) = 0           512
 Number of FCM request blocks              (FCM_NUM_RQB) = AUTOMATIC   AUTOMATIC
                                                            (0)         (256)
 Number of FCM connection entries      (FCM_NUM_CONNECT) = AUTOMATIC   AUTOMATIC
                                                            (-1)        (-1)
 Number of FCM message anchors         (FCM_NUM_ANCHORS) = AUTOMATIC   AUTOMATIC
                                                            (-1)          (-1)
```

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 install the database manager again to recover.

To set the configuration parameters to the default values shipped with the database manager, use the RESET DATABASE MANAGER CONFIGURATION command.

Related tasks:

- "Changing node and database configuration files" in the *Administration Guide: Implementation*
- "Configuring DB2 with configuration parameters" on page 779

Related reference:

- "RESET DATABASE MANAGER CONFIGURATION Command" in the *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION" on page 384
- "Configuration parameters summary" in the *Administration Guide: Performance*

IMPORT

Inserts data from an external file with a supported file format into a table, hierarchy, or view. LOAD is a faster alternative, but the load utility does not support loading data at the hierarchy level.

Authorization:

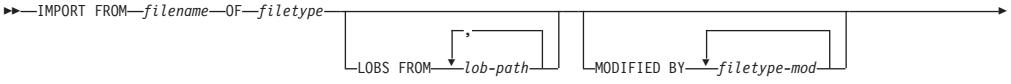
- IMPORT using the INSERT option requires one of the following:
  - *sysadm*
  - *dbadm*
  - CONTROL privilege on each participating table or view
  - INSERT and SELECT privilege on each participating table or view
- IMPORT to an existing table using the INSERT_UPDATE option, requires one of the following:
  - *sysadm*
  - *dbadm*
  - CONTROL privilege on the table or view

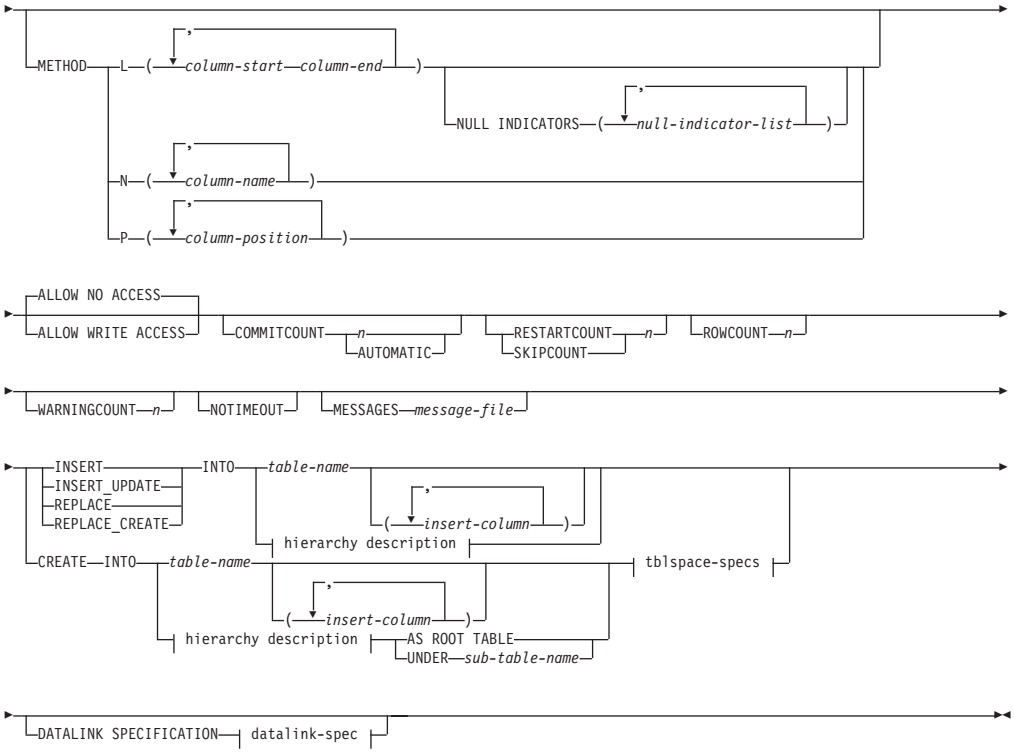
  - INSERT, SELECT, UPDATE and DELETE privilege on each participating table or view
- IMPORT to an existing table using the REPLACE or REPLACE_CREATE option, requires one of the following:
  - *sysadm*
  - *dbadm*
  - CONTROL privilege on the table or view
  - INSERT, SELECT, and DELETE privilege on the table or view
- IMPORT to a new table using the CREATE or REPLACE_CREATE option, requires one of the following:
  - *sysadm*
  - *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
    - CREATIN 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:
  - *sysadm*
  - *dbadm*
  - 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:
  - *sysadm*
  - *dbadm*
  - 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.

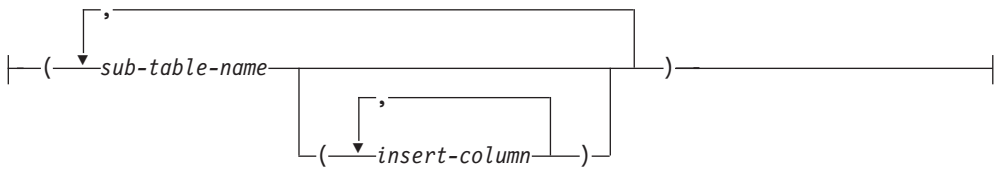
Command syntax:





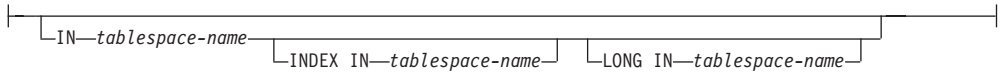
hierarchy description:

sub-table-list:

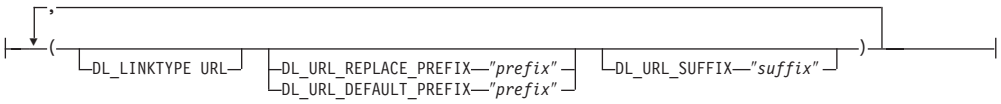


traversal-order-list:

tblspace-specs:



datalink-spec:



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.

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.

CREATE
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.

Note: If the data was exported from an MVS host database, and it contains LONGVAR fields whose lengths, calculated on the page size, are less than 254, CREATE might fail because the rows are too long. See Using import to recreate an exported table 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.

DATALINK SPECIFICATION

For each DATALINK column, there can be one column specification enclosed by parentheses. Each column specification consists of one or more DL_LINKTYPE, prefix, and a DL_URL_SUFFIX specification. The prefix specification can be either DL_URL_REPLACE_PREFIX or DL_URL_DEFAULT_PREFIX.

There can be as many DATALINK column specifications as the number of DATALINK columns defined in the table. The order of specifications follows the order of DATALINK columns found within the *insert-column* list, or within the table definition (if an *insert-column* list is not specified).

DL_LINKTYPE

If specified, it should match the LINKTYPE of the column definition. Thus, DL_LINKTYPE URL is acceptable if the column definition specifies LINKTYPE URL.

DL_URL_DEFAULT_PREFIX ″**prefix**″

If specified, it should act as the default prefix for all DATALINK values within the same column. In this context, prefix refers to the ″scheme host port″ part of the URL specification.

Examples of prefix are:

```
"http://server"
"file://server"
"file:"
"http://server:80"
```

If no prefix is found in a column's data, and a default prefix is specified with DL_URL_DEFAULT_PREFIX, the default prefix is prefixed to the column value (if not NULL).

For example, if DL_URL_DEFAULT_PREFIX specifies the default prefix `"http://toronto"`:

- The column input value ″/x/y/z″ is stored as ″http://toronto/x/y/z″.
- The column input value ″http://coyote/a/b/c″ is stored as ″http://coyote/a/b/c″.
- The column input value NULL is stored as NULL.

DL_URL_REPLACE_PREFIX ″**prefix**″

This clause is useful for loading or importing data previously generated by the export utility, when the user wants to globally replace the host name in the data with another host name. If specified, it becomes the prefix for *all* non-NULL column values. If a column value has a prefix, this will replace it. If a column value has no prefix, the prefix specified by DL_URL_REPLACE_PREFIX is prefixed to the column value.

For example, if DL_URL_REPLACE_PREFIX specifies the prefix `"http://toronto"`:

- The column input value ″/x/y/z″ is stored as ″http://toronto/x/y/z″.
- The column input value ″http://coyote/a/b/c″ is stored as ″http://toronto/a/b/c″. Note that ″toronto″ replaces ″coyote″.
- The column input value NULL is stored as NULL.

DL_URL_SUFFIX "suffix"
If specified, it is appended to every non-NULL column value for the column. It is, in fact, appended to the "path" component of the URL part of the DATALINK value.

FROM filename
Specifies the file that contains the data to be imported. If the path is omitted, the current working directory is used.

HIERARCHY
Specifies that hierarchical data is to be imported.

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 declared temporary table or a summary table.

One can use an alias for INSERT, INSERT_UPDATE, or REPLACE, except in the case of a down-level 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
Specifies one or more paths that store LOB files. 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. This option is ignored if the `lobsinfile` modifier is not specified.

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.

MESSAGES message-file
Specifies the destination for warning and error messages that occur during an import operation. If the file already exists, the import utility appends the information. If the complete path to the file is not specified, the utility uses the current directory and the default drive as the destination. If *message-file* is omitted, the messages are written to standard output.

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 to be imported.

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 import.

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 (integrated exchange format, PC version), which means it was exported from the same or another DB2 table. An IXF file also contains the table definition and definitions of any existing indexes, except when columns are specified in the SELECT statement.

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. It is not valid for tables with DATALINK columns. If this option is used when moving data between hierarchies, only the data for an entire hierarchy, not individual subtables, can be replaced.

REPLACE_CREATE
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 Using import to recreate an exported table for a list of restrictions.

This option can only be used with IXF files. It is not valid for tables with DATALINK columns. If this option is used when moving data between hierarchies, only the data for an entire hierarchy, not individual subtables, can be replaced.

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.

Examples:

Example 1

The following example shows how to import information from `myfile.ixf` to the STAFF table:

```
   db2 import from myfile.ixf of ixf messages msg.txt insert into staff

SQL3150N  The H record in the PC/IXF file has product "DB2    01.00", date
"19970220", and time "140848".

SQL3153N  The T record in the PC/IXF file has name "myfile",
qualifier "        ", and source "            ".

SQL3109N  The utility is beginning to load data from file "myfile".

SQL3110N  The utility has completed processing.  "58" rows were read
from the input file.

SQL3221W  ...Begin COMMIT WORK. Input Record Count = "58".

SQL3222W  ...COMMIT of any database changes was successful.

SQL3149N  "58" rows were processed from the input file.  "58" rows were
successfully inserted into the table.  "0" rows were rejected.
```

Example 2

The following example shows how to import the table MOVIETABLE from the input file `delfile1`, which has data in the DEL format:

```
   db2 import from delfile1 of del
       modified by dldel|
       insert into movietable (actorname, description, url_making_of,
       url_movie) datalink specification (dl_url_default_prefix
       "http://narang"), (dl_url_replace_prefix "http://bomdel"
       dl_url_suffix ".mpeg")
```

Notes:

1. The table has four columns:

```
      actorname             VARCHAR(n)
      description           VARCHAR(m)
      url_making_of         DATALINK (with LINKTYPE URL)
      url_movie             DATALINK (with LINKTYPE URL)
```

2. The DATALINK data in the input file has the vertical bar (|) character as the sub-field delimiter.
3. If any column value for url_making_of does not have the prefix character sequence, "http://narang" is used.
4. Each non-NULL column value for url_movie will get "http://bomdel" as its prefix. Existing values are replaced.
5. Each non-NULL column value for url_movie will get ".mpeg" appended to the path. For example, if a column value of url_movie is "http://server1/x/y/z", it will be stored as "http://bomdel/x/y/z.mpeg"; if the value is "/x/y/z", it will be stored as "http://bomdel/x/y/z.mpeg".

Example 3 (Importing into a Table with an Identity Column)

TABLE1 has 4 columns:

- C1 VARCHAR(30)
- C2 INT GENERATED BY DEFAULT AS IDENTITY
- C3 DECIMAL(7,2)
- C4 CHAR(1)

TABLE2 is the same as TABLE1, except that C2 is a GENERATED ALWAYS identity column.

Data records in DATAFILE1 (DEL format):

```
"Liszt"
"Hummel",,187.43, H
"Grieg",100, 66.34, G
"Satie",101, 818.23, I
```

Data records in DATAFILE2 (DEL format):

```
"Liszt", 74.49, A
"Hummel", 0.01, H
"Grieg", 66.34, G
"Satie", 818.23, I
```

The following command generates identity values for rows 1 and 2, since no identity values are supplied in DATAFILE1 for those rows. Rows 3 and 4, however, are assigned the user-supplied identity values of 100 and 101, respectively.

```
db2 import from datafile1.del of del replace into table1
```

To import DATAFILE1 into TABLE1 so that identity values are generated for all rows, issue one of the following commands:

```
db2 import from datafile1.del of del method P(1, 3, 4)
   replace into table1 (c1, c3, c4)
db2 import from datafile1.del of del modified by identityignore
   replace into table1
```

To import DATAFILE2 into TABLE1 so that identity values are generated for each row, issue one of the following commands:

```
db2 import from datafile2.del of del replace into table1 (c1, c3, c4)
db2 import from datafile2.del of del modified by identitymissing
   replace into table1
```

If DATAFILE1 is imported into TABLE2 without using any of the identity-related file type modifiers, rows 1 and 2 will be inserted, but rows 3 and 4 will be rejected, because they supply their own non-NULL values, and the identity column is GENERATED ALWAYS.

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.

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 60KB), the message file returned to the user on the client might be missing messages from the middle of the import operation. The first 30KB of message information and the last 30KB 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 UDB 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 declared 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.

DB2 Data Links Manager considerations:

Before running the DB2 import utility, do the following:

1. Copy the files that will be referenced to the appropriate Data Links servers. The `dlfm_import` utility can be used to extract files from an archive that is generated by the `dlfm_export` utility.
2. Register the required prefix names to the DB2 Data Links Managers. There might be other administrative tasks, such as registering the database, if required.
3. Update the Data Links server information in the URLs (of the DATALINK columns) from the exported data for the SQL table, if required. (If the original configuration's Data Links servers are the same at the target location, the Data Links server names need not be updated.)
4. Define the Data Links servers at the target configuration in the DB2 Data Links Manager configuration file.

When the import utility runs against the target database, files referred to by DATALINK column data are linked on the appropriate Data Links servers.

During the insert operation, DATALINK column processing links the files in the appropriate Data Links servers according to the column specifications at the target database.

Related concepts:

- "Import Overview" in the *Data Movement Utilities Guide and Reference*
- "Privileges, authorities, and authorization required to use import" in the *Data Movement Utilities Guide and Reference*

Related tasks:

- "Using import" in the *Data Movement Utilities Guide and Reference*

Related reference:

- "db2Import - Import" on page 412
- "db2look - DB2 Statistics and DDL Extraction Tool Command" in the *Command Reference*
- "Import Sessions - CLP Examples" in the *Data Movement Utilities Guide and Reference*
- "LOAD" on page 304
- "File type modifiers for import" in the *Command Reference*
- "Delimiter restrictions for moving data" in the *Command Reference*

INSPECT

Inspect database for architectural integrity, checking the pages of the database for page consistency. The inspection checks the structures of table objects and structures of table spaces are valid.

Scope:

In a single-partition system, the scope is that single partition only. In a partitioned database system, it is the collection of all logical partitions defined in db2nodes.cfg.

Authorization:

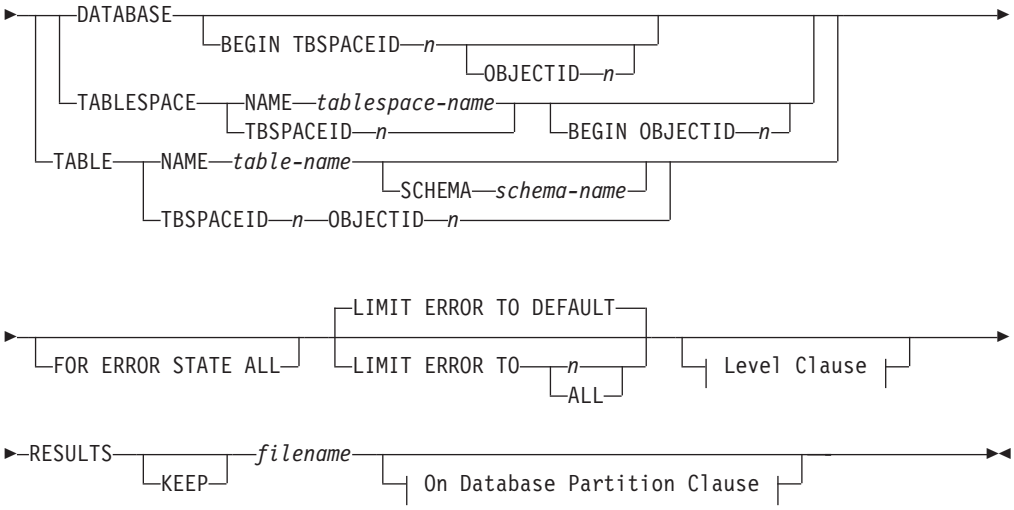
For INSPECT CHECK, one of the following:

- *sysadm*
- *dbadm*
- *sysctrl*
- *sysmaint*
- CONTROL privilege if single table.

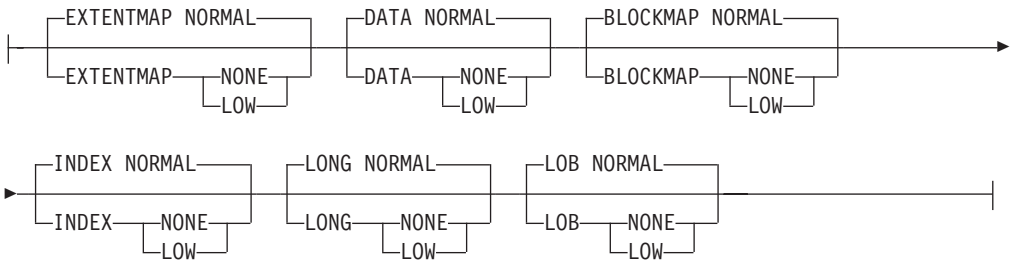
Required Connection:

Database

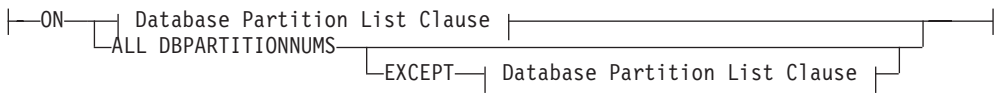
Command Syntax:



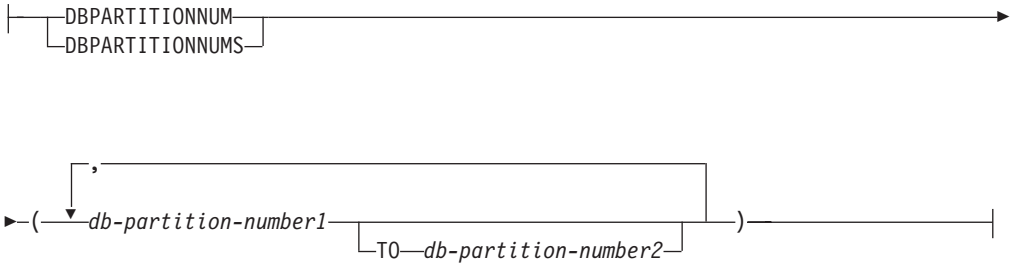
Level Clause:



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.

BEGIN TBSPACEID n 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.

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.

LIMIT ERROR TO *n*
Number of pages in error for an object to limit reporting for. 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.

LIMIT ERROR TO DEFAULT
Default number of pages in error for an object to limit reporting for. This value is the extent size of the object. This parameter is the default.

LIMIT ERROR TO ALL
No limit on number of pages in error reported.

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.

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.

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.

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 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 table space 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.
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. 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.

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:

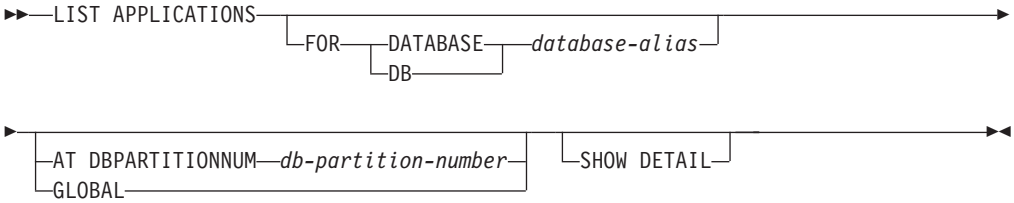
One of the following:

- *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 program name
- Application handle
- Application ID
- Database name.

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 system.

SHOW DETAIL
Output will include the following additional information:

- Sequence #
- Application status
- Status change time
- Database path.

Note: 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:

The following is sample output from LIST APPLICATIONS:

```
Auth Id  Application    Appl.      Application Id                DB       # of
         Name           Handle                                   Name    Agents
-------- -------------- ---------- ----------------------------- -------- -----
smith    db2bp_32       12         *LOCAL.smith.970220191502     TEST     1
smith    db2bp_32       11         *LOCAL.smith.970220191453     SAMPLE   1
```

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.

Compatibilities:

For compatibility with versions earlier than Version 8:

- The keyword NODE can be substituted for DBPARTITIONNUM.

Related reference:

- "GET SNAPSHOT Command" in the *Command Reference*
- "FORCE APPLICATION Command" in the *Command Reference*

LOAD

Loads data into a DB2 table. Data residing on the server can be in the form of a file, tape, or named pipe. Data residing on a remotely connected client can be in the form of a fully qualified file or named pipe. Data can also be loaded from a user-defined cursor.

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:

- *sysadm*
- *dbadm*
- 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.

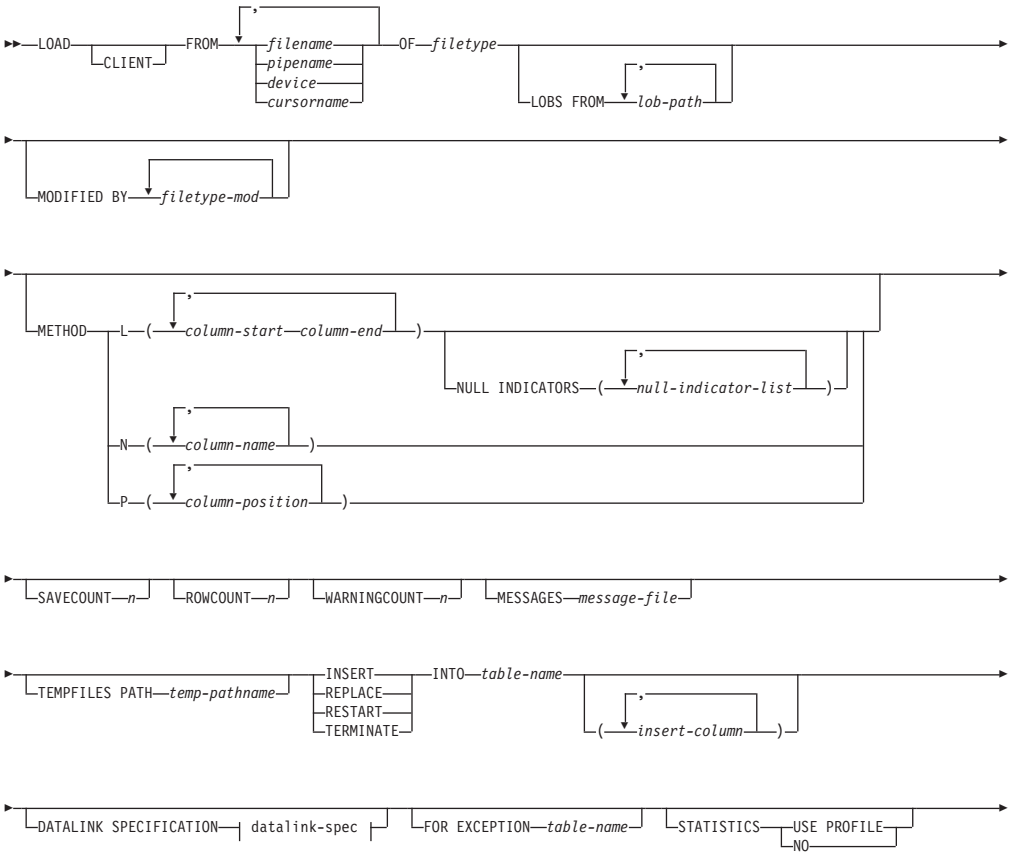
Since all load processes (and all DB2 server processes, in general) are owned by the instance owner, and all of these processes use the identification of the instance owner to access needed files, the instance owner must have read access to input data files. These input data files must be readable by the instance owner, regardless of who invokes the command.

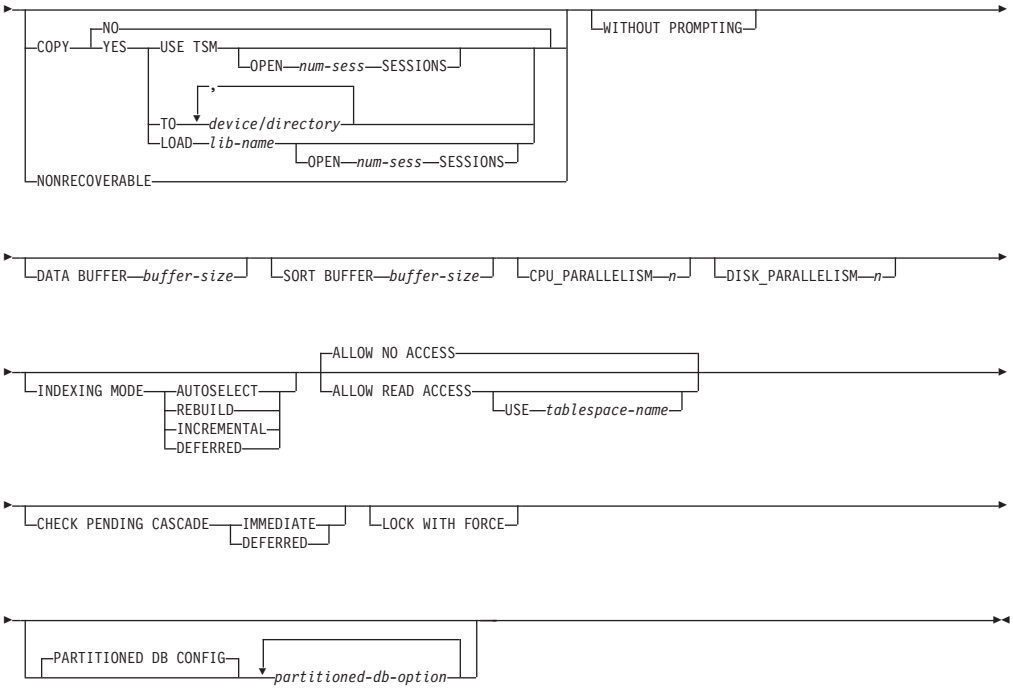
Required connection:

Database. If implicit connect is enabled, a connection to the default database is established.

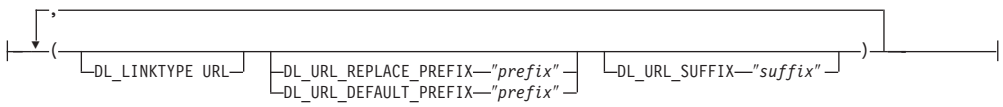
Instance. An explicit attachment is not required. If a connection to the database has been established, an implicit attachment to the local instance is attempted.

Command syntax:





datalink-spec:



Command parameters:

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 CHECK PENDING as well as LOAD IN PROGRESS. The SET INTEGRITY statement must be used to take the table out of CHECK PENDING.

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 CHECK PENDING as well as LOAD IN PROGRESS, and READ ACCESS. At the end of the load the table state LOAD IN PROGRESS state will be

removed but the table states CHECK PENDING and READ ACCESS will remain. The SET INTEGRITY statement must be used to take the table out of CHECK PENDING. While the table is in CHECK PENDING and READ ACCESS, 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.

CHECK PENDING CASCADE
If LOAD puts the table into a check pending state, the CHECK PENDING CASCADE option allows the user to specify whether or not the check 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 the check pending state (read or no access mode) for foreign key constraints is immediately extended to all descendent foreign key tables. If the table has descendent immediate materialized query tables or descendent immediate staging tables, the check pending state is extended immediately to the materialized query tables and the staging tables. Note that for a LOAD INSERT operation, the check 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 check pending read state will be put into check pending no access state.

DEFERRED
Indicates that only the loaded table will be placed in the check pending state (read or no access mode). 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 the check pending no access 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 the check pending no access 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 the check pending state. See the Notes section of the SET INTEGRITY statement in the SQL Reference for when these descendent tables will be put into the check pending state.

If the CHECK PENDING CASCADE option is not specified:

- Only the loaded table will be placed in the check 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 the check pending state when the loaded table is checked for constraint violations.

If LOAD does not put the target table into check pending state, the CHECK PENDING CASCADE option is ignored.

CLIENT
Specifies that the data to be loaded resides on a remotely connected client. This option is ignored if the load operation is not being invoked from a remote client. This option is not supported in conjunction with the CURSOR filetype.

Notes:

1. The `dumpfile` and `lobsinfile` modifiers refer to files on the server even when the CLIENT keyword is specified.
2. Code page conversion is not performed during a remote load operation. If the code page of the data is different from that of the server, the data code page should be specified using the `codepage` modifier.

In the following example, a data file (`/u/user/data.del`) residing on a remotely connected client is to be loaded into MYTABLE on the server database:

```
db2 load client from /u/user/data.del of del
   modified by codepage=850 insert into mytable
```

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.

Note: Index refresh is done automatically by the database when the index is accessed by a query.

COPY YES
Specifies that a copy of the loaded data will be saved. This option is invalid if forward recovery is disabled (both *logretain* and *userexit* are off). The option is not supported for tables with DATALINK columns.

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.

CPU_PARALLELISM n
Specifies the number of processes or threads that the load utility will spawn 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, or has not been specified, the load utility uses an intelligent default value (usually based on the number of CPUs available) at run time.

Notes:

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.

DATA BUFFER buffer-size
Specifies 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 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.

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.

DATALINK SPECIFICATION
For each DATALINK column, there can be one column specification enclosed by parentheses. Each column specification consists of one or more DL_LINKTYPE, prefix, and a DL_URL_SUFFIX specification. The prefix specification can be either DL_URL_REPLACE_PREFIX or DL_URL_DEFAULT_PREFIX.

There can be as many DATALINK column specifications as the number of DATALINK columns defined in the table. The order of specifications follows the order of DATALINK columns found within the *insert-column* list, or within the table definition (if an *insert-column* list is not specified).

DISK_PARALLELISM n
Specifies the number of processes or threads that the load utility will spawn 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.

DL_LINKTYPE
If specified, it should match the LINKTYPE of the column definition. Thus, DL_LINKTYPE URL is acceptable if the column definition specifies LINKTYPE URL.

DL_URL_DEFAULT_PREFIX ″**prefix**″
If specified, it should act as the default prefix for all DATALINK values within the same column. In this context, prefix refers to the ″scheme host port″ part of the URL specification.

Examples of prefix are:

```
"http://server"
"file://server"
"file:"
"http://server:80"
```

If no prefix is found in the column data, and a default prefix is specified with DL_URL_DEFAULT_PREFIX, the default prefix is prefixed to the column value (if not NULL).

For example, if DL_URL_DEFAULT_PREFIX specifies the default prefix `"http://toronto"`:

- The column input value ″/x/y/z″ is stored as ″http://toronto/x/y/z″.
- The column input value ″http://coyote/a/b/c″ is stored as ″http://coyote/a/b/c″.

- The column input value NULL is stored as NULL.

DL_URL_REPLACE_PREFIX "prefix"
This clause is useful when loading or importing data previously generated by the export utility, if the user wants to globally replace the host name in the data with another host name. If specified, it becomes the prefix for *all* non-NULL column values. If a column value has a prefix, this will replace it. If a column value has no prefix, the prefix specified by DL_URL_REPLACE_PREFIX is prefixed to the column value.

For example, if DL_URL_REPLACE_PREFIX specifies the prefix `"http://toronto"`:

- The column input value "/x/y/z" is stored as "http://toronto/x/y/z".
- The column input value "http://coyote/a/b/c" is stored as "http://toronto/a/b/c". Note that "toronto" replaces "coyote".
- The column input value NULL is stored as NULL.

DL_URL_SUFFIX "suffix"
If specified, it is appended to every non-NULL column value for the column. It is, in fact, appended to the "path" component of the data location part of the DATALINK value.

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. DATALINK exceptions are also captured in the exception table. 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 partitions on which the loading table is defined. The dump file, on the other hand, contains rows that cannot be loaded because they are invalid or have syntax errors.

FROM filename/pipename/device/cursorname
Specifies the file, pipe, device, or cursor referring to an SQL statement that contains the data being loaded. If the input source is a file, pipe, or device, it must reside on the database partition where the database resides, unless the CLIENT option is specified. If several names are specified, they will be processed in sequence. If the last item specified 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, 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.

Notes:

1. It is recommended that the fully qualified file name be used. If the server is remote, the fully qualified file name must be used. If the database resides on the same database partition as the caller, relative paths can be used.
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. If loading data that resides on a client machine, the data must be in the form of either a fully qualified file or a named pipe.

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.

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.

Incremental indexing is not supported when all of the following conditions are true:

- The LOAD COPY option is specified (*logretain* or *userexit* is enabled).
- The table resides in a DMS table space.
- The index object resides in a table space that is shared by other table objects belonging to the table being loaded.

To bypass this restriction, it is recommended that indexes be placed in a separate table space.

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.

Note: Deferred indexing is not supported for tables that have DATALINK columns.

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.

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,

```
db2 load from delfile1 of del modified by noeofchar noheader
   method P (1, 2, 3, 4, 5, 6, 7, 8, 9)
   insert into table1 (BLOB1, S2, I3, Int 4, I5, I6, DT7, I8, TM9)
```

will fail because of the `Int 4` column. The solution is to enclose such column names with double quotation marks:

```
db2 load from delfile1 of del modified by noeofchar noheader
   method P (1, 2, 3, 4, 5, 6, 7, 8, 9)
   insert into table1 (BLOB1, S2, I3, "Int 4", I5, I6, DT7, I8, TM9)
```

INTO table-name
Specifies the database table into which the data is to be loaded. This table cannot be a system table or a declared 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.

LOBS FROM lob-path
The path to the data files containing LOB values to be loaded. The path must end with a slash (/). If the CLIENT option is specified, the path must be fully qualified. 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. This option is ignored if `lobsinfile` is not specified within the *filetype-mod* string.

This option is not supported in conjunction with the CURSOR filetype.

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.

MESSAGES message-file

Specifies the destination for warning and error messages that occur during the load operation. If a message file is not specified, messages are written to standard output. If the complete path to the file is not specified, the load utility uses the current directory and the default drive as the destination. If the name of a file that already exists is specified, the utility appends the information.

The message file is usually populated with messages at the end of the load operation and, as such, is not suitable for monitoring the progress of the operation.

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.

Note: This method can only be used with ASC files, and is the only valid method for that file type.

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.

Note: 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.

Note: This method can only be used with file types IXF, DEL, or CURSOR, and is the only valid method for the DEL file type.

MODIFIED BY filetype-mod

Specifies file type modifier options. See File type modifiers for load.

NONRECOVERABLE

Specifies that the load transaction is to be marked as non-recoverable 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.

This option should not be used when DATALINK columns with the FILE LINK CONTROL attribute are present in, or being added to, the table.

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.

OF filetype

Specifies the format of the data:

- ASC (non-delimited ASCII format)
- DEL (delimited ASCII format)
- IXF (integrated exchange format, PC version), exported from the same or from another DB2 table
- CURSOR (a cursor declared against a SELECT or VALUES statement).

PARTITIONED DB CONFIG

Allows you to execute a load into a partitioned table. 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:

```
HOSTNAME x
FILE_TRANSFER_CMD x
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 Partitioned database load configuration options.

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.

This option is not supported for tables with DATALINK columns.

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.

RESTARTCOUNT
Reserved.

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.

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 not supported in conjunction with the CURSOR filetype.

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.

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.

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:

- 4 bytes for each duplicate or rejected row containing DATALINK values
- 136 bytes for each message that the load utility generates
- 15KB 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.

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.

The load terminate option will not remove a backup pending state from table spaces.

Note: This option is not supported for tables with DATALINK columns.

USING directory

Reserved.

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.

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.

If this option is not specified, and the tape device encounters an end of tape for the copy image, or the last item listed is a tape device, the user is prompted for a new tape on that device.

Examples:

Example 1

TABLE1 has 5 columns:

- COL1 VARCHAR 20 NOT NULL WITH DEFAULT
- COL2 SMALLINT
- COL3 CHAR 4
- COL4 CHAR 2 NOT NULL WITH DEFAULT
- COL5 CHAR 2 NOT NULL

ASCFILE1 has 6 elements:

- ELE1 positions 01 to 20
- ELE2 positions 21 to 22
- ELE5 positions 23 to 23
- ELE3 positions 24 to 27
- ELE4 positions 28 to 31
- ELE6 positions 32 to 32
- ELE6 positions 33 to 40

Data Records:

```
1...5....10...15...20...25...30...35...40
Test data 1         XXN 123abcdN
Test data 2 and 3   QQY    wxyzN
Test data 4,5 and 6 WWN6789    Y
```

The following command loads the table from the file:

```
db2 load from ascfile1 of asc modified by striptblanks reclen=40
   method L (1 20, 21 22, 24 27, 28 31)
   null indicators (0,0,23,32)
   insert into table1 (col1, col5, col2, col3)
```

Notes:

1. The specification of `striptblanks` in the MODIFIED BY parameter forces the truncation of blanks in VARCHAR columns (COL1, for example, which is 11, 17 and 19 bytes long, in rows 1, 2 and 3, respectively).
2. The specification of `reclen=40` in the MODIFIED BY parameter indicates that there is no new-line character at the end of each input record, and that each record is 40 bytes long. The last 8 bytes are not used to load the table.
3. Since COL4 is not provided in the input file, it will be inserted into TABLE1 with its default value (it is defined NOT NULL WITH DEFAULT).
4. Positions 23 and 32 are used to indicate whether COL2 and COL3 of TABLE1 will be loaded NULL for a given row. If there is a `Y` in the column's null indicator position for a given record, the column will be NULL. If there is an `N`, the data values in the column's data positions of the input record (as defined in L(.....)) are used as the source of column data for the row. In this example, neither column in row 1 is NULL; COL2 in row 2 is NULL; and COL3 in row 3 is NULL.
5. In this example, the NULL INDICATORS for COL1 and COL5 are specified as 0 (zero), indicating that the data is not nullable.
6. The NULL INDICATOR for a given column can be anywhere in the input record, but the position must be specified, and the `Y` or `N` values must be supplied.

Example 2 (Loading LOBs from Files)

TABLE1 has 3 columns:

- COL1 CHAR 4 NOT NULL WITH DEFAULT
- LOB1 LOB
- LOB2 LOB

ASCFILE1 has 3 elements:

- ELE1 positions 01 to 04
- ELE2 positions 06 to 13
- ELE3 positions 15 to 22

The following files reside in either `/u/user1` or `/u/user1/bin`:

- ASCFILE2 has LOB data
- ASCFILE3 has LOB data
- ASCFILE4 has LOB data
- ASCFILE5 has LOB data
- ASCFILE6 has LOB data
- ASCFILE7 has LOB data

Data Records in ASCFILE1:

```
1...5....10...15...20...25...30.
REC1 ASCFILE2 ASCFILE3
REC2 ASCFILE4 ASCFILE5
REC3 ASCFILE6 ASCFILE7
```

The following command loads the table from the file:

```
db2 load from ascfile1 of asc
   lobs from /u/user1, /u/user1/bin
   modified by lobsinfile reclen=22
   method L (1 4, 6 13, 15 22)
   insert into table1
```

Notes:

1. The specification of `lobsinfile` in the MODIFIED BY parameter tells the loader that all LOB data is to be loaded from files.
2. The specification of `reclen=22` in the MODIFIED BY parameter indicates that there is no new-line character at the end of each input record, and that each record is 22 bytes long.
3. LOB data is contained in 6 files, ASCFILE2 through ASCFILE7. Each file contains the data that will be used to load a LOB column for a specific row. The relationship between LOBs and other data is specified in ASCFILE1. The first record of this file tells the loader to place REC1 in COL1 of row 1. The contents of ASCFILE2 will be used to load LOB1 of row 1, and the contents of ASCFILE3 will be used to load LOB2 of row 1. Similarly, ASCFILE4 and ASCFILE5 will be used to load LOB1 and LOB2 of row 2, and ASCFILE6 and ASCFILE7 will be used to load the LOBs of row 3.
4. The LOBS FROM parameter contains 2 paths that will be searched for the named LOB files when those files are required by the loader.
5. To load LOBs directly from ASCFILE1 (a non-delimited ASCII file), without the `lobsinfile` modifier, the following rules must be observed:
   - The total length of any record, including LOBs, cannot exceed 32KB.
   - LOB fields in the input records must be of fixed length, and LOB data padded with blanks as necessary.
   - The `striptblanks` modifier must be specified, so that the trailing blanks used to pad LOBs can be removed as the LOBs are inserted into the database.

Example 3 (Using Dump Files)

Table FRIENDS is defined as:

```
table friends "( c1 INT NOT NULL, c2 INT, c3 CHAR(8) )"
```

If an attempt is made to load the following data records into this table,

```
23, 24, bobby
, 45, john
4,, mary
```

the second row is rejected because the first INT is NULL, and the column definition specifies NOT NULL. Columns which contain initial characters that are not consistent with the DEL format will generate an error, and the record will be rejected. Such records can be written to a dump file.

DEL data appearing in a column outside of character delimiters is ignored, but does generate a warning. For example:

```
22,34,"bob"
24,55,"sam" sdf
```

The utility will load ″sam″ in the third column of the table, and the characters ″sdf″ will be flagged in a warning. The record is not rejected. Another example:

```
22 3, 34,"bob"
```

The utility will load `22,34,"bob"`, and generate a warning that some data in column one following the `22` was ignored. The record is not rejected.

Example 4 (Loading DATALINK Data)

The following command loads the table MOVIETABLE from the input file `delfile1`, which has data in the DEL format:

```
db2 load from delfile1 of del
    modified by dldel|
    insert into movietable (actorname, description, url_making_of,
    url_movie) datalink specification (dl_url_default_prefix
    "http://narang"), (dl_url_replace_prefix "http://bomdel"
    dl_url_suffix ".mpeg") for exception excptab
```

Notes:

1. The table has four columns:

   ```
   actorname             VARCHAR(n)
   description           VARCHAR(m)
   url_making_of         DATALINK (with LINKTYPE URL)
   url_movie             DATALINK (with LINKTYPE URL)
   ```

2. The DATALINK data in the input file has the vertical bar (`|`) character as the sub-field delimiter.
3. If any column value for url_making_of does not have the prefix character sequence, ″http://narang″ is used.
4. Each non-NULL column value for url_movie will get ″http://bomdel″ as its prefix. Existing values are replaced.
5. Each non-NULL column value for url_movie will get ″.mpeg″ appended to the path. For example, if a column value of url_movie is ″http://server1/x/y/z″, it will be stored as ″http://bomdel/x/y/z.mpeg″; if the value is ″/x/y/z″, it will be stored as ″http://bomdel/x/y/z.mpeg″.

6. If any unique index or DATALINK exception occurs while loading the table, the affected records are deleted from the table and put into the exception table excptab.

Example 5 (Loading a Table with an Identity Column)

TABLE1 has 4 columns:

- C1 VARCHAR(30)
- C2 INT GENERATED BY DEFAULT AS IDENTITY
- C3 DECIMAL(7,2)
- C4 CHAR(1)

TABLE2 is the same as TABLE1, except that C2 is a GENERATED ALWAYS identity column.

Data records in DATAFILE1 (DEL format):

```
"Liszt"
"Hummel",,187.43, H
"Grieg",100, 66.34, G
"Satie",101, 818.23, I
```

Data records in DATAFILE2 (DEL format):

```
"Liszt", 74.49, A
"Hummel", 0.01, H
"Grieg", 66.34, G
"Satie", 818.23, I
```

Notes:

1. The following command generates identity values for rows 1 and 2, since no identity values are supplied in DATAFILE1 for those rows. Rows 3 and 4, however, are assigned the user-supplied identity values of 100 and 101, respectively.

   ```
   db2 load from datafile1.del of del replace into table1
   ```

2. To load DATAFILE1 into TABLE1 so that identity values are generated for all rows, issue one of the following commands:

   ```
   db2 load from datafile1.del of del method P(1, 3, 4)
      replace into table1 (c1, c3, c4)
   db2load from datafile1.del of del modified by identityignore
      replace into table1
   ```

3. To load DATAFILE2 into TABLE1 so that identity values are generated for each row, issue one of the following commands:

   ```
   db2 load from datafile2.del of del replace into table1 (c1, c3, c4)
   db2 load from datafile2.del of del modified by identitymissing
      replace into table1
   ```

4. To load DATAFILE1 into TABLE2 so that the identity values of 100 and 101 are assigned to rows 3 and 4, issue the following command:

   ```
   db2 load from datafile1.del of del modified by identityoverride
      replace into table2
   ```

   In this case, rows 1 and 2 will be rejected, because the utility has been instructed to override system-generated identity values in favor of user-supplied values. If user-supplied values are not present, however, the row must be rejected, because identity columns are implicitly not NULL.

5. If DATAFILE1 is loaded into TABLE2 without using any of the identity-related file type modifiers, rows 1 and 2 will be loaded, but rows 3 and 4 will be rejected, because they supply their own non-NULL values, and the identity column is GENERATED ALWAYS.

Example 6 (Loading using the CURSOR filetype)

Table ABC.TABLE1 has 3 columns:

```
ONE INT
TWO CHAR(10)
THREE DATE
```

Table ABC.TABLE2 has 3 columns:

```
ONE VARCHAR
TWO INT
THREE DATE
```

Executing the following commands will load all the data from ABC.TABLE1 into ABC.TABLE2:

```
db2 declare mycurs cursor for select two,one,three from abc.table1
db2 load from mycurs of cursor insert into abc.table2
```

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 check pending state. Summary tables that are defined with REFRESH IMMEDIATE, and that are dependent on tables being loaded, are also placed in check pending state. Issue the SET INTEGRITY statement to take the tables out of check pending state. Load operations cannot be carried out on replicated summary 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.

DB2 Data Links Manager considerations:

For each DATALINK column, there can be one column specification within parentheses. Each column specification consists of one or more of DL_LINKTYPE, *prefix* and a DL_URL_SUFFIX specification. The *prefix* information can be either DL_URL_REPLACE_PREFIX, or the DL_URL_DEFAULT_PREFIX specification.

There can be as many DATALINK column specifications as the number of DATALINK columns defined in the table. The order of specifications follows the order of DATALINK columns as found within the insert-column list (if specified by INSERT INTO (insert-column, ...)), or within the table definition (if insert-column is not specified).

For example, if a table has columns C1, C2, C3, C4, and C5, and among them only columns C2 and C5 are of type DATALINK, and the insert-column list is (C1, C5,

C3, C2), there should be two DATALINK column specifications. The first column specification will be for C5, and the second column specification will be for C2. If an insert-column list is not specified, the first column specification will be for C2, and the second column specification will be for C5.

If there are multiple DATALINK columns, and some columns do not need any particular specification, the column specification should have at least the parentheses to unambiguously identify the order of specifications. If there are no specifications for any of the columns, the entire list of empty parentheses can be dropped. Thus, in cases where the defaults are satisfactory, there need not be any DATALINK specification.

If data is being loaded into a table with a DATALINK column that is defined with FILE LINK CONTROL, perform the following steps before invoking the load utility. (If all the DATALINK columns are defined with NO LINK CONTROL, these steps are not necessary).

1. Ensure that the DB2 Data Links Manager is installed on the Data Links servers that will be referred to by the DATALINK column values.
2. Ensure that the database is registered with the DB2 Data Links Manager.
3. Copy to the appropriate Data Links servers, all files that will be inserted as DATALINK values.
4. Define the prefix name (or names) to the DB2 Data Links Managers on the Data Links servers.
5. Register the Data Links servers referred to by DATALINK data (to be loaded) in the DB2 Data Links Manager configuration file.

The connection between DB2 and the Data Links server might fail while running the load utility, causing the load operation to fail. If this occurs:

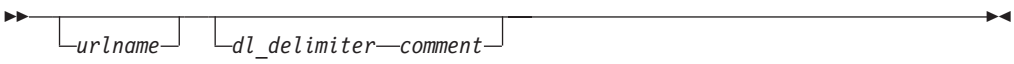
1. Start the Data Links server and the DB2 Data Links Manager.
2. Invoke a load restart operation.

Links that fail during the load operation are considered to be data integrity violations, and are handled in much the same way as unique index violations. Consequently, a special exception has been defined for loading tables that have one or more DATALINK columns.

Representation of DATALINK information in an input file

The LINKTYPE (currently only URL is supported) is not specified as part of DATALINK information. The LINKTYPE is specified in the LOAD or the IMPORT command, and for input files of type PC/IXF, in the appropriate column descriptor records.

The syntax of DATALINK information for a URL LINKTYPE is as follows:



Note that both *urlname* and *comment* are optional. If neither is provided, the NULL value is assigned.

urlname
The URL name must conform to valid URL syntax.

Notes:

1. Currently "http", "file", and "unc" are permitted as a schema name.
2. The prefix (schema, host, and port) of the URL name is optional. If a prefix is not present, it is taken from the DL_URL_DEFAULT_PREFIX or the DL_URL_REPLACE_PREFIX specification of the load or the import utility. If none of these is specified, the prefix defaults to "file://localhost". Thus, in the case of local files, the file name with full path name can be entered as the URL name, without the need for a DATALINK column specification within the LOAD or the IMPORT command.
3. Prefixes, even if present in URL names, are overridden by a different prefix name on the DL_URL_REPLACE_PREFIX specification during a load or import operation.
4. The "path" (after appending DL_URL_SUFFIX, if specified) is the full path name of the remote file in the remote server. Relative path names are not allowed. The http server default path-prefix is not taken into account.

dl_delimiter

For the delimited ASCII (DEL) file format, a character specified via the `dldel` modifier, or defaulted to on the LOAD or the IMPORT command. For the non-delimited ASCII (ASC) file format, this should correspond to the character sequence `\;` (a backslash followed by a semicolon). Whitespace characters (blanks, tabs, and so on) are permitted before and after the value specified for this parameter.

comment

The comment portion of a DATALINK value. If specified for the delimited ASCII (DEL) file format, the *comment* text must be enclosed by the character string delimiter, which is double quotation marks (") by default. This character string delimiter can be overridden by the MODIFIED BY *filetype-mod* specification of the LOAD or the IMPORT command.

If no comment is specified, the comment defaults to a string of length zero.

Following are DATALINK data examples for the delimited ASCII (DEL) file format:

- `http://www.almaden.ibm.com:80/mrep/intro.mpeg; "Intro Movie"`

  This is stored with the following parts:
  - scheme = http
  - server = www.almaden.ibm.com
  - path = /mrep/intro.mpeg
  - comment = "Intro Movie"
- `file://narang/u/narang; "InderPal's Home Page"`

  This is stored with the following parts:
  - scheme = file
  - server = narang
  - path = /u/narang
  - comment = "InderPal's Home Page"

Following are DATALINK data examples for the non-delimited ASCII (ASC) file format:

- `http://www.almaden.ibm.com:80/mrep/intro.mpeg\;Intro Movie`

  This is stored with the following parts:

  - scheme = http
  - server = www.almaden.ibm.com
  - path = /mrep/intro.mpeg
  - comment = "Intro Movie"
- `file://narang/u/narang\; InderPal's Home Page`

  This is stored with the following parts:
  - scheme = file
  - server = narang
  - path = /u/narang
  - comment = "InderPal's Home Page"

Following are DATALINK data examples in which the load or import specification for the column is assumed to be DL_URL_REPLACE_PREFIX ("http://qso"):

- `http://www.almaden.ibm.com/mrep/intro.mpeg`

  This is stored with the following parts:
  - schema = http
  - server = qso
  - path = /mrep/intro.mpeg
  - comment = NULL string
- `/u/me/myfile.ps`

  This is stored with the following parts:
  - schema = http
  - server = qso
  - path = /u/me/myfile.ps
  - comment = NULL string

Related concepts:

- "Load Overview" in the *Data Movement Utilities Guide and Reference*
- "Privileges, authorities, and authorizations required to use Load" on page 838

Related tasks:

- "Using Load" in the *Data Movement Utilities Guide and Reference*

Related reference:

- "QUIESCE TABLESPACES FOR TABLE" on page 340
- "db2atld - Autoloader Command" in the *Command Reference*
- "Load - CLP Examples" in the *Data Movement Utilities Guide and Reference*
- "Partitioned database load configuration options" in the *Data Movement Utilities Guide and Reference*
- "db2Load - Load" on page 437
- "File type modifiers for load" on page 326

File type modifiers for load

Table 51. Valid file type modifiers for load: 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.

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).
	Note: When this modifier is used, the table will be placed in CHECK PENDING state. To take the table out of CHECK PENDING state without verifying the user-supplied values, issue the following command after the load operation:
	`SET INTEGRITY FOR < table-name > GENERATED COLUMN IMMEDIATED UNCHECKED`
	
	To take the table out of CHECK 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.`
	
	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.

Table 51. Valid file type modifiers for load: All file formats (continued)

Modifier	Description
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. **Note:** 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 the percent free space threshold can be maintained. 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 registry variable DB2 INDEX FREE takes precedence over indexfreespace. The indexfreespace option affects index leaf pages only.
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 means nothing outside the context of the "lobsinfile" modifier. 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). The AutoLoader utility writes a header to each file contributing data to a table in a multiple-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.

Table 51. Valid file type modifiers for load: All file formats (continued)

Modifier	Description
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. **Note:** 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.
subtableconvert	Valid only when loading into a single sub-table. Typical usage is to export data from a regular table, and then to invoke a load operation (using this modifier) to convert the data into a single sub-table.
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. **Note:** 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: ",," is specified for the column • For DEL/ASC/WSF files: A row that does not have enough columns, or is not long enough for the original specification. Without this option, if a source column contains no data for a row instance, one of the following occurs: • If the column is nullable, a NULL is loaded • If the column is not nullable, the utility rejects the row.

Table 52. Valid file type modifiers for load: 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 52. Valid file type modifiers for load: 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.
	Notes:
	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. This modifier does not support file names with multiple file extensions. For example,
	`dumpfile = /home/svtdbm6/DUMP.FILE`
	is acceptable to the load utility, but
	`dumpfile = /home/svtdbm6/DUMP.LOAD.FILE`
	is not.

dumpfileaccessall = *x*	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-based operating system

Table 52. Valid file type modifiers for load: ASCII file formats (ASC/DEL) (continued)

Modifier	Description
fastparse	Reduced syntax checking is done on user-supplied column values, and performance is enhanced. Tables loaded under this option are guaranteed to be architecturally correct, and the utility is guaranteed to perform sufficient data checking to prevent a segmentation violation or trap. Data that is in correct form will be loaded correctly. For example, if a value of `123qwr4` were to be encountered as a field entry for an integer column in an ASC file, the load utility would ordinarily flag a syntax error, since the value does not represent a valid number. With `fastparse`, a syntax error is not detected, and an arbitrary number is loaded into the integer field. Care must be taken to use this modifier with clean data only. Performance improvements using this option with ASCII data can be quite substantial. This option is not supported in conjunction with the CURSOR or IXF file types.
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 52. Valid file type modifiers for load: 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)`
	`UUUUUU - Microsecond (6 digits ranging from 000000 - 999999; mutually exclusive with all other microsecond elements)`
	`UUUUU - Microsecond (5 digits ranging from 00000 - 99999, maps to range from 000000 - 999990; mutually exclusive with all other microseond elements)`
	`UUUU - Microsecond (4 digits ranging from 0000 - 9999, maps to range from 000000 - 999900; mutually exclusive with all other microseond elements)`
	`UUU - Microsecond (3 digits ranging from 000 - 999, maps to range from 000000 - 999000; mutually exclusive with all other microseond elements)`
	`UU - Microsecond (2 digits ranging from 00 - 99, maps to range from 000000 - 990000; mutually exclusive with all other microseond elements)`
	`U - Microsecond (1 digit ranging from 0 - 9, maps to range from 000000 - 900000; mutually exclusive with all other microseond elements)`
	`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`

noeofchar	The optional end-of-file character `x'1A'` is not recognized as the end of file. Processing continues as if it were a normal character.

Table 52. Valid file type modifiers for load: 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 or ASC 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.

Table 53. Valid file type modifiers for load: 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 `noeofchar` option is assumed. 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. **Note:** 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.

Table 53. Valid file type modifiers for load: ASC file formats (Non-delimited ASCII) (continued)

Modifier	Description

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. The `noeofchar` option is assumed.
	
	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 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. This option cannot be specified together with `striptnulls`. These are mutually exclusive options. **Note:** This option replaces the obsolete `t` option, which is supported for back-level 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. **Note:** This option replaces the obsolete `padwithzero` option, which is supported for back-level 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. The NOEOFCHAR option is assumed.
	
	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 54. Valid file type modifiers for load: 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 wish 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]
datesiso	Date format. Causes all date data values to be loaded in ISO format.
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]

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>.

dldel*x*	*x* is a single character DATALINK delimiter. The default value is a semicolon (;). The specified character is used in place of a semicolon as the inter-field separator for a DATALINK value. It is needed because a DATALINK value can have more than one sub-value. [234] **Note:** *x* must not be the same character specified as the row, column, or character string 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 tailing 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`

Table 54. Valid file type modifiers for load: DEL file formats (Delimited ASCII) (continued)

Modifier	Description
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 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 55. Valid file type modifiers for load: 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.

Notes:

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. The character 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 restrictions for moving data lists restrictions that apply to the characters that can be used as delimiter overrides.
4. Even if the DATALINK delimiter character is a valid character within the URL syntax, it will lose its special meaning within the scope of the load operation.
5. 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.

Table 56. 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.
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.

Related reference:

- "LOAD" on page 304
- "db2Load - Load" on page 437
- "Delimiter restrictions for moving data" in the *Command Reference*

MIGRATE DATABASE

Converts previous versions of DB2 databases to current formats.

Attention: The database pre-migration tool must be run prior to DB2 Version 8 installation (on Windows operating systems), or before instance migration (on UNIX based systems), because it cannot be executed on DB2 Version 8. On Windows the pre-migration tool is db2ckmig. On UNIX systems, db2imigr performs similar tasks. Backup all databases prior to migration, and prior to DB2 Version 8 installation on Windows operating systems.

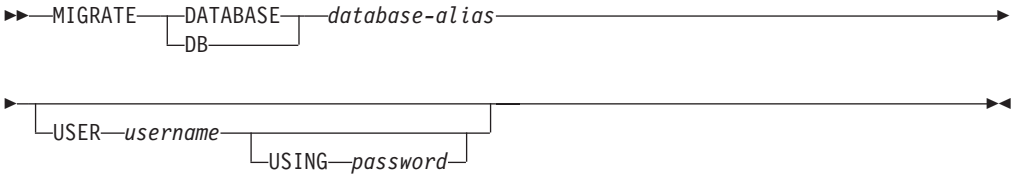
Authorization:

sysadm

Required connection:

This command establishes a database connection.

Command syntax:



Command parameters:

DATABASE database-alias
Specifies the alias of the database to be migrated to the currently installed version of the database manager.

USER username
Identifies the user name under which the database is to be migrated.

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 migrates the database cataloged under the database alias `sales`:

```
db2 migrate database sales
```

Usage notes:

This command 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.

If an error occurs during migration, it might be necessary to issue the TERMINATE command before attempting the suggested user response. For example, if a log full error occurs during migration (SQL1704: Database migration 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 migration failure occurs after the database has already been relocated (which is likely to be the case when a ″log full″ error returns).

When a database is migrated to Version 8, 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.

Related reference:

- "TERMINATE Command" in the *Command Reference*

QUIESCE

Forces all users off the specified instance and database and puts it into a quiesced mode. In quiesced mode, users cannot connect from outside of the database engine. While the database instance or database is in quiesced mode, you can perform administrative tasks on it. After administrative tasks are complete, use the UNQUIESCE command to activate the instance and database and allow other users to connect to the database but avoid having to shut down and perform another database start.

In this mode only users with authority in this restricted mode are allowed to attach or connect to the instance/database. Users with *sysadm*, *sysmaint*, and *sysctrl* authority always have access to an instance while it is quiesced, and users with *sysadm* and *dbadm*authority always have access to a database while it is quiesced.

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.

QUIESCE INSTANCE *instance-name* means the instance and the databases in the instance *instance-name* will be in quiesced mode. The instance will be accessible just for *sysadm*, *sysmaint*, and *sysctrl* and allowed user/group.

If an instance is in quiesced mode, a database in the instance cannot be put in quiesced mode.

Authorization:

One of the following:

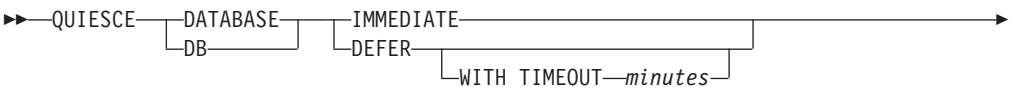
For database level quiesce:

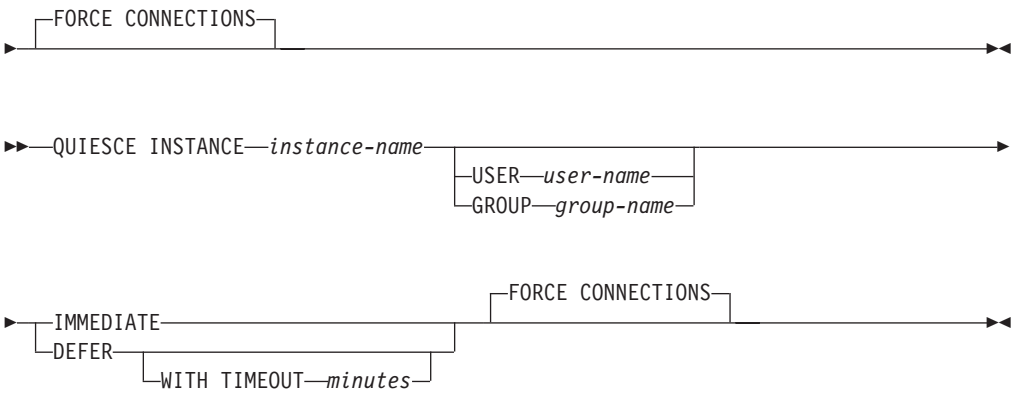
- *sysadm*
- *dbadm*

For instance level quiesce:

- *sysadm*
- *sysctrl*

Command syntax:





Required connection:
Database

(Database connection is not required for an instance quiesce.)

Command parameters:

DEFER
Wait for applications until they commit the current unit of work.

WITH TIMEOUT
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_timeout* 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.

INSTANCE *instance-name*
The instance *instance-name* and the databases in the instance will be placed in quiesced mode. The instance will be accessible only to users with *sysadm*, *sysmaint*, and *sysctrl* authority and specified users in specified groups.

USER *user-name*
Specifies the name of a user who will be allowed access to the instance while it is quiesced.

GROUP *group-name*
Specifies the name of a group that will be allowed access to the instance while the instance is quiesced.

Examples:

In the following example, the default behavior is to force connections, so it does not need to be explicitly stated and can be removed from this example.

```
db2 quiesce instance crankarm user frank immediate force connections
```

The following example forces off all users with connections to the database.

```
db2 quiesce db immediate
```

- The first example will quiesce the instance `crankarm`, while allowing user `frank` to continue using the database.

  The second example will quiesce the database you are attached to, preventing access by all users except those with one of the following authorities: *sysadm*, *sysmaint*, *sysctrl*, or *dbadm*.
- This command will force all users off the database or instance if FORCE CONNECTION option is supplied. FORCE CONNECTION is the default behavior; the parameter is allowed in the command for compatibility reasons.
- The command will be synchronized with the FORCE and will only complete once the FORCE has completed.

Usage notes:

- After QUIESCE INSTANCE, only users with *sysadm*, *sysmaint*, or *sysctrl* authority or a user name and group name provided as parameters to the command can connect to the instance.
- 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, and 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 node. It quiesces only that portion of table spaces belonging to the node on which the load operation is performed.

Authorization:

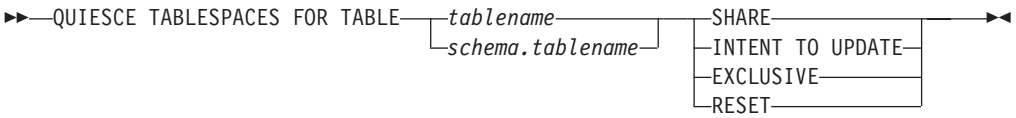
One of the following:

- *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.

Examples:

```
db2 quiesce tablespaces for table staff share

db2 quiesce tablespaces for table boss.org intent to update
```

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:

   ```
   db2 quiesce tablespaces for table mytable exclusive
   ```

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.

Related reference:

- "LOAD" on page 304

RECONCILE

Validates the references to files for the DATALINK data of a table. The rows for which the references to files cannot be established are copied to the exception table (if specified), and modified in the input table.

Reconcile produces a message file (reconcil.msg) in the instance path on UNIX based systems, and in the install path on Windows platforms. This file will contain warning and error messages that are generated during validation of the exception table.

Authorization:

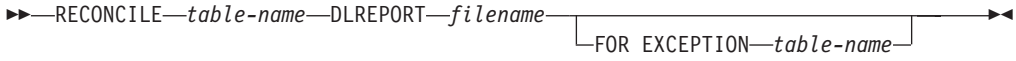
One of the following:

- *sysadm*
- *sysctrl*
- *sysmaint*
- *dbadm*
- CONTROL privilege on the table.

Required connection:

Database

Command syntax:



Command parameters:

RECONCILE table-name
Specifies the table on which reconciliation is to be performed. 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 authorization ID.

DLREPORT filename
Specifies the file that will contain information about the files that are unlinked during reconciliation. The name must be fully qualified (for example, `/u/johnh/report`). The reconcile utility appends a `.ulk` extension to the specified file name (for example, `report.ulk`). When no table is provided with the FOR EXCEPTION clause, a `.exp` file extension is appended to the exception report file.

FOR EXCEPTION table-name
Specifies the exception table into which rows that encounter link failures for DATALINK values are to be copied. If no table is specified, an exception report file is generated in the directory specified in the "DLREPORT" option.

Examples:

The following command reconciles the table DEPT, and writes exceptions to the exception table EXCPTAB, which was created by the user. Information about files that were unlinked during reconciliation is written into the file `report.ulk`, which is created in the directory `/u/johnh`. If `FOR EXCEPTION excptab` had not been specified, the exception information would have been written to the file `report.exp`, created in the `/u/johnh` directory.

```
db2 reconcile dept dlreport /u/johnh/report for exception excptab
```

Usage notes:

During reconciliation, attempts are made to link files which exist according to table data, but which do not exist according to Data Links File Manager metadata, if no other conflict exists. A required DB2 Data Links Manager is one which has a referenced DATALINK value in the table. Reconcile tolerates the unavailability of a required DB2 Data Links Manager as well as other DB2 Data Links Managers that are configured to the database but are not part of the table data.

Reconciliation is performed with respect to all DATALINK data in the table. If file references cannot be reestablished, the violating rows are inserted into the exception table (if specified). These rows are not deleted from the input table. To ensure file reference integrity, the offending DATALINK values are nulled. If the column is defined as not nullable, the DATALINK values are replaced by a zero length URL.

If a file is linked under a DATALINK column defined with WRITE PERMISSION ADMIN and modified but not yet committed (that is, the file is still in the update-in-progress state), the reconciliation process renames the modified file to a filename with `.mod` as the suffix. It also removes the file from the update-in-progress state. If the DATALINK column is defined with RECOVERY YES, the previous archive version is restored.

If an exception table is not specified, the host name, file name, column ID, and reason code for each of the DATALINK column values for which file references could not be reestablished are copied to an exception report file (`<`*`filename`*`>.exp`). If the file reference could not be reestablished because the DB2 Data Links Manager is unavailable or was dropped from the database using the DROP DATALINKS MANAGER command, the file name reported in the exception report file is not the full file name. The prefix will be missing. For example, if the original DATALINK value was http://host.com/dlfs/x/y/a.b, the value reported in the exception table will be http://host.com/x/y/a.b. The prefix name 'dlfs' will not be included.

If the DATALINK column is defined with RECOVERY YES, the previous archive version is restored.

At the end of the reconciliation process, the table is taken out of datalink reconcile pending (DRP) state only if reconcile processing is complete on all the required DB2 Data Links Managers. If reconcile processing is pending on any of the required DB2 Data Links Managers (because they were unavailable), the table will remain, or be placed, in DRP state. If for some reason, an exception occurred on one of the affected Data Links Managers such that the reconciliation could not be completed successfully, the table might also be placed in a DRNP state, for which further manual intervention will be required before full referential integrity for that table can be restored.

The exception table, if specified, should be created before the reconcile utility is run. The exception table used with the reconcile utility is identical to the exception table used by the load utility.

The exception table mimics the definition of the table being reconciled. It can have one or two optional columns following the data columns. The first optional column is the TIMESTAMP column. It will contain the time stamp for when the reconcile operation was started. The second optional column should be of type CLOB (32KB or larger). It will contain the IDs of columns with link failures, and the reasons for those failures. The DATALINK columns in the exception table should specify NO LINK CONTROL. This ensures that a file is not linked when a row (with a

DATALINK column) is inserted, and that an access token is not generated when rows are selected from the exception table.

Information in the MESSAGE column is organized according to the following structure:

Field number	Content	Size	Comments
1	Number of violations	5 characters	Right justified padded with '0'
2	Type of violation	1 character	'L' - DATALINK violation
3	Length of violation	5 characters	Right justified padded with '0'
4	Number of violating DATALINK columns	4 characters	Right justified padded with '0'
5	DATALINK column number of the first violating column	4 characters	Right justified padded with '0'
6	Reason for violation	5 characters	Right justified padded with '0'
			Repeat Fields 5 and 6 for each violating column

The following is a list of possible violations:

```
00001-File could not be found by DB2 Data Links Manager.
00002-File already linked.
00003-File in modified state.
00004-Prefix name not registered.
00005-File could not be retrieved.
00006-File entry missing. This will happen for
      RECOVERY NO, READ PERMISSION FS, WRITE PERMISSION FS DATALINK
      columns. Use update to relink the file.
00007-File is in unlink state.
00008-File restored but modified file has been copied to
      <filename>.MOD
00009-File is already linked to another table.
00010-DB2 Data Links Manager referenced by the DATALINK
      value has been dropped from the database using the
      DROP DATALINKS MANAGER command.
00999-File could not be linked.

Example:

00001L00022000200040002000500001

   00001 - Specifies that the number of violations is 1.
   L     - Specifies that the type of violation is 'DATALINK violation'.
   00022 - Specifies that the length of the violation is 12 bytes.
   0002  - Specifies that there are 2 columns in the row which
           encountered link failures.
   0004,00002
   0005,00001 - Specifies the column ID and the reason for the violation.
```

If the message column is present, the time stamp column must also be present.

Related concepts:

- "Failure and recovery overview" in the *DB2 Data Links Manager Administration Guide and Reference*

REORG INDEXES/TABLE

Reorganizes an index or a table.

The index option reorganizes all indexes defined on a table by rebuilding the index data into unfragmented, physically contiguous pages. If you specify the CLEANUP ONLY option of the index option, cleanup is performed without rebuilding the indexes. This command cannot be used against indexes on declared 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*
- CONTROL privilege on the table.

Required connection:

Database

Command syntax:

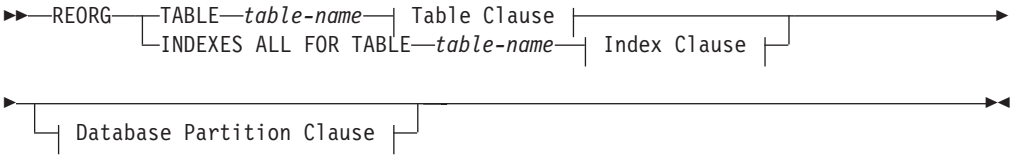
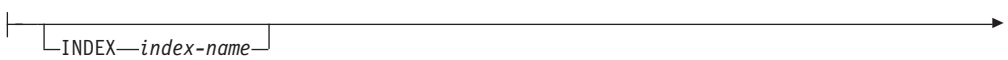
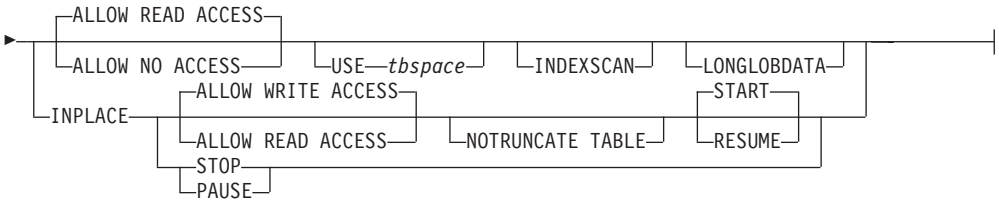
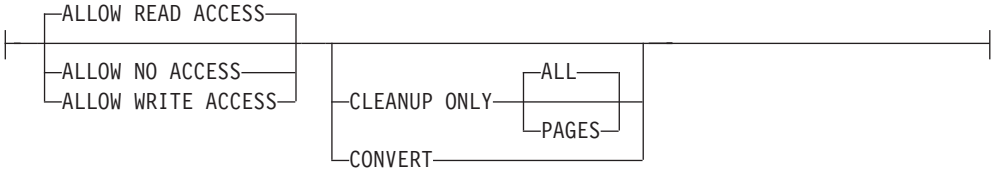


Table Clause:

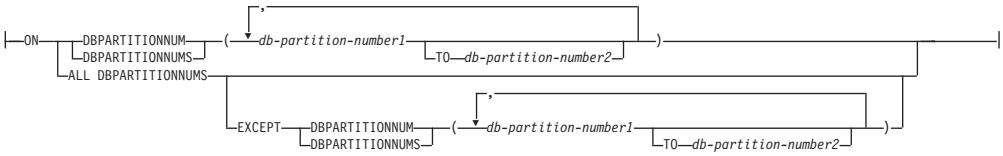




Index Clause:



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.

ALLOW NO ACCESS
Specifies that no other users can access the table while the indexes are being reorganized.

ALLOW READ ACCESS
Specifies that other users can have read-only access to the table while the indexes are being reorganized. This is the default.

ALLOW WRITE ACCESS
Specifies that other users can read from and write to the table while the indexes are being reorganized.

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
If you are not sure whether the table you are operating on has a type-1 or type-2 index, but want type-2 indexes, you can use the CONVERT option. If the index is type 1, this option will convert it into type 2. If the index is already type 2, this option has no effect.

All indexes created by DB2 prior to Version 8 are type-1 indexes. All indexes created by Version 8 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 will also be of type 1.

Using the INSPECT command to determine the index type can be slow. CONVERT allows you to ensure that the new index will be Type 2 without your needing to determine its original type.

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. Note that, 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 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.

Note: 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. Also note that 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.

INPLACE
Reorganizes the table while permitting user access.

In place table reorganization is allowed only on tables with type-2 indexes and without extended indexes. 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.

USE tablespace-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, the page size of any system temporary table space that you specify must match the page size of the table spaces in which the table data resides, including any LONG or LOB column data.

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.

Examples:

For a classing REORG TABLE like the default in DB2, Version 7, enter the following command:

```
db2 reorg table employee index empid allow no access indexscan
 longlobdata
```

Note that the defaults are different in DB2, Version 8.

To reorganize a table to reclaim space and use the temporary table space `mytemp1`, enter the following command:

```
db2 reorg table homer.employee use mytemp1
```

To reorganize tables in a partitiongroup consisting of nodes 1, 2, 3, and 4 of a four-node system, you can enter either of the following commands:

```
db2 reorg table employee index empid on dbpartitionnum (1,3,4)

db2 reorg table homer.employee index homer.empid on all
  dbpartitionnums except dbpartitionnum (2)
```

To clean up the pseudo deleted keys and pseudo empty pages in all the indexes on the EMPLOYEE table while allowing other transactions to read and update the table, enter:

```
db2 reorg indexes all for table homer.employee allow write
 access cleanup only
```

To clean up the pseudo empty pages in all the indexes on the EMPLOYEE table while allowing other transactions to read and update the table, enter:

```
db2 reorg indexes all for table homer.employee allow write
 access cleanup only pages
```

To reorganize the EMPLOYEE table using the system temporary table space TEMPSPACE1 as a work area, enter:

```
db2 reorg table homer.employee use tempspace1
```

To start, pause, and resume an in place reorg of the EMPLOYEE table with the default schema HOMER, which is specified explicitly in previous examples, enter the following commands:

```
db2 reorg table employee index empid inplace start
db2 reorg table employee inplace pause
db2 reorg table homer.employee inplace allow read access
 notruncate table resume
```

Note that the command to resume the reorg contains additional keywords to specify read access only and to skip the truncation step, which share-locks the table.

Usage notes:

Restrictions:

- The REORG utility does not support the use of nicknames.
- The REORG TABLE command is not supported for declared 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 DMS table while an online backup of a table space in which the table resides is being performed.
- REORG TABLE cannot use an index that is based on an index extension.

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 INDEXES can be used to reorganize all of the indexes on a table, to remove any fragmentation and restore physical clustering to the leaf pages. Use REORGCHK to help determine if an index needs reorganizing. Be sure to complete all database operations and release all locks before invoking REORG INDEXES. This can be done by issuing a COMMIT after closing all cursors opened WITH HOLD, or by issuing a ROLLBACK.

Indexes might not be optimal following an in-place REORG TABLE operation, since only the data object and not the indexes are reorganized. It is recommended that you perform a REORG INDEXES after an in place REORG TABLE operation. Indexes are completely rebuilt during the last phase of a classic REORG TABLE, however, so reorganizing indexes is not necessary.

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 REORGCHK 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 partitioned onto several database partitions, and the table reorganization fails on any of the affected database partitions, only the failing database partitions will have the table reorganization rolled back.

Note: 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).

Related reference:

- "GET SNAPSHOT Command" in the *Command Reference*
- "REORGCHK Command" in the *Command Reference*
- "RUNSTATS Command" in the *Command Reference*
- "REBIND" on page 866
- "SNAPSHOT_TBREORG table function" in the *SQL Administrative Routines*

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.

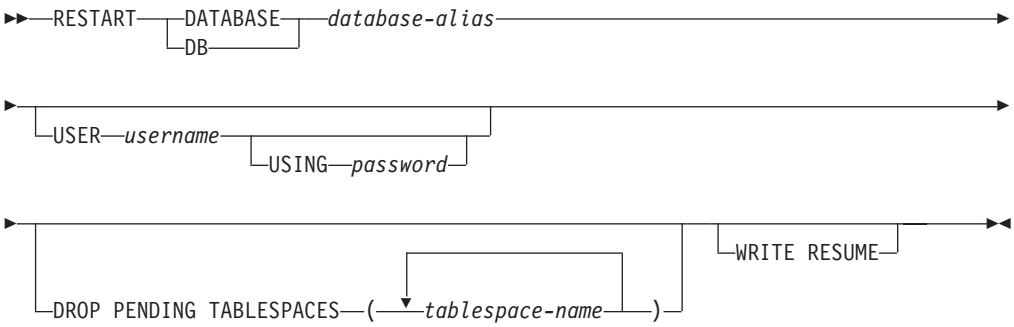
Authorization:

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.

Note: 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).

At the completion of RESTART DATABASE, 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 RESTART DATABASE must be issued before the database can be used again. Use the LIST INDOUBT TRANSACTIONS command to generate a list of indoubt transactions.

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.

Related tasks:

- "Manually resolving indoubt transactions" in the *Administration Guide: Planning*

Related reference:

- "LIST INDOUBT TRANSACTIONS Command" in the *Command Reference*

RESTORE DATABASE

Rebuilds a damaged or corrupted database that has been backed up using the DB2 backup utility. The restored database is in the same state it was in when the backup copy was made. This utility can also overwrite a database with a different image, or restore to a new database.

You can restore databases created on a DB2 Version 8 32-bit Windows platform to a DB2 Version 8 64-bit Windows platform, or the reverse. You can restore databases created on a DB2 Version 8 32-bit Linux (Intel) platform to a DB2 Version 8 64-bit Linux (Intel) platform, or the reverse. You can restore databases created on DB2 Version 8 AIX, HP-UX, or the Solaris Operating Environment platforms, in 32-bit or 64-bit, to DB2 Version 8 AIX, HP-UX, or Solaris Operating Environment platforms (32-bit or 64-bit).

The restore utility can also be used to restore backup images that were produced on a previous version of DB2 (up to two versions earlier) as long as the word size (32-bit or 64-bit) is the same. Cross-platform restore operations from a backup image created with a previous version of DB2 are not supported. If a migration 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 the state it was in prior to the occurrence of the damage or corruption by invoking the rollforward utility after successful completion of a restore operation.

This utility can also restore a table space level backup.

Incremental and delta images cannot be restored when there is a difference in operating systems or word size (32-bit or 64-bit).

Following a successful restore 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 to the same environment.)

Even with a successful restore 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.)

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*

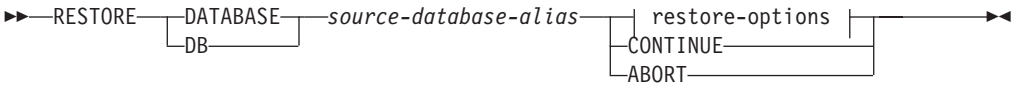
Required connection:

The required connection will vary based on the type of restore action you wish to do:

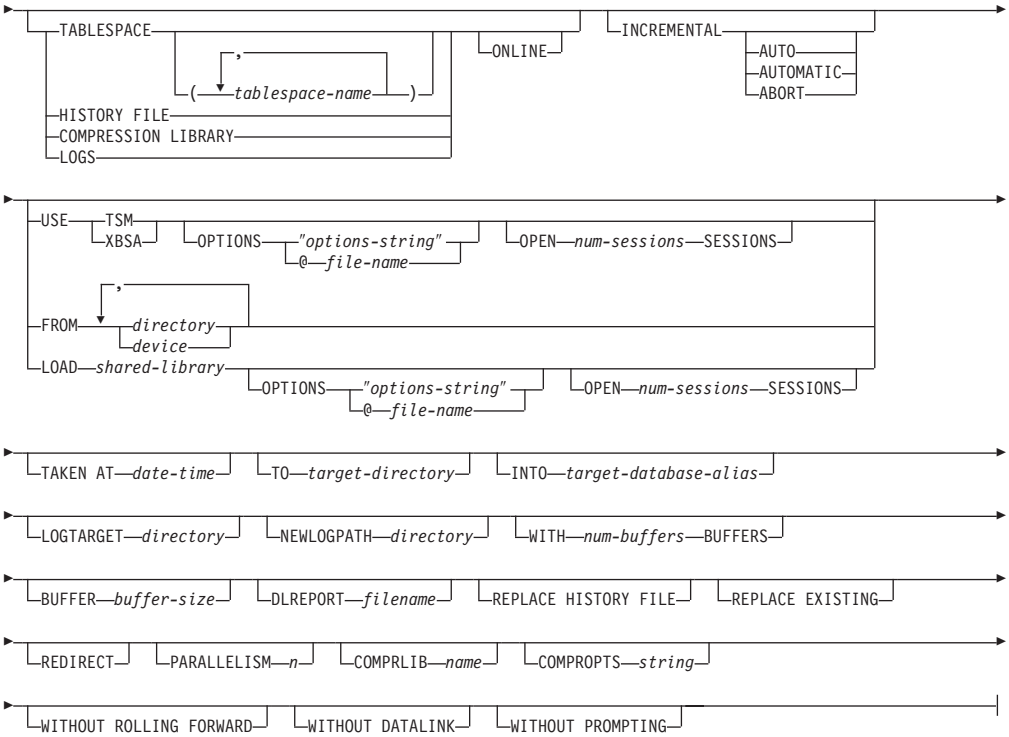
- Database, to restore to an existing database. This command automatically establishes an exclusive connection to the specified database.
- Instance and database, to restore to a new database. The instance attachment is required to create the database.

  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.

Command syntax:



restore-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.

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 TSM-managed output.

OPTIONS

"options-string"
Specifies options to be used for the restore operation.The string will be passed to the vendor support library, for example TSM, exactly as it was entered, without the quotes.

Note: Specifying this option overrides the value specified by the VENDOROPT database configuration parameter.

@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, for example TSM. 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.

USE 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.

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 database server.

On Windows operating systems, the specified directory must not be a DB2-generated directory. For example, given the following commands:

```
db2 backup database sample to c:\backup
db2 restore database sample from c:\backup
```

Using these commands, DB2 generates subdirectories under the `c:\backup` directory to allow more than one backup to be placed in the specified top level directory. The DB2-generated subdirectories should be ignored. To specify precisely which backup image to restore, use the TAKEN AT parameter. There can be several backup images stored on the same path.

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*
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.

Note: On Windows operating systems, when using this parameter, specify only the drive letter. If you specify a path, an error is returned.

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*
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.

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. DB2 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. 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.

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.

DLREPORT *filename*
The file name, if specified, must be specified as an absolute path. Reports the files that become unlinked, as a result of a fast reconcile, during a restore operation. This option is only to be used if the table being restored has a DATALINK column type and linked files.

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.

Note: All commands associated with a single redirected restore operation must be invoked from the same window or CLP session.

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.

WITHOUT DATALINK
Specifies that any tables with DATALINK columns are to be put in DataLink_Reconcile_Pending (DRP) state, and that no reconciliation of linked files is to be performed.

PARALLELISM *n*
Specifies the number of buffer manipulators that are to be spawned during the restore operation. DB2 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. 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 will fail.

COMPROPTS *string*
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 decompression 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.

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 partitions, numbered 0 through 3. The path /dev3/backup is accessible from all 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.

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.

      ```
      db2 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:

      ```
      db2 restore db mydb abort
      ```

   b. Issue a SET TABLESPACE CONTAINERS command for each table space whose containers must be redefined. For example:

      ```
      db2 set tablespace containers for 5 using
         (file 'f:\ts3con1' 20000, file 'f:\ts3con2' 20000)
      ```

      To verify that the containers of the restored database are the ones specified in this step, issue the LIST TABLESPACE CONTAINERS command.

   c. After successful completion of steps 1 and 2, issue:

      ```
      db2 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 retreive 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.
   - Specfying a delimited string:

     ```
     db2 restore db sample use TSM options "-fromnode bar -fromowner dmcinnis"
     ```

   - Specifying a fully qualified file:

     ```
     db2 restore db sample use TSM options @/u/dmcinnis/myoptions.txt
     ```

     The file myoptions.txt contains the following information: -fromnode=bar -fromowner=dmcinnis

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 of the table spaces found in a table space image. Any RESTORE DATABASE command of the form `db2 restore db <name> tablespace` will perform a table space restore of the table spaces found in the image. Any RESTORE DATABASE command in which a list of table spaces is provided will perform a restore of whatever table spaces are explicitly listed.
- Following the restore of an online backup, you must perform a roll forward recovery.
- If a backup image is compressed, DB2 will detect this and automatically decompress the data before restoring it. If a library is specified on the db2Restore API, it will be used for decompressing the data. Otherwise, if a library that is stored in the backup image will be used. Otherwise, the data cannot be decompressed, so the restore will fail.
- If the compression library is to be restored from a backup image (either explicitly by specifying the DB2RESTORE_COMPR_LIB restore type 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.
-
- To restore log files from a backup image which contains them, the LOGTARGET option must be specified, providing a fully qualified and valid path which 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 which does not include logs, the restore will return an error before attempting to restore any table space data. A restore 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 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 can not be extracted, for any reason, the restore 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 already 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.
- It is also possible to 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 logs included in the target image of the restore operation will be retrived from the backup image. Any logs 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.

Related reference:

- "BACKUP DATABASE" on page 227
- "ROLLFORWARD DATABASE" on page 363
- "db2move - Database Movement Tool Command" in the *Command Reference*

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 partitions that are listed in the `db2nodes.cfg` file. A database or table space rollforward operation to the end of logs affects the partitions that are specified. If no partitions are specified, it affects all partitions that are listed in the `db2nodes.cfg` file; if rollforward recovery is not needed on a particular partition, that partition is ignored.

Authorization:

One of the following:

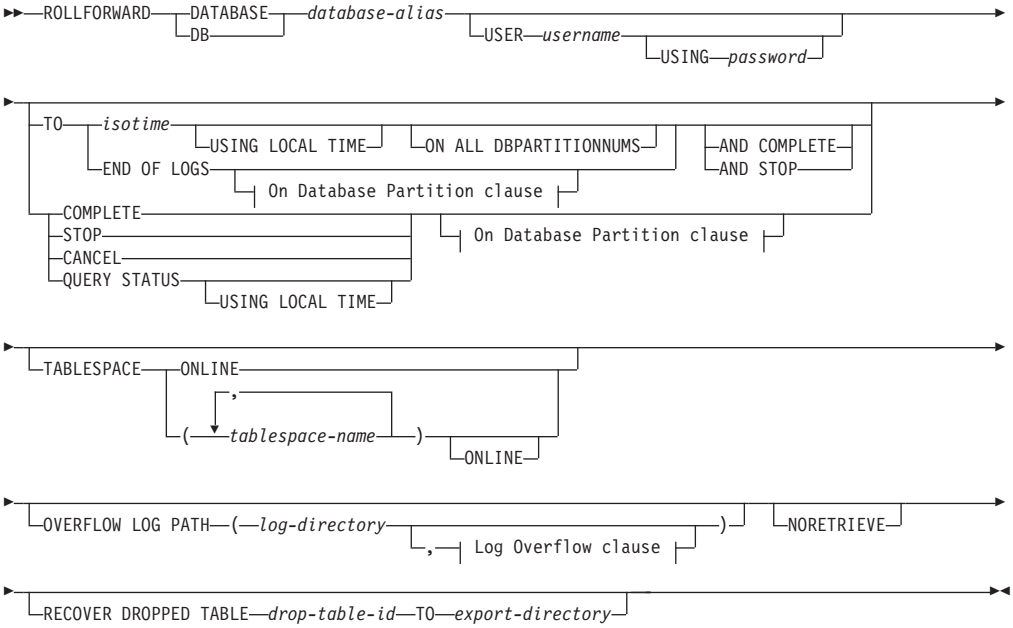
- *sysadm*

- *sysctrl*
- *sysmaint*

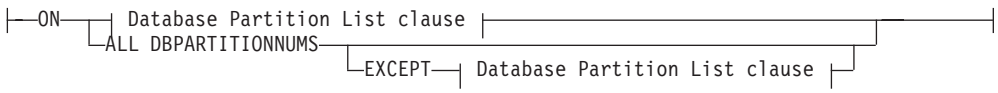
Required connection:

None. This command establishes a database connection.

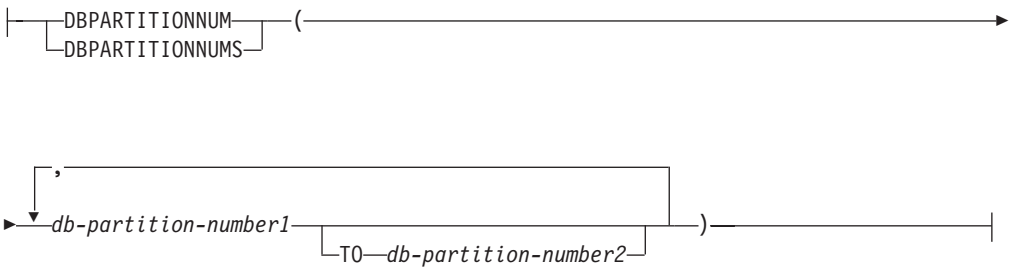
Command syntax:



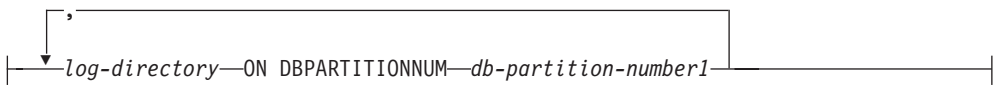
On Database Partition clause:



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, the user is 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.nnnnnn* (year, month, day, hour, minutes, seconds, microseconds), expressed in Coordinated Universal Time (UTC). 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 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.

Notes:

1. If the user specifies a local time for rollforward, all messages returned to the user will also be in local time. Note that all times are converted on the server, and in partitioned database environments, on the catalog database partition.
2. 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.
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.

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.

ALL DBPARTITIONNUMS
Specifies that transactions are to be rolled forward on all 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 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 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`.

Note: 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 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.

Note: 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 CUT) of the last committed transaction since rollforward processing began. In a partitioned database environment, this status information is returned for each 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.

Last committed transaction
A string containing a time stamp in ISO format (*yyyy-mm-dd-hh.mm.ss*). 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.

Note: 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 partition that is rolling forward. If it *does* exist, it must be in the correct state.

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 partitions*. A relative overflow log path can be specified for single-partition databases.

Note: 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 partition.

NORETRIEVE
Allows the user to control which log files to be rolled forward on the standby machine by allowing the user to disable the retrieval of archived logs. The benefits of this are:

- By controlling the logfiles 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.
- 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.

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.

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 1998-04-03-14.21.56.245378 and stop
   tablespace(TBS2, TBS3) online
```

Note that two rollforward operations cannot be run concurrently. The second command can only be invoked after the first rollforward operation completes successfully.

Example 4

After restoring the database, roll forward to a point in time, using OVERFLOW LOG PATH to specify the directory where the user exit saves archived logs:

```
db2 rollforward db sample to 1998-04-03-14.21.56.245378 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 partitions, and table space TBS2 is defined on 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 1998-04-03-14.21.56.245378 and stop
   tablespace(TBS1)
```

This fails, because TBS1 is not ready for rollforward recovery on database partition 1; all pieces must be rolled forward together.

Note: With table space rollforward to a point in time, the 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 1998-04-03-14.21.56.245378 and stop
   tablespace(TBS1)
```

This completes successfully.

Example 7 (partitioned database environment)

After restoring a table space on all database partitions, roll forward to PIT2, but do not specify AND STOP. The rollforward operation is still in progress. Cancel and roll forward to PIT1:

```
db2 rollforward db sample to pit2 tablespace(TBS1)
db2 rollforward db sample cancel tablespace(TBS1)

** restore TBS1 on all 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.

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, 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 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.

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.

Related reference:

- "BACKUP DATABASE" on page 227
- "RESTORE DATABASE" on page 354

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.

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.

Related concepts:

- "High availability through online split mirror and suspended I/O support" in the *Data Recovery and High Availability Guide and Reference*

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:

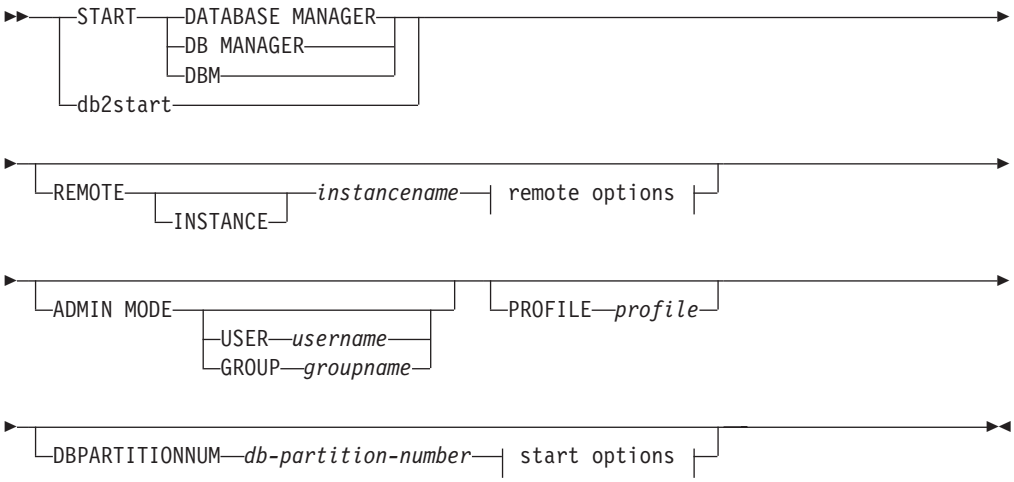
- *sysadm*
- *sysctrl*
- *sysmaint*

Note: The ADD DBPARTITIONNUM start option requires either *sysadm* or *sysctrl* authority.

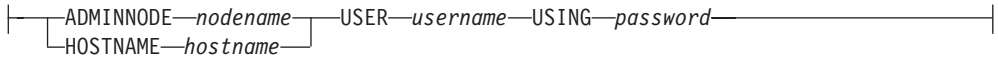
Required connection:

None

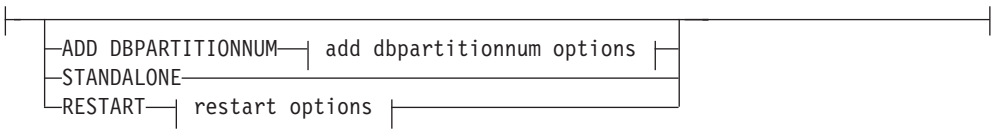
Command syntax:



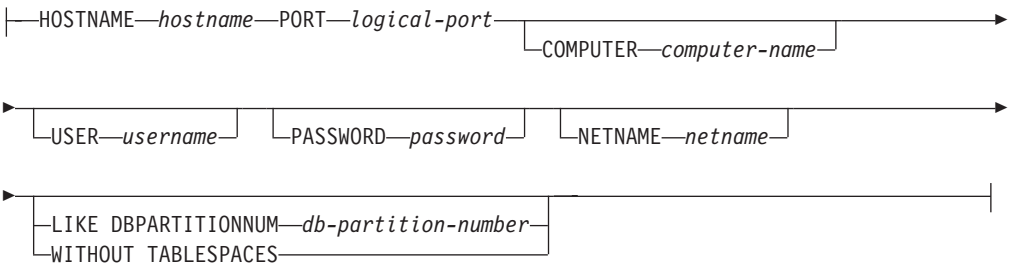
remote options:



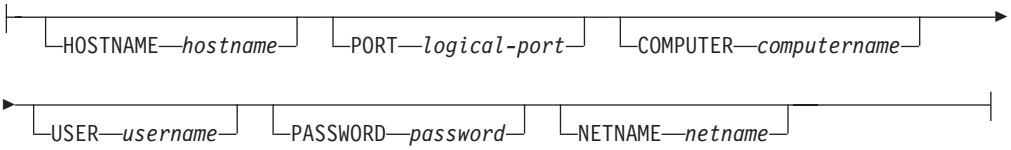
start options:



add dbpartitionnum options:



restart options:



Command parameters:

REMOTE [INSTANCE] instancename
Specifies the name of the remote instance you wish 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 the 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.

Note: 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.

Note: 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 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 utility is executed internally to create all existing databases on the database partition being added. After a database partition is added, the `db2nodes.cfg` file is not updated with the new database partition until a **db2stop** is issued. The database partition is not part of the MPP system until the next **db2start** following the **db2stop**.

Note: When the database partitions are created on the new node, their configuration parameters are set to the default.

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 NT, 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 NT, 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 NT, 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*.

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.

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.

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
With RESTART, specifies the host name to be used to override that in the database partition configuration file.

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 NT, 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 NT, 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 NT, 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:

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
```

On UNIX platforms, the START DATABASE MANAGER command supports the SIGINT and SIGALRM signals. The SIGINT signal is issued if CTRL+C is pressed. The SIGALRM signal is issued if the value specified for the *start_stop_time* database manager configuration parameter is reached. If either signal occurs, all in-progress startups are interrupted and a message (SQL1044N for SIGINT and SQL6037N for SIGALRM) 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 the Windows NT operating system, neither the `db2start` command nor the `NET START` command returns warnings if any communication subsystem failed to start. The database manager in a Windows NT environment is implemented as an NT

service, and does not return an error if the service is started successfully. Be sure to examine the NT 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.

Related reference:

- "STOP DATABASE MANAGER" on page 378
- "ADD DBPARTITIONNUM Command" in the *Command Reference*

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.

On partitioned database system, 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 systems 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:

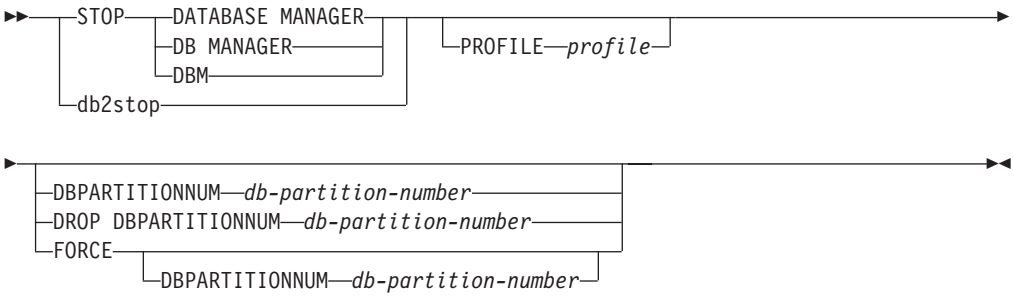
One of the following:

- *sysadm*
- *sysctrl*
- *sysmaint*

Required connection:

None

Command syntax:



Command parameters:

PROFILE profile
partitioned database systems 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 systems 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 systems 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 systems 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:

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 system since the previous STOP DATABASE MANAGER command was issued will be updated in the `db2nodes.cfg` file.
- On UNIX platforms, this command supports the SIGALRM signal, which is issued if the value specified for the *start_stop_time* database manager configuration parameter is reached. If this signal occurs, 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.

Related reference:

- "FORCE APPLICATION Command" in the *Command Reference*
- "START DATABASE MANAGER" on page 373
- "DEACTIVATE DATABASE Command" in the *Command Reference*
- "DROP DBPARTITIONNUM VERIFY Command" in the *Command Reference*

UNQUIESCE

Restores user access to instances or databases which have been quiesced for maintenance or other reasons. UNQUIESCE restores user access without necessitating a shutdown and database restart.

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.

UNQUIESCE INSTANCE *instance-name* restores user access to the instance and the databases in the instance *instance-name*.

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:

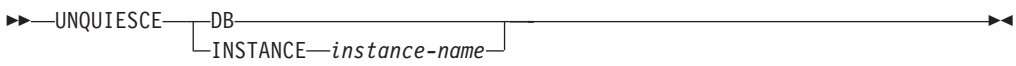
For database level unquiesce:

- *sysadm*
- *dbadm*

For instance level unquiesce:

- *sysadm*
- *sysctrl*

Command syntax:



Required connection:
Database

(Database connection is not required for an instance unquiesce.)

Command parameters:

DB Unquiesce the database. User access will be restored to all objects in the database.

INSTANCE *instance-name*
Access is restored to the instance *instance-name* and the databases in the instance.

Examples:

Unquiescing a Database

```
db2 unquiesce db
```

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 node on which the database has been created.

Scope:

This command only affects the node on which it is executed.

Authorization:

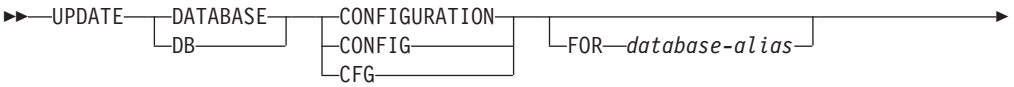
One of the following:

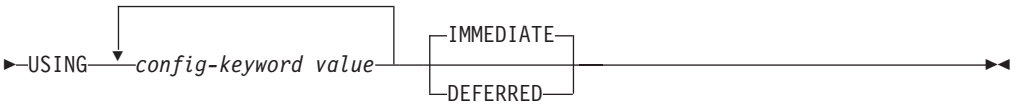
- *sysadm*
- *sysctrl*
- *sysmaint*

Required connection:

Instance. An explicit attachment is not required, but a database connection is recommended when the database is active. If the database is listed as remote, an instance attachment to the remote node is established for the duration of the command. To change a parameter online, you must be connected to the database.

Command syntax:





Command parameters:

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.

IMMEDIATE
Make the changes immediately, while the database is running. IMMEDIATE is the default action, but it requires a database connection to be effective.

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:

To view or print a list of the database configuration parameters, use the GET DATABASE CONFIGURATION command.

To reset all the database configuration parameters to the recommended defaults, use the RESET DATABASE CONFIGURATION command.

To change a database configuration parameter, use the UPDATE DATABASE CONFIGURATION command. For example, to change the logging mode to "archival logging" on a single-partition database environment containing a database called ZELLMART, use:

```
db2 update db cfg for zellmart using logretain recovery
```

To check that the *logretain* configuration parameter has changed, use:

```
db2 get db cfg for zellmart
```

When changing configuration parameters in a multiple-partitioned database environment, the `db2_all` command should be used. Using the `db2_all` command results in the update being issued against all partitions.

For example, to change the logging mode to "archival logging" in a multiple-partitioned database environment containing a database called "zellmart", use:

```
db2_all ";db2 update db cfg for zellmart using logretain recovery"
```

To check that the *logretain* configuration parameter has changed on all database partitions, use:

```
db2_all ";db2 get db cfg for zellmart"
```

If you are working on a UNIX operating system, and you have the "grep" command, you can use the following command to view only the *logretain* values:

```
db2_all ";db2 get db cfg for zellmart | grep -i logretain"
```

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.

For example, to change the *sortheap* database configuration parameter online for the SALES database, enter the following commands:

```
db2 connect to sales
db2 update db cfg using sortheap 1000
db2 connect reset
```

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.

Related concepts:

- "rah and db2_all commands overview" in the *Administration Guide: Implementation*

Related tasks:

- "Configuring DB2 with configuration parameters" on page 779

Related reference:

- "GET DATABASE CONFIGURATION" on page 275
- "RESET DATABASE CONFIGURATION Command" in the *Command Reference*
- "Configuration parameters summary" in the *Administration Guide: Performance*

UPDATE DATABASE MANAGER CONFIGURATION

Modifies individual entries in the database manager configuration file.

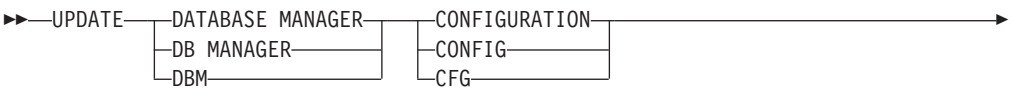
Authorization:

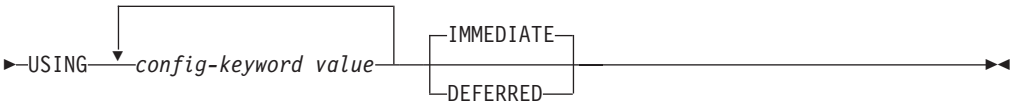
sysadm

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 update the database manager configuration for a remote instance, it is necessary to first attach to that instance. To update a configuration parameter on-line, it is also necessary to first attach to the instance.

Command syntax:





Command parameters:

DEFERRED
Make the changes only in the configuration file, so that the changes take effect when the instance is restarted.

IMMEDIATE
Make the changes right now, dynamically, while the instance is running. IMMEDIATE is the default, but it requires an instance attachment to be effective.

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.

Usage notes:

To view or print a list of the database manager configuration parameters, use the GET DATABASE MANAGER CONFIGURATION command. To reset the database manager configuration parameters to the recommended database manager defaults,

use the RESET DATABASE MANAGER CONFIGURATION command. For more information about database manager configuration parameters and the values of these parameters appropriate for each type of database node configured (server, client, or server with remote clients), see individual configuration parameter descriptions.

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 on-line 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.

For example, to change the *DIAGLEVEL* database manager configuration parameter on-line for the **eastern** instance of the database manager, enter the following command:

```
db2 attach to eastern
db2 update dbm cfg using DIAGLEVEL 1
db2 detach
```

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, NNAME, 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.

Related tasks:

- "Configuring DB2 with configuration parameters" on page 779

Related reference:

- "GET DATABASE MANAGER CONFIGURATION" on page 281
- "RESET DATABASE MANAGER CONFIGURATION Command" in the *Command Reference*
- "TERMINATE Command" in the *Command Reference*
- "Configuration parameters summary" in the *Administration Guide: Performance*

Chapter 14. DB2 UDB APIs for Administrators

Following are the application programming interfaces (APIs) that correspond to the DB2 UDB commands that are used for the Common Criteria evaluation. Note that:

- APIs are not used for the Common Criteria certification. The APIs are included in this document for reasons of completeness only.
- Not every API has a corresponding command, and vice versa.

db2Backup - Backup database

Creates a backup copy of a database or a table space.

Scope:

This API only affects the database partition on which it is executed.

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

C API syntax:

```
/* File: db2ApiDf.h */
/* API: db2Backup */
/* ... */
SQL_API_RC SQL_API_FN
```

© Copyright IBM Corp. 1993-2004

```
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;
} db2BackupStruct;

typedef SQL_STRUCTURE db2TablespaceStruct
{
  char                         **tablespaces;
  db2Uint32                    numTablespaces;
} db2TablespaceStruct;

typedef SQL_STRUCTURE db2MediaListStruct
{
  char                         **locations;
  db2Uint32                    numLocations;
  char                         locationType;
} db2MediaListStruct;
/* ... */
```

Generic API syntax:

```
/* File: db2ApiDf.h */
/* API: db2Backup */
/* ... */
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;
} 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 db2Char
{
   char            *pioData;
   db2Uint32       iLength;
   db2Uint32       oLength;
} db2Char;
/* ... */
```

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.

piDBAlias
Input. A string containing the database alias (as cataloged in the system database directory) of the database to back up.

iDBAliasLen
Input. A 4-byte unsigned integer representing the length in bytes of the database alias.

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.

poApplicationId
Output. Supply a buffer of length SQLU_APPLID_LEN+1 (defined in `sqlutil.h`). 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.

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`).

oTimestamp
Output. The API will return the time stamp of the backup image

poTimestamp
Output. Supply a buffer of length SQLU_TIME_STAMP_LEN+1 (defined in `sqlutil.h`). The API will return the time stamp of the backup image.

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`).

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. The information provided depends on the value of the *locationType* parameter. The valid values for *locationType* (defined in `sqlutil.h`) are:

SQLU_LOCAL_MEDIA
Local devices (a combination of tapes, disks, or diskettes).

SQLU_TSM_MEDIA
TSM. If the locations pointer is set to NULL, the TSM shared library provided with DB2 is used. If a different version of the TSM shared library is desired, use SQLU_OTHER_MEDIA and provide the shared library name.

SQLU_OTHER_MEDIA
Vendor product. Provide the shared library name in the locations field.

SQLU_USER_EXIT
User exit. No additional input is required (only available when server is on OS/2).

For more information, see the *db2MediaListStruct* structure .

piUsername
Input. A string containing the user name to be used when attempting a connection. Can be NULL.

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.

piPassword
Input. A string containing the password to be used with the user name. Can be NULL.

iPasswordLen
Input. A 4-byte unsigned integer representing the length in bytes of the password. Set to zero if no password is provided.

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.h`) 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.h`) 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.

iUtilImpactPriority
Specifies the priority value that will be used during a backup. The priority value can be any number between 0 and 100, with 0 representing unthrottled and 100 representing the highest priority.

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.

piComprLibraryLen
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.

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.

tablespaces
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
Number of entries in the *tablespaces* parameter.

locations
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
The number of entries in the *locations* parameter.

locationType
A character indicated the media type. Valid values (defined in `sqlutil.h`.) are:

SQLU_LOCAL_MEDIA
Local devices (tapes, disks, diskettes, or named pipes).

SQLU_TSM_MEDIA
Tivoli Storage Manager.

SQLU_OTHER_MEDIA
Vendor library.

SQLU_USER_EXIT
User exit (only available when the server is on OS/2).

pioData
A pointer to the character data buffer.

iLength
Input. The size of the *pioData* buffer.

oLength
Output. Reserved for future use.

Related reference:

- "sqlemgdb - Migrate Database" on page 508
- "db2Rollforward - Rollforward Database" on page 474
- "SQLCA" in the *Administrative API Reference*
- "db2Restore - Restore database" on page 463

Related samples:

- "dbrecov.sqc -- How to recover a database (C)"
- "dbrecov.sqC -- How to recover a database (C++)"

db2CfgGet - Get 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

C API syntax:

```
/* File: db2ApiDf.h */
/* API: db2CfgGet */
/* ... */
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;
/* ... */
```

Generic API syntax:

```
/* File: db2ApiDf.h */
/* API db2gCfgGet */
/* ... */
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;
/* ... */
```

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.

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 (db2Cfg structure)
Input. Specifies the type of action to be taken. Valid values (defined in `db2ApiDf.h`) 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.

dbname_len
Input. The length in bytes of *dbname*.

dbname
Input. The database name.

token Input. The configuration parameter identifier.

Valid entries and data types for the db2CfgParam *token* element are listed in Configuration parameters summary.

ptrvalue_len
Input. The length in bytes of *ptrvalue*.

ptrvalue
Output. The configuration parameter value.

flags (db2CfgParam structure)
Input. Provides specific information for each parameter in a request. Valid values (defined in `db2ApiDf.h`) 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.

Related concepts:

- "Configuration parameter tuning" in the *Administration Guide: Performance*

Related tasks:

- "Configuring DB2 with configuration parameters" on page 779

Related reference:

- "SQLCA" in the *Administrative API Reference*
- "Configuration parameters summary" in the *Administration Guide: Performance*
- "db2CfgSet - Set Configuration Parameters" on page 397

Related samples:

- "dbinfo.c -- Set and get information at the database level (C)"
- "dbrecov.sqc -- How to recover a database (C)"
- "inauth.sqc -- How to display authorities at instance level (C)"
- "ininfo.c -- Set and get information at the instance level (C)"
- "tscreate.sqc -- How to create and drop buffer pools and table spaces (C)"
- "dbinfo.C -- Set and get information at the database level (C++)"
- "dbrecov.sqC -- How to recover a database (C++)"
- "inauth.sqC -- How to display authorities at instance level (C++)"
- "ininfo.C -- Set and get information at the instance level (C++)"
- "tscreate.sqC -- How to create and drop buffer pools and table spaces (C++)"

db2CfgSet - Set 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 the node on which it is executed.

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 instance attachment is required. Otherwise a connection to a database or an attachment to an instance is not required.

API include file:

db2ApiDf.h

C API syntax:

```
/* File: db2ApiDf.h */
/* API: db2CfgSet */
/* ... */
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;
} db2Cfg;

typedef SQL_STRUCTURE db2CfgParam
{
   db2Uint32                            token;
   char                                 *ptrvalue;
   db2Uint32                            flags;
} db2CfgParam;
/* ... */
```

Generic API syntax:

```
/* File: db2ApiDf.h */
/* API db2gCfgGet */
/* ... */
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;
/* ... */
```

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.

numItems
Input. The number of configuration parameters in the *paramArray* array.

paramArray
Input. A pointer to the *db2CfgParam* structure.

flags (db2Cfg structure)
Input. Specifies the type of action to be taken. Valid values (defined in `db2ApiDf.h`) 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
Sets the current values of the configuration parameters in memory.

db2CfgDelayed
Sets 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.

db2CfgReset
Resets the configuration parameters to the default values.

dbname_len
Input. The length in bytes of *dbname*.

dbname
Input. The database name.

token Input. The configuration parameter identifier.

Valid entries and data types for the db2CfgParam *token* element are listed in Configuration parameters summary.

ptrvalue_len
Input. The length in bytes of *ptrvalue*.

ptrvalue
Input. The configuration parameter value.

flags (db2CfgParam structure)
Input. Specifies the type of action to be taken for each parameter in a request. By default, this field should be set to zero. Valid values (defined in `db2ApiDf.h`) are:

db2CfgParamAutomatic
Sets the configuration parameter value to *automatic*. DB2 will automatically adjust this parameter to reflect the current resource requirements. Only parameters that support the automatic behavior can be set to *automatic*.

Related concepts:

- "Configuration parameters" on page 769

Related tasks:

- "Configuring DB2 with configuration parameters" on page 779

Related reference:

- "SQLCA" in the *Administrative API Reference*
- "Configuration parameters summary" in the *Administration Guide: Performance*
- "db2CfgGet - Get Configuration Parameters" on page 394

Related samples:

- "dbinfo.c -- Set and get information at the database level (C)"
- "dbrecov.sqc -- How to recover a database (C)"
- "ininfo.c -- Set and get information at the instance level (C)"
- "dbinfo.C -- Set and get information at the database level (C++)"
- "dbrecov.sqC -- How to recover a database (C++)"
- "ininfo.C -- Set and get information at the instance level (C++)"

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

C API syntax:

```
/* File: db2ApiDf.h */
/* API: db2DatabaseRestart */
/* ... */
SQL_API_RC SQL_API_FN
db2DatabaseRestart (
  db2Uint32 versionNumber;
  void *pParamStruct;
  struct sqlca *pSqlca);

typedef struct
{
  char *piDatabaseName;
  char *piUserId;
  char *piPassword;
  char *piTablespaceNames;
  int  *iOption;

} db2RestartDbStruct;
/* ... */
```

Generic API syntax:

```
/* File: db2ApiDf.h */
/* API: db2DatabaseRestart */
/* ... */
SQL_API_RC SQL_API_FN
  db2DatabaseRestart (
```

```
    db2Uint32 versionNumber;
    void *pParamStruct;
    struct sqlca *pSqlca);

typedef struct
{
  char *piDatabaseName;
  char *piUserId;
  char *piPassword;
  char *piTablespaceNames;
  int  *iOption;

} db2RestartDbStruct;
/* ... */
```

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.

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.

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.

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.

Related reference:

- "SQLCA" in the *Administrative API Reference*

Related samples:

- "dbconn.sqc -- How to connect to and disconnect from a database (C)"
- "dbconn.sqC -- How to connect to and disconnect from a database (C++)"

db2DatabaseQuiesce - Database Quiesce

Forces all users off the database, immediately rolls back all active transactions, 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. 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

C API syntax:

```
/* File: db2ApiDf.h */
/* API: db2DatabaseQuiesce */
/* ... */
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;
/* ... */
```

Generic API syntax:

```
/* File: db2ApiDf.h */
/* API: db2gDatabaseQuiesce */
/* ... */
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;
/* ... */
```

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.

iDatabaseNameLen
Input. Specifies the length in bytes of *piDatabaseName*.

piDatabaseName
Input. The database name.

iImmediate
Input. Reserved for future use.

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_timeout* database manager configuration parameter will be used.

Related reference:

- "SQLCA" in the *Administrative API Reference*
- "db2DatabaseUnquiesce - Database Unquiesce" on page 404

db2DatabaseUnquiesce - Database Unquiesce

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

C API syntax:

```
/* File: db2ApiDf.h */
/* API: db2DatabaseUnquiesce */
/* ... */
SQL_API_RC SQL_API_FN
db2DatabaseUnquiesce (
  db2Uint32 versionNumber,
  void *pParmStruct,
  struct sqlca *pSqlca);

typedef SQL_STRUCTURE db2DbUnquiesceStruct
{
             char          *piDatabaseName;
} db2DbUnquiesceStruct;
/* ... */
```

Generic API syntax:

```
/* File: db2ApiDf.h */
/* API: db2gDatabaseunquiesce */
/* ... */
SQL_API_RC SQL_API_FN
db2gDatabaseUnquiesce (
  db2Uint32 versionNumber,
  void *pParmStruct,
  struct sqlca *pSqlca);

typedef SQL_STRUCTURE db2gDbUnquiesceStruct
```

```
{
            db2Uint32     iDatabaseNameLen;
            char          *piDatabaseName;
} db2gDbUnquiesceStruct;
/* ... */
```

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.

iDatabaseNameLen
Input. Specifies the length in bytes of *piDatabaseName*.

piDatabaseName
Input. The database name.

Related reference:

- "SQLCA" in the *Administrative API Reference*
- "db2DatabaseQuiesce - Database Quiesce" on page 402

db2Export - 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.

Authorization:

One of the following:

- *sysadm*
- *dbadm*

or CONTROL or SELECT privilege on each participating table or view.

Required connection:

Database. If implicit connect is enabled, a connection to the default database is established.

API include file:

db2ApiDf.h

C API syntax:

```
/* File: db2ApiDf.h */
/* API: db2Export */
/* ... */

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;
} db2ExportStruct;

typedef SQL_STRUCTURE db2ExportOut
{
   db2Uint64                           oRowsExported;
} db2ExportOut;
/* ... */
```

Generic API syntax:

```
/* File: db2ApiDf.h */
/* API: db2gExport */
/* ... */

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;
} db2gExportStruct;
/* ... */
```

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.

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.

piDataFileName
Input. A string containing the path and the name of the external file into which the data is to be exported.

piLobPathList
Input. An *sqlu_media_list* using *media_type* `SQLU_LOCAL_MEDIA`, and the *sqlu_media_entry* structure listing paths on the client where the LOB files are to be stored.

When file space is exhausted on the first path in this list, the API will use the second path, and so on.

piLobFileList
Input. An *sqlu_media_list* using *media_type* `SQLU_CLIENT_LOCATION`, and the *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 *pLobFilePath*), and then appending a 3-digit sequence number. For example, if the current LOB path is the directory `/u/foo/lob/path`, and the current LOB file name is `bar`, the created LOB files will be `/u/foo/lob/path/bar.001`, `/u/foo/lob/pah/bar.002`, and so on.

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`) 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 *pActionString*.

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`) 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 for exchange with Lotus Symphony and 1-2-3 programs.

SQL_IXF
PC version of the Integrated 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 File type modifiers for export.

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 overwritten. If it does not exist, a file is created.

iCallerAction
Input. An action requested by the caller. Valid values (defined in `sqlutil`) 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.

oRowsExported
Output. Returns the number of records exported to the target file.

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

IXF PC version of Integrated 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.

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.

A warning message is issued if the number of columns (*dcolnum*) 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.

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.

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 iSeries. Only PC/IXF export is supported.

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 *pActionString* 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* 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.

DB2 Data Links Manager considerations:

To ensure that a consistent copy of the table and the corresponding files referenced by the DATALINK columns are copied for export, do the following:

1. Issue the command: QUIESCE TABLESPACES FOR TABLE tablename SHARE.
   This ensures that no update transactions are in progress when EXPORT is run.
2. Issue the EXPORT command.
3. Run the **dlfm_export** utility at each Data Links server. Input to the **dlfm_export** utility is the control file name, which is generated by the export utility. This produces a tar (or equivalent) archive of the files listed within the control file. **dlfm_export** does not capture the ACLs information of the files that are archived.
4. Issue the command: QUIESCE TABLESPACES FOR TABLE tablename RESET.
   This makes the table available for updates.

EXPORT is executed as an SQL application. The rows and columns satisfying the SELECT statement conditions are extracted from the database. For the DATALINK columns, the SELECT statement should not specify any scalar function.

Successful execution of EXPORT results in generation of the following files:

- An export data file as specified in the EXPORT command. A DATALINK column value in this file has the same format as that used by the IMPORT and LOAD utilities. When the DATALINK column value is the SQL NULL value, handling is the same as that for other data types.
- Control files *server_name*, which are generated for each Data Links server. On the Windows NT operating system, a single control file, `ctrlfile.lst`, is used by all Data Links servers. These control files are placed in the directory <data-file path>/dlfm/YYYYMMDD/HHMMSS (on the Windows NT operating system, `ctrlfile.lst` is placed in the directory <data-file path>\dlfm\YYYYMMDD\HHMMSS). YYYYMMDD represents the date (year month day), and HHMMSS represents the time (hour minute second).

The **dlfm_export** utility is provided to export files from a Data Links server. This utility generates an archive file, which can be used to restore files in the target Data Links server.

Related concepts:

- "Moving DB2 Data Links Manager Data Using Export - Concepts" in the *Data Movement Utilities Guide and Reference*

Related reference:

- "SQLCA" in the *Administrative API Reference*
- "SQLCHAR" in the *Administrative API Reference*
- "SQLDCOL" in the *Administrative API Reference*
- "SQLU-MEDIA-LIST" in the *Administrative API Reference*
- "File type modifiers for export" in the *Command Reference*
- "Delimiter restrictions for moving data" in the *Command Reference*

Related samples:

- "expsamp.sqb -- Export and import tables with table data to a DRDA database (IBM COBOL)"
- "impexp.sqb -- Export and import tables with table data (IBM COBOL)"
- "tload.sqb -- How to export and load table data (IBM COBOL)"
- "tbmove.sqc -- How to move table data (C)"
- "tbmove.sqC -- How to move table data (C++)"

db2Import - Import

Inserts data from an external file with a supported file format into a table, hierarchy, or view. A faster alternative is `Load` however, the load utility does not support loading data at the hierarchy level.

Authorization:

- IMPORT using the INSERT option requires one of the following:
  - *sysadm*
  - *dbadm*
  - CONTROL privilege on each participating table or view
  - 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:
  - *sysadm*
  - *dbadm*
  - CONTROL privilege on the table or view
  - 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:
  - *sysadm*
  - *dbadm*
  - 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:
  - *sysadm*
  - *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
    - CREATIN 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:
  - *sysadm*
  - *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:
  – *sysadm*
  – *dbadm*
  – 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

C API syntax:

```
/* db2Import - API */
SQL_API_RC SQL_API_FN
  db2Import (
   db2Uint32 versionNumber,
   void * pParmStruct,
   struct sqlca * pSqlca);

/* db2Import parameter structure */
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;
} db2ImportStruct;

/* Import input structure */
typedef SQL_STRUCTURE db2ImportIn
{
   db2Uint64                           iRowcount;
   db2Uint64                           iRestartcount;
   db2Uint64                           iSkipcount;
   db2int32                            *piCommitcount;
   db2Uint32                           iWarningcount;
   db2Uint16                           iNoTimeout;
   db2Uint16                           iAccessLevel;
} db2ImportIn;

/* Import output structure */
typedef SQL_STRUCTURE db2ImportOut
{
   db2Uint64                           oRowsRead;
   db2Uint64                           oRowsSkipped;
```

```
   db2Uint64                          oRowsInserted;
   db2Uint64                          oRowsUpdated;
   db2Uint64                          oRowsRejected;
   db2Uint64                          oRowsCommitted;
} db2ImportOut;
```

Generic API syntax:

```
/* db2gImport - Generic API */
SQL_API_RC SQL_API_FN
  db2gImport (
   db2Uint32 versionNumber,
   void * pParmStruct,
   struct sqlca * pSqlca);

/* db2gImport parameter structure */
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;
} db2gImportStruct;

/* Generic Import input structure */
typedef SQL_STRUCTURE db2gImportIn
{
   db2Uint64                          iRowcount;
   db2Uint64                          iRestartcount;
   db2Uint64                          iSkipcount;
   db2int32                           *piCommitcount;
   db2Uint32                          iWarningcount;
   db2Uint16                          iNoTimeout;
   db2Uint16                          iAccessLevel;
} db2gImportIn;

/* Generic Import output structure */
typedef SQL_STRUCTURE db2gImportOut
{
   db2Uint64                          oRowsRead;
   db2Uint64                          oRowsSkipped;
   db2Uint64                          oRowsInserted;
   db2Uint64                          oRowsUpdated;
   db2Uint64                          oRowsRejected;
   db2Uint64                          oRowsCommitted;
} db2gImportOut;
```

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.

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. An *sqlu_media_list* using *media_type* SQLU_LOCAL_MEDIA, and the *sqlu_media_entry* structure listing paths on the client where the LOB files can be found.

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
Input. Pointer to an *sqlchar* structure containing a 2-byte long field, followed by an array of characters identifying the columns into which data is to be imported.

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]}
[DATALINK SPECIFICATION datalink-spec]
```

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.

Attention: If an error occurs after the existing data is deleted, that data is lost.

CREATE
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.

REPLACE_CREATE
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.

Attention: If an error occurs after the existing data is deleted, that data is lost.

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 down-level server, 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.

DATALINK SPECIFICATION *datalink-spec*
Specifies parameters pertaining to DB2 Data Links Manager. These parameters can be specified using the same syntax as in the IMPORT command.

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).

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 Integrated 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.

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 File type modifiers for import.

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.

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.

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.

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.

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 *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.

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. Non-default values for *piDataDescriptor*, or specifying an explicit list of table columns in *piActionString*, 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, or a summary table.

Views cannot be created through the import utility.

On the Windows NT operating system:

- Importing logically split PC/IXF files is not supported.
- Importing bad format PC/IXF or WSF files is not supported.

DB2 Data Links Manager Considerations

Before running the DB2 import utility, do the following:

1. Copy the files that will be referenced to the appropriate Data Links servers. The `dlfm_import` utility can be used to extract files from an archive that is generated by the `dlfm_export` utility.
2. Register the required prefix names to the DB2 Data Links Managers. There may be other administrative tasks, such as registering the database, if required.
3. Update the Data Links server information in the URLs (of the DATALINK columns) from the exported data for the SQL table, if required. (If the original configuration's Data Links servers are the same at the target location, the Data Links server names need not be updated.)
4. Define the Data Links servers at the target configuration in the DB2 Data Links Manager configuration file.

When the import utility runs against the target database, files referred to by DATALINK column data are linked on the appropriate Data Links servers.

During the insert operation, DATALINK column processing links the files in the appropriate Data Links servers according to the column specifications at the target database.

Related reference:

- "SQLCA" in the *Administrative API Reference*
- "SQLDCOL" in the *Administrative API Reference*
- "SQLU-MEDIA-LIST" in the *Administrative API Reference*

- "File type modifiers for import" in the *Command Reference*
- "Delimiter restrictions for moving data" in the *Command Reference*

Related samples:

- "dtformat.sqc -- Load and import data format extensions (C)"
- "tbmove.sqc -- How to move table data (C)"
- "expsamp.sqb -- Export and import tables with table data to a DRDA database (IBM COBOL)"
- "impexp.sqb -- Export and import tables with table data (IBM COBOL)"
- "tbmove.sqC -- How to move table data (C++)"

db2Inspect - Inspect database

Inspects the database for architectural integrity and checks the pages of the database for page consistency.

Scope:

In a single partition database, the scope is the single partition only. In a partitioned database environment, it is the collection of all logical partitions defined in `db2nodes.cfg`.

Authorization:

One of the following:

- *sysadm*
- *sysctrl*
- *sysmaint*
- *dbadm*
- CONTROL privilege on the table

Required connection:

Database

API include file:

db2ApiDf.h

C API syntax:

```
/* File: db2ApiDf.h */
/* API: db2Inspect */
/* ... */
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                   iBeginCheckOption;
  db2int32                    iLimitErrorReported;
  db2Uint16                   iObjectErrorState;
  db2Uint16                   iKeepResultfile;
  db2Uint16                   iAllNodeFlag;
  db2Uint16                   iNumNodes;
  db2Uint16                   iLevelObjectData;
  db2Uint16                   iLevelObjectIndex;
  db2Uint16                   iLevelObjectLong;
  db2Uint16                   iLevelObjectLOB;
  db2Uint16                   iLevelObjectBlkMap;
  db2Uint16                   iLevelExtentMap;
} db2InspectStruct;
/* ... */
```

Generic API syntax:

```
/* File: db2ApiDf.h */
/* API: db2gInspect */
/* ... */
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                   iBeginCheckOption;
  db2int32                    iLimitErrorReported;
  db2Uint16                   iObjectErrorState;
  db2Uint16                   iKeepResultfile;
  db2Uint16                   iAllNodeFlag;
  db2Uint16                   iNumNodes;
  db2Uint16                   iLevelObjectData;
  db2Uint16                   iLevelObjectIndex;
  db2Uint16                   iLevelObjectLong;
  db2Uint16                   iLevelObjectLOB;
  db2Uint16                   iLevelObjectBlkMap;
  db2Uint16                   iLevelExtentMap;
} db2gInspectStruct;
/* ... */
```

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.

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 partition numbers on which to perform the operation.

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.

iAction
Input. Specifies the inspect action. Valid values are:

DB2INSPECT_ACT_CHECK_DB
Inspect the entire database.

DB2INSPECT_ACT_CHECK_TABSPACE
Inspect a table space.

DB2INSPECT_ACT_CHECK_TABLE
Inspect 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.

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.

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 are:

DB2INSPECT_LEVEL_NORMAL
Level is normal.

DB2INSPECT_LEVEL_LOW
Level is low.

DB2INSPECT_LEVEL_NONE
Level is none.

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 issuing a COMMIT or ROLLBACK 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.

The processing of table spaces will process only the objects that reside in that table space.

Related reference:
- "SQLCA" in the *Administrative API Reference*

db2InstanceStart - Instance Start

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

C API syntax:

```
/* File: db2ApiDf.h */
/* API: db2InstanceStart */
/* ... */
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;
/* ... */
```

Generic API syntax:

```
/* File: db2ApiDf.h */
/* API: db2gInstanceStart */
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;
/* ... */
```

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.

iIsRemote
Input. An indicator set to `TRUE` or `FALSE`. This parameter should be set to `TRUE` if this is a remote start.

iRemoteInstLen
Input. Specifies the length in bytes of *piRemoteInstName*.

piRemoteInstName
Input. The name of the remote instance.

piCommData
Input. A pointer to the *db2DasCommData* structure.

piStartOpts
Input. A pointer to the *db2StartOptionsStruct* structure.

iCommParam
Input. An indicator set to `TRUE` or `FALSE`. This parameter should be set to `TRUE` if this is a remote start.

iNodeOrHostNameLen
Input. Specifies the length in bytes of *piNodeOrHostName*.

piNodeOrHostName
Input. The database partition or hostname.

iUserIdLen
Input. Specifies the length in bytes of *piUserId*.

piUserId
Input. The user name.

iUserPwLen
Input. Specifies the length in bytes of *piUserPw*.

piUserPw
Input. The user password.

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.h`) are:

SQLE_NONE
Issue the normal db2start operation.

SQLE_ADDNODE
Issue the ADD NODE command.

SQLE_RESTART
Issue the RESTART DATABASE command.

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.

iIsQRequested
Input. An indicator set to `TRUE` or `FALSE`. This parameter should be set to `TRUE` if quiesce is requested.

iQUsrNameLen
Input. Specifies the length in bytes of *piQusrName*.

piQUsrName
Input. The quiesced username.

iQGrpNameLen
Input. Specifies the length in bytes of *piQGrpName.*

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.

Related reference:

- "SQLCA" in the *Administrative API Reference*
- "db2InstanceStop - Instance Stop" on page 433

Related samples:

- "instart.c -- Stop and start the current local instance (C)"
- "instart.C -- Stop and start the current local instance (C++)"

db2InstanceStop - Instance Stop

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

sqlenv.h

C API syntax:

```
/* File: db2ApiDf.h */
/* API: db2InstanceStop */
/* ... */
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;
/* ... */
```

Generic API syntax:

```
/* File: db2ApiDf.h */
/* API: db2gInstanceStop */
/* ... */
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;

typedef SQL_STRUCTURE db2StopOptionsStruct
{
            db2Uint32     iIsProfile;
            char          *piProfile;
            db2Uint32     iIsNodeNum;
            db2NodeType   iNodeNum;
            db2Uint32     iStopOption;
            db2Uint32     iCallerac;
} db2StopOptionsStruct;
/* ... */
```

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.

iIsRemote
Input. An indicator set to `TRUE` or `FALSE`. This parameter should be set to `TRUE` if this is a remote start.

iRemoteInstLen
Input. Specifies the length in bytes of *piRemoteInstName*.

piRemoteInstName
Input. The name of the remote instance.

piCommData
Input. A pointer to the *db2DasCommData* structure.

piStopOpts
Input. A pointer to the *db2StopOptionsStruct* structure.

iCommParam
Input. An indicator set to `TRUE` or `FALSE`. This parameter should be set to `TRUE` if this is a remote stop.

iNodeOrHostNameLen
Input. Specifies the length in bytes of *piNodeOrHostName*.

piNodeOrHostName
Input. The database partition or hostname.

iUserIdLen
Input. Specifies the length in bytes of *piUserId*.

piUserId
Input. The user name.

iUserPwLen
Input. Specifies the length in bytes of *piUserPw*.

piUserPw
Input. The user password.

iIsRemote
Input. An indicator set to TRUE or FALSE. This parameter should be set to TRUE if this is a remote stop.

iRemoteInstLen
Input. Specifies the length in bytes of *piRemoteInstName*.

piRemoteInstName
Input. The remote instance name.

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.

Related reference:

- "SQLCA" in the *Administrative API Reference*
- "db2InstanceStart - Instance Start" on page 428

Related samples:

- "instart.c -- Stop and start the current local instance (C)"
- "instart.C -- Stop and start the current local instance (C++)"

db2Load - Load

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. The load utility does not support loading data at the hierarchy level.

Authorization:

One of the following:

- *sysadm*
- *dbadm*
- 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.

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.

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

C API syntax:

```
/* File: db2ApiDf.h */
/* API: Load */
/* ... */
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;
} db2LoadStruct;

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;
} 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;
/* ... */
```

Generic API syntax:

```
/* File: db2ApiDf.h */
/* API: Load */
/* ... */
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;
} 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;
} db2gLoadIn;

typedef SQL_STRUCTURE db2LoadOut
{
   db2Uint64                          oRowsRead;
   db2Uint64                          oRowsSkipped;
   db2Uint64                          oRowsLoaded;
   db2Uint64                          oRowsRejected;
   db2Uint64                          oRowsDeleted;
   db2Uint64                          oRowsCommitted;
} db2LoadOut;

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;
   SQL_PDB_NODE_TYPE                  *piRunStatDBPartNum;
   db2Uint16                          iHostnameLen;
   db2Uint16                          iFileTransferLen;
   db2Uint16                          iPartFileLocLen;
   db2Uint16                          iMapFileInputLen;
   db2Uint16                          iMapFileOutputLen;
   db2Uint16                          iDistfileLen;
} db2gPartLoadIn;

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;
/* ... */
```

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.

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`) 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.

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`) 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 *pFileType* parameter is set to `SQL_ASC`, the *dcolmeth* field of this structure must either be set to `SQL_METH_L` or be set to `SQL_METH_D` and specifies a file name with POSITIONSFILE *pFileTypeMod* modifier which contains starting and ending pairs and null indicator positions. 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

Input. Pointer to an *sqlchar* structure, followed by an array of characters specifying an action that affects the table.

The character array is of the form:

```
"INSERT|REPLACE|RESTART|TERMINATE
INTO tbname [(column_list)]
[DATALINK SPECIFICATION datalink-spec]
[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 or a declared 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.

DATALINK SPECIFICATION *datalink-spec*
Specifies parameters pertaining to DB2 Data Links. These parameters can be specified using the same syntax as in the LOAD command.

FOR EXCEPTION *e_tbname*
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. DATALINK exceptions are also captured in 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 Integrated 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 of type SQLU_SQL_STMT, and refers to an actual SQL query and not a cursor declared against one.

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 File type modifiers for load.

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 field (defined in `sqlutil`) 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 *pDataDescriptor* 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
Input. 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`) 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.

iFileTypeLen
Input. Specifies the length in bytes of *iFileType*.

iLocalMsgFileLen
Input. Specifies the length in bytes of *iLocalMsgFileName*.

iTempFilesPathLen
Input. Specifies the length in bytes of *iTempFilesPath*.

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 tablespace *iUseTablespaceName* and copied over to the original tablespace 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 tablespace as the index object.

If the shadow copy is created in the same tablespace 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 tablespace 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 *savecnt* 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 spawn 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 spawn 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`) 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
Input. Specifies to put the table into check pending state. If SQLU_CHECK_PENDING_CASCADE_IMMEDIATE is specified, check pending state will be immediately cascaded to all dependent and descendent tables. If SQLU_CHECK_PENDING_CASCADE_DEFERRED is specified, the cascade of check pending state to dependent tables will be deferred until the target table is checked for integrity violations. SQLU_CHECK_PENDING_CASCADE_DEFERRED is the default if the option is not specified.

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.

iUseTablespaceLen
Input. The length in bytes of *piUseTablespace*.

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.

piHostname
Input. The hostname for the *iFileTransferCmd* parameter. If NULL, the hostname will default to "nohost".

piFileTransferCmd
Input. File transfer command parameter. If not required, it must be set to NULL. See the Data Movement Guide for a full description of this parameter.

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 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 partition for PARTITION_ONLY mode, or the location of the files to be read from each 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 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. Refer to the Load command in the Data Movement Guide and Reference for a description of how the default is determined.

piMode
Input. Specifies the load mode for partitioned databases. Valid values (defined in db2ApiDf) are:

DB2LOAD_PARTITION_AND_LOAD
Data is partitioned (perhaps in parallel) and loaded simultaneously on the corresponding database partitions.

DB2LOAD_PARTITION_ONLY
Data is partitioned (perhaps in parallel) and the output is written to files in a specified location on each loading partition. For file types other than SQL_CURSOR, the name of the output file on each partition will have the form `filename.xxx`, where `filename` is the name of the first input file specified by *piSourceList* and `xxx` is the partition number.For the SQL_CURSOR file type, the name of the output file on each partition will be determined by the *piPartFileLocation* parameter. Refer to the *piPartFileLocation* parameter for information about how to specify the location of the partition file on each partition.

Note: This mode cannot be used for a CLI LOAD.

DB2LOAD_LOAD_ONLY
Data is assumed to be already partitioned; the partition 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 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 3-digit partition number. For the SQL_CURSOR file type, the name of the input file on each partition will be determined by the *piPartFileLocation* parameter. Refer to the *piPartFileLocation* parameter for information about how to specify the location of the partition file on each 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 partitioned, but the data file does not contain a partition header. The partitioning 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 partition. Rows containing partition violations are placed in a dumpfile if the dumpfile file type modifier is specified. Otherwise, the rows are discarded. If partition violations exist on a particular loading partition, a single warning will be written to the load message file for that partition. The input file name on each 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 3-digit 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 partitioning 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 partitions. Valid values (defined in `db2ApiDf`) are:

DB2LOAD_SETUP_ERRS_ONLY
In this mode, errors that occur on a partition during setup, such as problems accessing a partition or problems accessing a table space or table on a partition, will cause the load operation to stop on the failing partitions but to continue on the remaining partitions. Errors that occur on a partition while data is being loaded will cause the entire operation to fail and rollback to the last point of consistency on each partition.

DB2LOAD_LOAD_ERRS_ONLY
In this mode, errors that occur on a partition during setup will cause the entire load operation to fail. When an error occurs while data is being loaded, the partitions with errors will be rolled back to their last point of consistency. The load operation will continue on the remaining partitions until a failure occurs or until all the data is loaded. On the 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 partitions the transaction will be aborted. Data on all of the partitions will remain invisible until a load restart operation is performed. This will make the newly loaded data visible on the partitions where the load operation completed and resume the load operation on 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, partition-level errors during setup or loading data cause processing to stop only on the affected partitions. As with the DB2LOAD_LOAD_ERRS_ONLY mode, when partition errors do occur while data is being loaded, the data on all partitions will remain invisible until a load restart operation is performed.

Note: This mode cannot be 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. Partition 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. Partition 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 partition distribution file. If a NULL is specified, the value defaults to "DISTFILE".

piOmitHeader
Input. Indicates that a partition map header should not be included in the 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 partition list.

iHostnameLen
Input. The length in bytes of *piHostname*.

iFileTransferLen
Input. The length in bytes of *piFileTransferCmd*.

iPartFileLocLen
Input. The length in bytes of *piPartFileLocation*.

iMapFileInputLen
Input. The length in bytes of *piMapFileInput*.

iMapFileOutputLen
Input. The length in bytes of *piMapFileOutput*.

iDistfileLen
Input. The length in bytes of *piDistfile*.

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.

iPortMin
Input. Lower port number.

iPortMax
Input. Higher port number.

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-partitioing 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 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 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 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*.

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 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 CHECK 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 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 partitions.

DB2LOADQUERY_LOADPENDING
Indicates that the load job aborted during processing but left the table on the partition in the LOAD PENDING state, indicating that the load job on that partition must be either terminated or restarted.

oNodeNum
Output. The number of the partition on which the agent executed.

oAgentType
Output. The agent type. Valid values (defined in `db2ApiDf`) are :

DB2LOAD_LOAD_AGENT

DB2LOAD_PARTITIONING_AGENT

DB2LOAD_PRE_PARTITIONING_AGENT

DB2LOAD_FILE_TRANSFER_AGENT

DB2LOAD_LOAD_TO_FILE_AGENT

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 check pending state. Summary tables that are defined with REFRESH IMMEDIATE, and that are dependent on tables being loaded, are also placed in check pending state. Issue the SET INTEGRITY statement to take the tables out of check 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.

DB2 Data Links Manager Considerations

For each DATALINK column, there can be one column specification within parentheses. Each column specification consists of one or more of DL_LINKTYPE, *prefix* and a DL_URL_SUFFIX specification. The *prefix* information can be either DL_URL_REPLACE_PREFIX, or the DL_URL_DEFAULT_PREFIX specification.

There can be as many DATALINK column specifications as the number of DATALINK columns defined in the table. The order of specifications follows the order of DATALINK columns as found within the insert-column list (if specified by INSERT INTO (insert-column, ...)), or within the table definition (if insert-column is not specified).

For example, if a table has columns C1, C2, C3, C4, and C5, and among them only columns C2 and C5 are of type DATALINK, and the insert-column list is (C1, C5, C3, C2), there should be two DATALINK column specifications. The first column specification will be for C5, and the second column specification will be for C2. If an insert-column list is not specified, the first column specification will be for C2, and the second column specification will be for C5.

If there are multiple DATALINK columns, and some columns do not need any particular specification, the column specification should have at least the parentheses to unambiguously identify the order of specifications. If there are no specifications for any of the columns, the entire list of empty parentheses can be dropped. Thus, in cases where the defaults are satisfactory, there need not be any DATALINK specification.

If data is being loaded into a table with a DATALINK column that is defined with FILE LINK CONTROL, perform the following steps before invoking the load utility. (If all the DATALINK columns are defined with NO LINK CONTROL, these steps are not necessary).

1. Ensure that the DB2 Data Links Manager is installed on the Data Links servers that will be referred to by the DATALINK column values.
2. Ensure that the database is registered with the DB2 Data Links Manager.
3. Copy to the appropriate Data Links servers, all files that will be inserted as DATALINK values.
4. Define the prefix name (or names) to the DB2 Data Links Managers on the Data Links servers.

5. Register the Data Links servers referred to by DATALINK data (to be loaded) in the DB2 Data Links Manager configuration file.

The connection between DB2 and the Data Links server may fail while running the load utility, causing the load operation to fail. If this occurs:

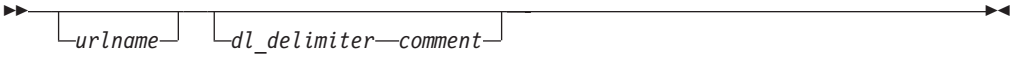
1. Start the Data Links server and the DB2 Data Links Manager.
2. Invoke a load restart operation.

Links that fail during the load operation are considered to be data integrity violations, and are handled in much the same way as unique index violations. Consequently, a special exception has been defined for loading tables that have one or more DATALINK columns.

Representation of DATALINK Information in an Input File

The LINKTYPE (currently only URL is supported) is not specified as part of DATALINK information. The LINKTYPE is specified in the LOAD or the IMPORT command, and for input files of type PC/IXF, in the appropriate column descriptor records.

The syntax of DATALINK information for a URL LINKTYPE is as follows:



Note that both *urlname* and *comment* are optional. If neither is provided, the NULL value is assigned.

urlname
The URL name must conform to valid URL syntax.

Notes:

1. Currently "http", "file", and "unc" are permitted as a schema name.
2. The prefix (schema, host, and port) of the URL name is optional. If a prefix is not present, it is taken from the DL_URL_DEFAULT_PREFIX or the DL_URL_REPLACE_PREFIX specification of the load or the import utility. If none of these is specified, the prefix defaults to "file://localhost". Thus, in the case of local files, the file name with full path name can be entered as the URL name, without the need for a DATALINK column specification within the LOAD or the IMPORT command.
3. Prefixes, even if present in URL names, are overridden by a different prefix name on the DL_URL_REPLACE_PREFIX specification during a load or import operation.
4. The "path" (after appending DL_URL_SUFFIX, if specified) is the full path name of the remote file in the remote server. Relative path names are not allowed. The http server default path-prefix is not taken into account.

dl_delimiter
For the delimited ASCII (DEL) file format, a character specified via the `dldel` modifier, or defaulted to on the LOAD or the IMPORT command. For the non-delimited ASCII (ASC) file format, this should correspond to the character sequence `\;` (a backslash followed by a semicolon).

Whitespace characters (blanks, tabs, and so on) are permitted before and after the value specified for this parameter.

comment
The comment portion of a DATALINK value. If specified for the delimited ASCII (DEL) file format, the *comment* text must be enclosed by the character string delimiter, which is double quotation marks (″) by default. This character string delimiter can be overridden by the MODIFIED BY *filetype-mod* specification of the LOAD or the IMPORT command.

If no comment is specified, the comment defaults to a string of length zero.

Following are DATALINK data examples for the delimited ASCII (DEL) file format:

- `http://www.almaden.ibm.com:80/mrep/intro.mpeg; "Intro Movie"`
  This is stored with the following parts:
  - scheme = http
  - server = www.almaden.ibm.com
  - path = /mrep/intro.mpeg
  - comment = ″Intro Movie″
- `file://narang/u/narang; "InderPal's Home Page"`
  This is stored with the following parts:
  - scheme = file
  - server = narang
  - path = /u/narang
  - comment = ″InderPal's Home Page″

Following are DATALINK data examples for the non-delimited ASCII (ASC) file format:

- `http://www.almaden.ibm.com:80/mrep/intro.mpeg\;Intro Movie`
  This is stored with the following parts:
  - scheme = http
  - server = www.almaden.ibm.com
  - path = /mrep/intro.mpeg
  - comment = ″Intro Movie″
- `file://narang/u/narang\; InderPal's Home Page`
  This is stored with the following parts:
  - scheme = file
  - server = narang
  - path = /u/narang
  - comment = ″InderPal's Home Page″

Following are DATALINK data examples in which the load or import specification for the column is assumed to be DL_URL_REPLACE_PREFIX (″http://qso″):

- `http://www.almaden.ibm.com/mrep/intro.mpeg`
  This is stored with the following parts:
  - schema = http
  - server = qso
  - path = /mrep/intro.mpeg
  - comment = NULL string

- `/u/me/myfile.ps`
  This is stored with the following parts:
  - schema = http
  - server = qso
  - path = /u/me/myfile.ps
  - comment = NULL string

Related reference:
- "sqluvqdp - Quiesce Table Spaces for Table" on page 514
- "db2LoadQuery - Load Query" in the *Administrative API Reference*
- "SQLDCOL" in the *Administrative API Reference*
- "SQLU-MEDIA-LIST" in the *Administrative API Reference*
- "db2Export - Export" on page 405
- "db2Import - Import" on page 412
- "db2DatabaseQuiesce - Database Quiesce" on page 402
- "db2InstanceQuiesce - Instance Quiesce" in the *Administrative API Reference*
- "File type modifiers for load" on page 326
- "Delimiter restrictions for moving data" in the *Command Reference*

Related samples:
- "dtformat.sqc -- Load and import data format extensions (C)"
- "tbload.sqc -- How to load into a partitioned database (C)"
- "tbmove.sqc -- How to move table data (C)"
- "tbmove.sqC -- How to move table data (C++)"

db2Reorg - Reorganize

Reorganize 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*
- CONTROL privilege on the table

Required connection:

Database

API include file:

db2ApiDf.h

C API syntax:

```
/* File: db2ApiDf.h */
/* API: db2Reorg */
/* ... */
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;
} db2ReorgTable;

typedef SQL_STRUCTURE db2ReorgIndexesAll
{
  char *pTableName;
} db2ReorgIndexesAll;
/* ... */
```

Generic API syntax:

```
/* File: db2ApiDf.h */
/* API: db2gReorg */
/* ... */
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;
} db2gReorgTable;

typedef SQL_STRUCTURE db2gReorgIndexesAll
{
  db2Uint32  tableNameLen;
  char       *pTableName;
} db2gReorgIndexesAll;
/* ... */
```

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.

reorgType
Input. Specifies the type of reorganization. Valid values (defined in `db2ApiDf.h`) are:

DB2REORG_OBJ_TABLE_OFFLINE
Reorganize the table offline.

DB2REORG_OBJ_TABLE_INPLACE
Reorganize the table inplace.

DB2REORG_OBJ_INDEXESALL
Reorganize all indexes.

reorgFlags
Input. Reorganization options. Valid values (defined in `db2ApiDf.h`) are:

DB2REORG_OPTION_NONE
Default action.

DB2REORG_LONGLOB
Reorganize long fields and lobs, used when DB2REORG_OBJ_TABLE_OFFLINE is specified as the *reorgType*.

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 is specified as the *reorgType*.

DB2REORG_CLEANUP_ALL
Clean up the indexes on a table by removing the committed pseudo deleted keys and committed pseudo empty pages, used when DB2REORG_OBJ_INDEXESALL is 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 is specified as the *reorgType*.

DB2REORG_CONVERT_NONE
No conversion is required, used when DB2REORG_OBJ_INDEXESALL is specified as the *reorgType*.

DB2REORG_CONVERT
Convert to type 2 index, used when DB2REORG_OBJ_INDEXESALL is specified as the *reorgType*.

nodeListFlag
Input. Specifies which nodes to reorganize. Valid values (defined in `db2ApiDf.h`) 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.

tableStruct
Specifies the options for a table reorganization.

indexesAllStruct
Specifies the options for an index reorganization.

tableNameLen
Input. Specifies the length in bytes of pTableName.

pTableName
Input. Specifies the name of the object to reorganize.

orderByIndexLen
Input. Specifies the length in byte of pOrderByIndex.

pOrderByIndex
Input. Specifies the index to order the table by.

sysTempSpaceLen
Input. Specifies the length in bytes of pSysTempSpace.

pSysTempSpace
Input. Specifies the system temporary table space to create temporary object in.

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. 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 is partitioned onto several nodes and the reorganization fails on any of the affected nodes, then only the failing nodes will have the table reorganization rolled back.

Note: If table 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 data and LONG table spaces must be roll-forward enabled.

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/DMS).

When reorganizing indexes, use the access option to allow other transactions either read only or read-write access to the table. There is a brief lock-out period when the reorganized indexes are being made available during which no access to the table is allowed.

If an index reorganization with allow read or allow write access fails because the indexes need to be rebuilt, the reorganization will be switched to allow no access and carry on. A message will be written to both the administration notification log and the diagnostics log to warn the user about the change in access mode.

If an index reorganization with no access fails, the indexes are not available and have to be rebuilt on the next table access.

This API cannot be used with:

- views or an index that is based on an index extension
- a DMS table while an online backup of a table space in which the table resides is being performed
- declared temporary tables

Related reference:

- "sqlarbnd - Rebind" on page 873
- "SQLCA" in the *Administrative API Reference*
- "db2Runstats - Runstats" in the *Administrative API Reference*

Related samples:

- "dbstat.sqb -- Reorganize table and run statistics (MF COBOL)"
- "tbreorg.sqc -- How to reorganize a table and update its statistics (C)"
- "tbreorg.sqC -- How to reorganize a table and update its statistics (C++)"

db2Restore - Restore database

Rebuilds a damaged or corrupted database that has been backed up using db2Backup - Backup Database. 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).

This utility can also be used to restore DB2 databases created in the two previous releases.

This utility can also 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.

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

C API syntax:

```
/* File: db2ApiDf.h */
/* API: db2Restore */
/* ... */
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;
} db2RestoreStruct;

typedef SQL_STRUCTURE db2TablespaceStruct
{
  char                        **tablespaces;
  db2Uint32                   numTablespaces;
} db2TablespaceStruct;

typedef SQL_STRUCTURE db2MediaListStruct
{
  char                        **locations;
  db2Uint32                   numLocations;
  char                        locationType;
} db2MediaListStruct;
/* ... */
```

Generic API syntax:

```
/* File: db2ApiDf.h */
/* API: db2gRestore */
/* ... */
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;
} 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 db2Char
{
   char            *pioData;
   db2Uint32       iLength;
   db2Uint32       oLength;
} db2Char;
/* ... */
```

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.

piSourceDBAlias
Input. A string containing the database alias of the source database backup image.

iSourceDBAliasLen
Input. A 4-byte unsigned integer representing the length in bytes of the source database alias.

piTargetDBAlias
Input. A string containing the target database alias. If this parameter is null, the *piSourceDBAlias* will be used.

iTargetDBAliasLen
Input. A 4-byte unsigned integer representing the length in bytes of the target database alias.

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.

poApplicationId
Output. Supply a buffer of length SQLU_APPLID_LEN+1 (defined in `sqlutil`). 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.

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`).

piTimestamp
Input. A string representing the timestamp of the backup image. This field is optional if there is only one backup image in the source specified.

iTimestampLen
Input. A 4-byte unsigned integer representing the length in bytes of the piTimestamp buffer.

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. The datalinks files that become unlinked during restore (as a result of a fast reconcile) will be reported.

iReportFileLen
Input. A 4-byte unsigned integer representing the length in bytes of the piReportFile buffer.

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. See the *DB2TablespaceStruct* structure. The following restrictions apply:

- The database must be recoverable; 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.

piMediaList
Input. Source media for the backup image. The information provided depends on the value of the locationType field. The valid values for locationType (defined in `sqlutil`) are:

SQLU_LOCAL_MEDIA
Local devices (a combination of tapes, disks, or diskettes).

SQLU_XBSA_MEDIA
XBSA interface. Backup Services APIs (XBSA) are an open application programming interface for applications or facilities needing data storage management for backup or archiving purposes.

SQLU_TSM_MEDIA
TSM. If the locations pointer is set to NULL, the TSM shared

library provided with DB2 is used. If a different version of the TSM shared library is desired, use SQLU_OTHER_MEDIA and provide the shared library name.

SQLU_OTHER_MEDIA
Vendor product. Provide the shared library name in the locations field.

SQLU_USER_EXIT
User exit. No additional input is required (only available when server is on OS/2).

piUsername
Input. A string containing the user name to be used when attempting a connection. Can be NULL.

iUsernameLen
Input. A 4-byte unsigned integer representing the length in bytes of piUsername. Set to zero if no user name is provided.

piPassword
Input. A string containing the password to be used with the user name. Can be NULL.

iPasswordLen
Input. A 4-byte unsigned integer representing the length in bytes of piPassword. Set to zero if no password is provided.

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.

iNewLogPathLen
Input. A 4-byte unsigned integer representing the length in bytes of *piNewLogPath*.

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 of the *piVendorOptions*, 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`) 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 successful return of this call, it is expected that the user will issue a call with DB2RESTORE_CONTINUE to proceed with the action.

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`) 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 restore type. 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
Specify to restore only the set of log files contained in the backup image. If the backup image did not include log files, the restore operation will fail. If this option is specified, the *piLogTarget* parameter must also be supplied.

DB2RESTORE_INCREMENTAL
Perform a manual cumulative restore operation.

DB2RESTORE_AUTOMATIC
Perform an automatic cumulative (incremental) restore operation. Must be specified with DB2RESTORE_INCREMENTAL.

DB2RESTORE_DATALINK
Perform reconciliation operations. Tables with a defined DATALINK column must have RECOVERY YES option specified.

DB2RESTORE_NODATALINK
Do not perform reconciliation operations. Tables with DATALINK columns are placed into DataLink_Roconcile_pending (DRP) state. Tables with a defined DATALINK column must have the RECOVERY YES option specified.

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, db2Rollforward - Rollforward Database must be executed before the database can be used.

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.

piComprLibraryLen
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.

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. 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.

piLogTarget
Input. The absolute path 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 parameter is specified, any log files included in the backup image will be extracted into the target directory. If this parameter is not specified, log files included in the backup image will not be extracted. To extract only the log files from the backup image, use the DB2RESTORE_LOGS parameter.

iLogTargetLen
Input. A four-byte unsigned integer representing the length, in bytes, of the path in *piLogTarget*.

tablespaces
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
Number of entries in the *tablespaces* parameter.

locations
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
The number of entries in the *locations* parameter.

locationType
A character indicated the media type. Valid values (defined in `sqlutil`) are:

SQLU_LOCAL_MEDIA
Local devices(tapes, disks, diskettes, or named pipes).

SQLU_XBSA_MEDIA
XBSA interface.

SQLU_TSM_MEDIA
Tivoli Storage Manager.

SQLU_OTHER_MEDIA
Vendor library.

SQLU_USER_EXIT
User exit (only available when the server is on OS/2).

pioData
A pointer to the character data buffer.

iLength
Input. The size of the *pioData* buffer

oLength
Output. Reserved for future use.

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. If the file is replaced, a warning message is returned.

The database or table space must have been backed up using db2Backup - Backup Database.

If the caller action is DB2RESTORE_NOINTERRUPT, the restore continues without prompting the application. If the caller action 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 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 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 SQLUD_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 to DB2RESTORE_PARM_CHK.

Set caller action to DB2RESTORE_RESTORE_STORDEF when performing a redirected restore; used in conjunction with sqlbstsc - Set Tablespace Containers.

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 on 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 useable 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 which contains them, the LOGTARGET option must be specified, providing a fully qualified and valid path which 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 which does not include logs, the restore will return an error before attempting to restore any table space data. A restore 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 command is issued, a warning prompt will be returned to user. This warning will not be returned if WITHOUT PROMPTING is specified.

During a restore where a LOGTARGET is specified, if any log file can not be extracted, for any reason, the restore 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 already in the LOGTARGET path, the restore operation will fail and an error will be returned. The restore utility will not overwrite existing log files in the LOGTARGET directory.

It is also possible to 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 will terminate immediately and an error will be returned.

During an automatic incremental restore, only the logs included in the target image of the restore operation will be retrived from the backup image. Any logs included in intermediate images referenced during the incremental restore process

will not be extracted from those intermediate backup images. During a manual incremental restore, the LOGTARGET path should only be specified with the final restore command to be issued.

If a backup is compressed, DB2 will detect this and automatically decompress the data before restoring it. If a library is specified on the db2Restore API, it will be used for decompressing the data. Otherwise, if a library that is stored in the backup image will be used. Otherwise, the data cannot be decompressed, so the restore will fail.

If the compression library is to be restored from a backup image (either explicitly by specifying the DB2RESTORE_COMPR_LIB restore type 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.

Related reference:

- "sqlemgdb - Migrate Database" on page 508
- "db2Rollforward - Rollforward Database" on page 474
- "SQLCA" in the *Administrative API Reference*
- "db2Backup - Backup database" on page 387
- "db2CfgGet - Get Configuration Parameters" on page 394

Related samples:

- "dbrecov.sqc -- How to recover a database (C)"
- "dbrecov.sqC -- How to recover a database (C++)"

db2Rollforward - Rollforward 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, the *logarchmeth1* database configuration parameter must be set to `on`) before the database can be recovered with rollforward recovery.

Scope:

In a partitioned database environment, this API can only be called from the catalog partition. A database or table space rollforward call specifying a point-in-time affects all database partition servers that are listed in the `db2nodes.cfg` file. A database or table space rollforward call specifying end of logs affects the database partition servers that are specified. If no database partition servers are specified, it affects all database partition servers that are listed in the `db2nodes.cfg` file; if no roll forward is needed on a particular database partition server, that database partition server is ignored.

Authorization:

One of the following:

- *sysadm*
- *sysctrl*

- *sysmaint*

Required connection:

None. This API establishes a database connection.

API include file:

db2ApiDf.h

C API syntax:

```
/* File: db2ApiDf.h */
/* API: db2Rollforward */
/* ... */
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;
} db2RfwdOutputStruct;

typedef SQL_STRUCTURE sqlurf_newlogpath
{
  SQL_PDB_NODE_TYPE           nodenum;
   unsigned short             pathlen;
   char                       logpath[SQL_LOGPATH_SZ+SQL_LOGFILE_NAME_SZ+1];
} sqlurf_newlogpath;

typedef SQL_STRUCTURE sqlu_tablespace_bkrst_list
{
  long                         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;

typedef 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];
} sqlurf_info;
/* ... */
```

Generic API syntax:

```
/* File: db2ApiDf.h */
/* API: db2Rollforward */
/* ... */
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;

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;
```

```
typedef SQL_STRUCTURE db2RfwdOutputStruct
{
  char                        *poApplicationId;
  sqlint32                    *poNumReplies;
  struct sqlurf_info          *poNodeInfo;
} db2RfwdOutputStruct;

typedef SQL_STRUCTURE sqlurf_newlogpath
{
  SQL_PDB_NODE_TYPE           nodenum;
   unsigned short             pathlen;
   char                       logpath[SQL_LOGPATH_SZ+SQL_LOGFILE_NAME_SZ+1];
} sqlurf_newlogpath;

typedef SQL_STRUCTURE sqlu_tablespace_bkrst_list
{
  long                         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;

typedef 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];
} sqlurf_info;
/* ... */
```

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.

piRfwdInput
Input. A pointer to the *db2RfwdInputStruct* structure.

poRfwdOutput
Output. A pointer to the *db2RfwdOutputStruct* structure.

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.

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.h) are:

DB2ROLLFORWARD_ROLLFWD
Rollforward to the point in time specified by *piStopTime*. 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. No new log records are processed and uncommitted transactions are backed out. The *rollforward-pending* state of the database or table spaces is turned off. Synonym 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. Synonym 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. May be NULL.

piPassword
Input. A string containing the password of the supplied user name (if any). May 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 user 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.h`) 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.

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 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 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.

piExportDir
Input. The directory into which the dropped table data will be exported.

RollforwardFlags
Input. Specifies the rollforward flags. Valid values (defined in `db2ApiDf.h`):

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.

poApplicationId
Output. The application ID.

poNumReplies
Output. The number of replies received.

poNodeInfo
Output. Database partition reply information.

nodenum
Node number.

pathlen
The length of the new logpath.

logpath
The new overflow log path.

num_entry
Number of entries in the list pointed to by the *tablespace* field.

tablespace
A pointer to the *sqlu_tablepsace_entry* structure.

reserve_len
Length of the character string provided in the *tablespace_entry* field. For languages other than C.

tablespace_entry
Table space name.

state State information.

nextarclog
A buffer to hold the returned name of the next archived log file required. If a caller action other than DB2ROLLFORWARD_QUERY is supplied, the value returned in this field indicates that an error occurred when accessing the file. Possible causes are:

- The file was not found in the database log directory, nor on the path specified by the overflow log path parameter
- The log archiving method failed to return the archived file.

firstarcdel
A buffer to hold the returned name of the first archived log file no longer needed for recovery. This file, and all files up to and including lastarcdel, can be moved to make room on the disk.

For example, if the values returned in firstarcdel and lastarcdel are S0000001.LOG and S0000005.LOG, the following log files can be moved:

- `S0000001.LOG`
- `S0000002.LOG`
- `S0000003.LOG`
- `S0000004.LOG`
- `S0000005.LOG`

lastarcdel
A buffer to hold the returned name of the last archived log file that can be removed from the database log directory.

lastcommit
A string containing a time stamp in ISO format. This value represents the time stamp of the last committed transaction after the rollforward operation terminates.

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 *lascommit* 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.

Related reference:

- "SQLCA" in the *Administrative API Reference*
- "db2Restore - Restore database" on page 463

Related samples:

- "dbrecov.sqc -- How to recover a database (C)"
- "dbrecov.sqC -- How to recover a database (C++)"

db2SetWriteForDB - Set or Resume I/O

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.

Authorization:

One of the following:

- *sysadm*

- *sysctrl*
- *sysmaint*

Required connection:

Database

API include file:

db2ApiDf.h

C API syntax:

```
/* File: db2ApiDf.h */
/* API: db2SetWriteForDB */
/* ... */
SQL_API_RC SQL_API_FN
db2SetWriteForDB (
  db2Uint32 versionNumber,
  void *pParmStruct,
  struct sqlca *pSqlca);

typedef struct db2SetWriteDbStruct
{
  db2int32                    iOption;
  char                        *piTablespaceNames;
} db2SetWriteDbStruct;
/* ... */
```

API parameters:

version
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.

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

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:

- *sysadm* or *dbadm* authority
- BINDADD privilege if a package does not exist and one of:
  - IMPLICIT_SCHEMA authority on the database if the schema name of the package does not exist
  - CREATEIN privilege on the schema if the schema name of the package exists
- ALTERIN privilege on the schema if the package exists
- BIND privilege on the package if it exists.

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. If the user has *sysadm* authority, but not explicit privileges to complete the bind, the database manager grants explicit *dbadm* authority automatically.

Required connection:

Database

API include file:

sql.h

C API syntax:

```
/* File: sql.h */
/* API: sqlabndx */
/* ... */
SQL_API_RC SQL_API_FN
  sqlabndx (
    _SQLOLDCHAR *pBindFileName,
    _SQLOLDCHAR *pMsgFileName,
    struct sqlopt *pBindOptions,
    struct sqlca *pSqlca);
/* ... */
```

Generic API syntax:

```
/* File: sql.h */
/* API: sqlgbndx */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgbndx (
    unsigned short MsgFileNameLen,
    unsigned short BindFileNameLen,
    struct sqlca *pSqlca,
    struct sqlopt *pBindOptions,
    _SQLOLDCHAR *pMsgFileName,
    _SQLOLDCHAR *pBindFileName);
/* ... */
```

API parameters:

MsgFileNameLen
Input. A 2-byte unsigned integer representing the length of the message file name in bytes.

BindFileNameLen
Input. A 2-byte unsigned integer representing the length of the bind file name in bytes.

pSqlca
Output. A pointer to the *sqlca* structure.

pBindOptions
Input. A structure used to pass bind options to the API. For more information about this structure, see SQLOPT.

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.

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 may 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.

REXX API syntax:

This API can be called from REXX through the SQLDB2 interface.

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 V8 client to a V8 server, and then executed against a V7 server.

The Bind option types and values are defined in `sql`.

Related reference:

- "sqlaprep - Precompile Program" on page 871
- "SQLCA" in the *Administrative API Reference*
- "SQLCHAR" in the *Administrative API Reference*
- "SQLOPT" in the *Administrative API Reference*

Related samples:

- "dbpkg.sqc -- How to work with packages (C)"
- "dbsample.sqc -- Creates a sample database (C)"
- "dbpkg.sqC -- How to work with packages (C++)"

sqlbftpq - Fetch Table Space Query

Fetches a specified number of rows of table space query data, each row consisting of data for a table space.

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*
- *dbadm*
- *load*

Required connection:

Database

API include file:

sqlutil.h

C API syntax:

```
/* File: sqlutil.h */
/* API: sqlbftpq */
/* ... */
SQL_API_RC SQL_API_FN
  sqlbftpq (
    struct sqlca *pSqlca,
    sqluint32 MaxTablespaces,
    struct SQLB_TBSPQRY_DATA *pTablespaceData,
    sqluint32 *pNumTablespaces);
/* ... */
```

Generic API syntax:

```
/* File: sqlutil.h */
/* API: sqlgftpq */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgftpq (
    struct sqlca *pSqlca,
    sqluint32 MaxTablespaces,
    struct SQLB_TBSPQRY_DATA *pTablespaceData,
    sqluint32 *pNumTablespaces);
/* ... */
```

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.

Related reference:

- "sqlbotsq - Open Table Space Query" in the *Administrative API Reference*
- "sqlbctsq - Close Table Space Query" in the *Administrative API Reference*

- "sqlbmtsq - Table Space Query" on page 489
- "sqlbgtss - Get Table Space Statistics" in the *Administrative API Reference*
- "sqlbstpq - Single Table Space Query" on page 493
- "SQLCA" in the *Administrative API Reference*
- "SQLB-TBSPQRY-DATA" in the *Administrative API Reference*

Related samples:

- "tabspace.sqb -- How to get tablespace information (IBM COBOL)"
- "tspace.sqb -- How to copy and free memory in a tablespace (IBM COBOL)"
- "tsinfo.sqc -- How to get information at the table space level (C)"
- "tsinfo.sqC -- How to get information at the table space level (C++)"

sqlbmtsq - Table Space Query

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.

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*
- *dbadm*
- *load*

Required connection:

Database

API include file:

sqlutil.h

C API syntax:

```
/* File: sqlutil.h */
/* API: sqlbmtsq */
/* ... */
SQL_API_RC SQL_API_FN
  sqlbmtsq (
    struct sqlca *pSqlca,
    sqluint32 *pNumTablespaces,
    struct SQLB_TBSPQRY_DATA ***pppTablespaceData,
    sqluint32 reserved1,
    sqluint32 reserved2);
/* ... */
```

Generic API syntax:

```
/* File: sqlutil.h */
/* API: sqlgmtsq */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgmtsq (
    struct sqlca *pSqlca,
    sqluint32 *pNumTablespaces,
    struct SQLB_TBSPQRY_DATA ***pppTablespaceData,
    sqluint32 reserved1,
    sqluint32 reserved2);
/* ... */
```

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.

Related reference:

- "sqlbotsq - Open Table Space Query" in the *Administrative API Reference*
- "sqlbftpq - Fetch Table Space Query" on page 487
- "sqlbctsq - Close Table Space Query" in the *Administrative API Reference*
- "sqlbgtss - Get Table Space Statistics" in the *Administrative API Reference*
- "sqlbstpq - Single Table Space Query" on page 493
- "sqlefmem - Free Memory" in the *Administrative API Reference*
- "SQLCA" in the *Administrative API Reference*

Related samples:

- "dbrecov.sqc -- How to recover a database (C)"
- "tsinfo.sqc -- How to get information at the table space level (C)"
- "dbrecov.sqC -- How to recover a database (C++)"
- "tsinfo.sqC -- How to get information at the table space level (C++)"
- "tspace.sqb -- How to copy and free memory in a tablespace (IBM COBOL)"

sqlbotcq - Open 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*
- *dbadm*

Required connection:

Database

API include file:

sqlutil.h

C API syntax:

```
/* File: sqlutil.h */
/* API: sqlbotcq */
/* ... */
SQL_API_RC SQL_API_FN
  sqlbotcq (
    struct sqlca *pSqlca,
    sqluint32 TablespaceId,
    sqluint32 *pNumContainers);
/* ... */
```

Generic API syntax:

```
/* File: sqlutil.h */
/* API: sqlgotcq */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgotcq (
    struct sqlca *pSqlca,
    sqluint32 TablespaceId,
    sqluint32 *pNumContainers);
/* ... */
```

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.

Related reference:

- "sqlbftcq - Fetch Table Space Container Query" in the *Administrative API Reference*
- "sqlbctcq - Close Table Space Container Query" in the *Administrative API Reference*
- "sqlbtcq - Table Space Container Query" in the *Administrative API Reference*
- "sqlbstsc - Set Table Space Containers" in the *Administrative API Reference*
- "sqlefmem - Free Memory" in the *Administrative API Reference*

- "SQLCA" in the *Administrative API Reference*

Related samples:

- "tabscont.sqb -- How to get tablespace container information (IBM COBOL)"
- "tspace.sqb -- How to copy and free memory in a tablespace (IBM COBOL)"
- "tsinfo.sqc -- How to get information at the table space level (C)"
- "tsinfo.sqC -- How to get information at the table space level (C++)"

sqlbstpq - Single Table Space Query

Retrieves information about a single currently defined table space.

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*
- *dbadm*
- *load*

Required connection:

Database

API include file:

sqlutil.h

C API syntax:

```
/* File: sqlutil.h */
/* API: sqlbstpq */
/* ... */
SQL_API_RC SQL_API_FN
  sqlbstpq (
    struct sqlca *pSqlca,
    sqluint32 TablespaceId,
    struct SQLB_TBSPQRY_DATA *pTablespaceData,
    sqluint32 reserved);
/* ... */
```

Generic API syntax:

```
/* File: sqlutil.h */
/* API: sqlgstpq */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgstpq (
    struct sqlca *pSqlca,
```

```
    sqluint32 TablespaceId,
    struct SQLB_TBSPQRY_DATA *pTablespaceData,
    sqluint32reserved);
/* ... */
```

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.

For more information, see sqlbotsq.

Related reference:

- "sqlbotsq - Open Table Space Query" in the *Administrative API Reference*
- "sqlbftpq - Fetch Table Space Query" on page 487
- "sqlbctsq - Close Table Space Query" in the *Administrative API Reference*
- "sqlbmtsq - Table Space Query" on page 489
- "sqlbgtss - Get Table Space Statistics" in the *Administrative API Reference*
- "SQLCA" in the *Administrative API Reference*

Related samples:

- "tabspace.sqb -- How to get tablespace information (IBM COBOL)"
- "tspace.sqb -- How to copy and free memory in a tablespace (IBM COBOL)"
- "tsinfo.sqc -- How to get information at the table space level (C)"
- "tsinfo.sqC -- How to get information at the table space level (C++)"

sqlecadb - 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 node.

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

C API syntax:

```
/* File: sqlenv.h */
/* API: sqlecadb */
/* ... */
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);
/* ... */
```

Generic API syntax:

```
/* File: sqlenv.h */
/* API: sqlgcadb */
/* ... */
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);
/* ... */
```

API 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 zero 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 zero if no path is provided.

NodeNameLen
Input. A 2-byte unsigned integer representing the length in bytes of the node name. Set to zero 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.

pSqlca
Output. A pointer to the *sqlca* structure.

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.

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) are:

SQL_AUTHENTICATION_SERVER
Specifies that authentication takes place on the node containing the target database.

SQL_AUTHENTICATION_CLIENT
Specifies that authentication takes place on the node 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 node containing the target database, and that the authentication password is to be encrypted.

SQL_AUTHENTICATION_DATAENC
Specifies that authentication takes place on the node 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.

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.

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.

pPath Input. A string which, on UNIX based 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*.

pNodeName
Input. A string containing the name of the node 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*.

Type Input. A single character that designates whether the database is indirect, remote, or is cataloged via DCE. Valid values (defined in sqlenv) 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.

pDbAlias
Input. A string containing an alias for the database.

pDbName
Input. A string containing the database name.

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 node containing the target database. This is the default.

CLIENT
Authentication occurs at the node 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 node containing the target database, and that the authentication password is to be encrypted.

DATAENC
Specifies that authentication takes place on the node 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.

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"'
```

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.

Related reference:

- "db2DbDirCloseScan - Close Database Directory Scan" in the *Administrative API Reference*
- "db2DbDirGetNextEntry - Get Next Database Directory Entry" in the *Administrative API Reference*
- "db2DbDirOpenScan - Open Database Directory Scan" in the *Administrative API Reference*
- "sqleuncd - Uncatalog Database" in the *Administrative API Reference*
- "SQLCA" in the *Administrative API Reference*

Related samples:

- "dbcat.cbl -- Catalog to and uncatalog from a database (IBM COBOL)"

- "ininfo.c -- Set and get information at the instance level (C)"
- "ininfo.C -- Set and get information at the instance level (C++)"

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

C API syntax:

```
/* File: sqlenv.h */
/* API: sqlecrea */
/* ... */
SQL_API_RC SQL_API_FN
  sqlecrea (
    char *pDbName,
    char *pLocalDbAlias,
    char *pPath,
    struct sqledbdesc *pDbDescriptor,
    struct sqledbterritoryinfo *pTerritoryInfo,
    char Reserved2,
    void *pReserved1,
    struct sqlca *pSqlca);
/* ... */
```

Generic API syntax:

```
/* File: sqlenv.h */
/* API: sqlgcrea */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgcrea (
    unsigned short PathLen,
    unsigned short LocalDbAliasLen,
    unsigned short DbNameLen,
```

```
    struct sqlca *pSqlca,
    void *pReserved1,
    unsigned short Reserved2,
    struct sqledbterritoryinfo *pTerritoryInfo,
    struct sqledbdesc *pDbDescriptor,
    char *pPath,
    char *pLocalDbAlias,
    char *pDbName);
/* ... */
```

API 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.

pSqlca
Output. A pointer to the *sqlca* structure.

pReserved1
Input. A spare pointer that is set to null or points to zero.

Reserved2
Input. Reserved for future use.

pTerritoryInfo
Input. A pointer to the *sqledbterritoryinfo* structure, containing the locale and the code set for the database. May be NULL.

pDbDescriptor
Input. A pointer to the database description block used when creating the database. The database description block may be used to supply values that are permanently stored in the configuration file of the database, such as collating sequence. May be NULL. For information about the supported collating sequences for Unicode databases, see the topic about the database description block (SQLEDBDESC).

pPath Input. On UNIX based 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. May 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.

pLocalDbAlias
Input. A string containing the alias to be placed in the client's system database directory. May be NULL. If no local alias is specified, the database name is the default.

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.

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
Collating sequence based on the database territory.

IDENTITY
The collation sequence as determined by the binary order of each byte of the string, where strings are compared byte for byte, starting with the leftmost byte.

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 segment directories that will be created and used to store the DAT, IDX, and LF files.

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 4KB 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.

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 `$DB2INSTANCE/NODE`*xxxx* directory under the specified subdirectory at each database partition server, where *xxxx* represents the local database partition server number. In a single-partition environment, creates a `$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.

- 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 authority, and CONNECT, CREATETAB, BINDADD, CREATE_NOT_FENCED, IMPLICIT_SCHEMA, and LOAD privileges to the database creator
  - CONNECT, CREATETAB, BINDADD, and IMPLICIT_SCHEMA privileges 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.

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, `$DB2INSTANCE/NODE`*xxxx*, under the specified or default path on all database partition servers. The *xxxx* is the node number as defined in the `db2nodes.cfg` file (that is, node 0 becomes `NODE0000`). Subdirectories `SQL00001` through `SQL`*nnnnn* will reside on this path. This ensures that the database objects associated with different database partition servers are stored in different directories (even if the subdirectory `$DB2INSTANCE` under the specified or default path is shared by all database partition servers).

On Windows and AIX, the length of the code set name is limited to a maximum of 9 characters. For example, specify a code set name such as `ISO0885915` instead of `ISO8859-15`.

CREATE DATABASE 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 "eye-catcher" 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 CREATE DATABASE cannot be changed later, and all character comparisons in the database use the specified collating sequence. This affects the structure of indexes as well as the results of queries.

Use **sqlecadb** to define different alias names for the new database.

Related reference:

- "sqlabndx - Bind" on page 484
- "sqlecadb - Catalog Database" on page 494
- "sqledrpd - Drop Database" on page 506
- "sqlecran - Create Database at Node" in the *Administrative API Reference*
- "sqledpan - Drop Database at Node" in the *Administrative API Reference*
- "SQLEDBTERRITORYINFO" in the *Administrative API Reference*
- "SQLCA" in the *Administrative API Reference*
- "SQLEDBDESC" in the *Administrative API Reference*
- "CREATE DATABASE" on page 252

Related samples:

- "db_udcs.cbl -- How to use user-defined collating sequence (IBM COBOL)"
- "dbconf.cbl -- Update database configuration (IBM COBOL)"
- "ebcdicdb.cbl -- Create a database with EBCDIC 037 standard collating sequence (IBM COBOL)"
- "dbcreate.c -- Create and drop databases (C)"
- "dbrecov.sqc -- How to recover a database (C)"
- "dbsample.sqc -- Creates a sample database (C)"
- "dbcreate.C -- Create and drop databases (C++)"
- "dbrecov.sqC -- How to recover a database (C++)"

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

C API syntax:

```
/* File: sqlenv.h */
/* API: sqledrpd */
/* ... */
SQL_API_RC SQL_API_FN
  sqledrpd (
    _SQLOLDCHAR *pDbAlias,
    struct sqlca *pSqlca);
/* ... */
```

Generic API syntax:

```
/* File: sqlenv.h */
/* API: sqlgdrpd */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgdrpd (
    unsigned short Reserved1,
    unsigned short DbAliasLen,
    struct sqlca *pSqlca,
    _SQLOLDCHAR *pReserved2,
    _SQLOLDCHAR *pDbAlias);
/* ... */
```

API parameters:

Reserved1
Reserved for future use.

DbAliasLen
Input. A 2-byte unsigned integer representing the length in bytes of the database alias.

pSqlca
Output. A pointer to the *sqlca* structure.

pReserved2
A spare pointer that is set to null or points to zero. Reserved for future use.

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.

REXX API syntax:

```
DROP DATABASE dbalias
```

REXX API parameters:

dbalias
The alias of the database to be dropped.

Usage notes:

sqledrpd 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.

This API unlinks all files that are linked through any DATALINK columns. Since the unlink operation is performed asynchronously on the DB2 Data Links Manager, its effects may not be seen immediately on the DB2 Data Links Manager, and the unlinked files may not be immediately available for other operations. When the API is called, all the DB2 Data Links Managers configured to that database must be available; otherwise, the drop database operation will fail.

Related reference:

- "sqlecadb - Catalog Database" on page 494
- "sqlecrea - Create Database" on page 500
- "sqleuncd - Uncatalog Database" in the *Administrative API Reference*
- "sqlecran - Create Database at Node" in the *Administrative API Reference*
- "sqledpan - Drop Database at Node" in the *Administrative API Reference*
- "SQLCA" in the *Administrative API Reference*

Related samples:

- "dbconf.cbl -- Update database configuration (IBM COBOL)"
- "dbcreate.c -- Create and drop databases (C)"
- "dbcreate.C -- Create and drop databases (C++)"

sqlemgdb - Migrate Database

Converts previous (Version 2.x or higher) versions of DB2 databases to current formats.

Authorization:

sysadm

Required connection:

This API establishes a database connection.

API include file:

sqlenv.h

C API syntax:

```
/* File: sqlenv.h */
/* API: sqlemgdb */
/* ... */
SQL_API_RC SQL_API_FN
  sqlemgdb (
    _SQLOLDCHAR *pDbAlias,
    _SQLOLDCHAR *pUserName,
    _SQLOLDCHAR *pPassword,
    struct sqlca *pSqlca);
/* ... */
```

Generic API syntax:

```
/* File: sqlenv.h */
/* API: sqlgmgdb */
/* ... */
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);
/* ... */
```

API 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.

pSqlca
Output. A pointer to the *sqlca* structure.

pPassword
Input. A string containing the password of the supplied user name (if any). May be NULL.

pUserName
Input. A string containing the user name of the application. May be NULL.

pDbAlias
Input. A string containing the alias of the database that is cataloged in the system database directory.

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.

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.

Related reference:

- "SQLCA" in the *Administrative API Reference*

Related samples:

- "dbmigrat.c -- Migrate a database (C)"
- "dbmigrat.C -- Migrate a database (C++)"
- "migrate.cbl -- Demonstrates how to migrate to a database (IBM COBOL)"

sqluadau - Get Authorizations

Reports the authorities of the current user from values found in the database manager configuration file and the authorization system catalog view (SYSCAT.DBAUTH).

Authorization:

None

Required connection:

Database

API include file:

sqlutil.h

C API syntax:

```
/* File: sqlutil.h */
/* API: sqluadau */
/* ... */
SQL_API_RC SQL_API_FN
  sqluadau (
    struct sql_authorizations *pAuthorizations,
    struct sqlca *pSqlca);
/* ... */
```

Generic API syntax:

```
/* File: sqlutil.h */
/* API: sqlgadau */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgadau (
    struct sql_authorizations *pAuthorizations,
    struct sqlca *pSqlca);
/* ... */
```

API parameters:

pAuthorizations
Input/Output. Pointer to the *sql_authorizations* structure. This array of short integers indicates which authorizations the current user holds. The first element in the structure, *sql_authorizations_len*, must be initialized to the size of the buffer being passed, prior to calling this API.

pSqlca
Output. A pointer to the *sqlca* structure.

REXX API syntax:

```
GET AUTHORIZATIONS :value
```

REXX API parameters:

value A compound REXX host variable to which the authorization level is returned. In the following, XXX represents the host variable name. Values are `0` for no, and `1` for yes.

XXX.0	Number of elements in the variable (always 18)
XXX.1	Direct SYSADM authority
XXX.2	Direct DBADM authority
XXX.3	Direct CREATETAB authority
XXX.4	Direct BINDADD authority
XXX.5	Direct CONNECT authority
XXX.6	Indirect SYSADM authority
XXX.7	Indirect DBADM authority
XXX.8	Indirect CREATETAB authority
XXX.9	Indirect BINDADD authority
XXX.10	Indirect CONNECT authority
XXX.11	Direct SYSCTRL authority
XXX.12	Indirect SYSCTRL authority
XXX.13	Direct SYSMAINT authority
XXX.14	Indirect SYSMAINT authority
XXX.15	Direct CREATE_NOT_FENC authority
XXX.16	Indirect CREATE_NOT_FENC authority
XXX.17	Direct IMPLICIT_SCHEMA authority
XXX.18	Indirect IMPLICIT_SCHEMA authority.

XXX.19	Direct LOAD authority.
XXX.20	Indirect LOAD authority.

Usage notes:

Direct authorities are acquired by explicit commands that grant the authorities to a user ID. Indirect authorities are based on authorities acquired by the groups to which a user belongs.

Note: PUBLIC is a special group to which all users belong.

If there are no errors, each element of the *sql_authorizations* structure contains a `0` or a 1. A value of 1 indicates that the user holds that authorization; `0` indicates that the user does not.

Related reference:

- "SQL-AUTHORIZATIONS" on page 1016
- "SQLCA" in the *Administrative API Reference*

Related samples:

- "dbauth.sqb -- How to grant and display authorities on a database (IBM COBOL)"
- "dbauth.sqc -- How to grant, display, and revoke authorities at database level (C)"
- "inauth.sqc -- How to display authorities at instance level (C)"
- "dbauth.sqC -- How to grant, display, and revoke authorities at database level (C++)"
- "inauth.sqC -- How to display authorities at instance level (C++)"

sqlurcon - Reconcile

Validates the references to files for the DATALINK data of a table. The rows for which the references to files cannot be established are copied to the exception table (if specified), and modified in the input table.

Authorization:

One of the following:

- *sysadm*
- *sysctrl*
- *sysmaint*
- *dbadm*
- CONTROL privilege on the table.

Required connection:

Database

API include file:

sqlutil.h

C API syntax:

```
/* File: sqlutil.h */
/* API: sqlurcon */
/* ... */
SQL_API_RC SQL_API_FN
  sqlurcon (
    char *pTableName,
    char *pExTableName,
    char *pReportFileName,
    void *pReserved,
    struct sqlca *pSqlca);
/* ... */
```

Generic API syntax:

```
/* File: sqlutil.h */
/* API: sqlgrcon */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgrcon (
    unsigned short TableNameLen,
    char *pTableName,
    unsigned short ExTableNameLen,
    char *pExTableName,
    unsigned short ReportFleNameLen,
    char *pReportFileName,
    void *pReserved,
    struct sqlca *pSqlca);
/* ... */
```

API parameters:

TableNameLen
Input. A 2-byte unsigned integer representing the length in bytes of the table name.

pTableName
Input. Specifies the table on which reconciliation is to be performed. 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 authorization ID.

ExTableNameLen
Input. A 2-byte unsigned integer representing the length in bytes of the exception table name.

pExTableName
Input. Specifies the exception table into which rows that encounter link failures for DATALINK values are to be copied.

ReportFileNameLen
Input. A 2-byte unsigned integer representing the length in bytes of the report file name.

pReportFileName
Input. Specifies the file that will contain information about the files that are unlinked during reconciliation. The name must be fully qualified (for example, `/u/johnh/report`). The reconcile utility appends a `.ulk` extension to the specified file name (for example, `report.ulk`).

pReserved
Reserved for future use.

pSqlca
Output. A pointer to the *sqlca* structure.

Usage notes:

During reconciliation, attempts are made to link files which exist according to table data, but which do not exist according to Data Links File Manager metadata, if no other conflict exists.

Reconciliation is performed with respect to all DATALINK data in the table. If file references cannot be re-established, the violating rows are inserted into the exception table (if specified). These rows are not deleted from the input table. To ensure file reference integrity, the offending DATALINK values are nulled. If the column is defined as not nullable, the DATALINK values are replaced by a zero length URL.

If an exception table is not specified, the DATALINK column values for which file references cannot be re-established are copied to an exception report file (`<pReportFileName>.exp`), along with the column ID and a comment.

At the end of the reconciliation process, the table is taken out of datalink reconcile pending (DRP) state.

Related reference:
- "SQLCA" in the *Administrative API Reference*

sqluvqdp - Quiesce Table Spaces for 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, and 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

C API syntax:

```
/* File: sqlutil.h */
/* API: sqluvqdp */
/* ... */
SQL_API_RC SQL_API_FN
  sqluvqdp (
    char *pTableName,
    sqlint32 QuiesceMode,
    void *pReserved,
    struct sqlca *pSqlca);
/* ... */
```

Generic API syntax:

```
/* File: sqlutil.h */
/* API: sqlgvqdp */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgvqdp (
    unsigned short TableNameLen,
    char *pTableName,
    sqlint32 QuiesceMode,
    void *pReserved,
    struct sqlca *pSqlca);
/* ... */
```

API parameters:

TableNameLen
Input. A 2-byte unsigned integer representing the length in bytes of the table name.

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.

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.

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.

Related reference:

- "SQLCA" in the *Administrative API Reference*
- "db2DatabaseQuiesce - Database Quiesce" on page 402
- "db2InstanceQuiesce - Instance Quiesce" in the *Administrative API Reference*

Related samples:

- "tbmove.sqc -- How to move table data (C)"
- "tbmove.sqC -- How to move table data (C++)"
- "tload.sqb -- How to export and load table data (IBM COBOL)"

Chapter 15. SQL Statements for Administrators

ALTER FUNCTION

The ALTER FUNCTION statement modifies the properties of an existing function.

Invocation:

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization:

The privileges held by the authorization ID of the statement must include at least one of the following:

- SYSADM or DBADM authority
- ALTERIN privilege on the schema of the function
- Definer of the function, as recorded in the DEFINER column of SYSCAT.ROUTINES

To alter the EXTERNAL NAME of a function, the privileges held by the authorization ID of the statement must also include at least one of the following:

- SYSADM or DBADM authority
- CREATE_EXTERNAL_ROUTINE authority on the database

To alter a function to be not fenced, the privileges held by the authorization ID of the statement must also include at least one of the following:

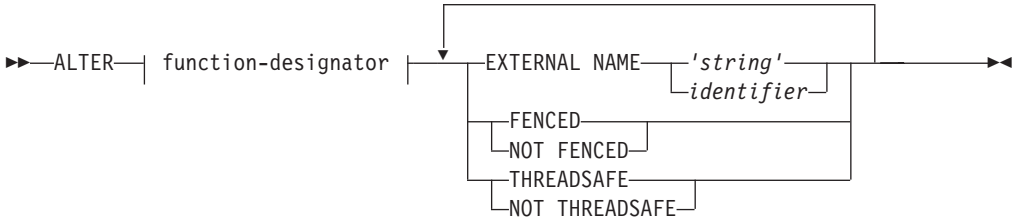
- SYSADM or DBADM authority
- CREATE_NOT_FENCED_ROUTINE authority on the database

© Copyright IBM Corp. 1993-2004

To alter a function to be fenced, no additional authorities or privileges are required.

If the authorization ID has insufficient authority to perform the operation, an error (SQLSTATE 42502) is raised.

Syntax:



Description:

function-designator
Uniquely identifies the function to be altered. For more information, see Common syntax elements .

EXTERNAL NAME *'string'* **or** *identifier*
Identifies the name of the user-written code that implements the function. This option can only be specified when altering external functions (SQLSTATE 42849).

FENCED or **NOT FENCED**
Specifies whether the function is considered safe to run in the database manager operating environment's process or address space (NOT FENCED), or not (FENCED). Most functions have the option of running as FENCED or NOT FENCED.

If a function is altered to be FENCED, the database manager insulates its internal resources (for example, data buffers) from access by the function. In general, a function running as FENCED will not perform as well as a similar one running as NOT FENCED.

CAUTION:
Use of NOT FENCED for functions that were not adequately coded, reviewed, and tested can compromise the integrity of DB2. DB2 takes some precautions against many of the common types of inadvertent failures that might occur, but cannot guarantee complete integrity when NOT FENCED user-defined functions are used.

A function declared as NOT THREADSAFE cannot be altered to be NOT FENCED (SQLSTATE 42613).

If a function has any parameters defined AS LOCATOR, and was defined with the NO SQL option, the function cannot be altered to be FENCED (SQLSTATE 42613).

This option cannot be altered for LANGUAGE OLE, OLEDB, or CLR functions (SQLSTATE 42849).

THREADSAFE or **NOT THREADSAFE**
Specifies whether the function is considered safe to run in the same process as other routines (THREADSAFE), or not (NOT THREADSAFE).

If the function is defined with LANGUAGE other than OLE and OLEDB:

- If the function is defined as THREADSAFE, the database manager can invoke the function in the same process as other routines. In general, to be threadsafe, a function should not use any global or static data areas. Most programming references include a discussion of writing threadsafe routines. Both FENCED and NOT FENCED functions can be THREADSAFE.
- If the function is defined as NOT THREADSAFE, the database manager will never invoke the function in the same process as another routine. Only a fenced function can be NOT THREADSAFE (SQLSTATE 42613).

This option may not be altered for LANGUAGE OLE or OLEDB functions (SQLSTATE 42849).

Notes:

- It is not possible to alter a function that is in the SYSIBM, SYSFUN, or SYSPROC schema (SQLSTATE 42832).
- Functions declared as LANGUAGE SQL, sourced functions, or template functions cannot be altered (SQLSTATE 42917).

Example:

The function MAIL() has been thoroughly tested. To improve its performance, alter the function to be not fenced.

```
ALTER FUNCTION MAIL() NOT FENCED
```

Related reference:

- "CREATE FUNCTION (OLE DB External Table) statement" in the *SQL Reference, Volume 2*
- "CREATE FUNCTION (External Scalar) statement" in the *SQL Reference, Volume 2*
- "CREATE FUNCTION (External Table) statement" in the *SQL Reference, Volume 2*
- "Common syntax elements" in the *SQL Reference, Volume 2*

ALTER METHOD

The ALTER METHOD statement modifies an existing method by changing the method body associated with the method.

Invocation:

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization:

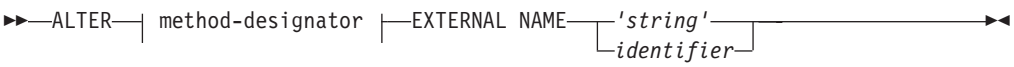
The privileges held by the authorization ID of the statement must include at least one of the following:

- SYSADM or DBADM authority
- CREATE_EXTERNAL_ROUTINE authority on the database, and at least one of:
  - ALTERIN privilege on the schema of the type

– Definer of the type, as recorded in the DEFINER column of SYSCAT.DATATYPES

If the authorization ID has insufficient authority to perform the operation, an error (SQLSTATE 42502) is raised.

Syntax:



Description:

method-designator
Uniquely identifies the method to be altered. For more information, see Common syntax elements .

EXTERNAL NAME *'string'* **or** *identifier*
Identifies the name of the user-written code that implements the method. This option can only be specified when altering external methods (SQLSTATE 42849).

Notes:

- It is not possible to alter a method that is in the SYSIBM, SYSFUN, or SYSPROC schema (SQLSTATE 42832).
- Methods declared as LANGUAGE SQL cannot be altered (SQLSTATE 42917).
- Methods declared as LANGUAGE CLR cannot be altered (SQLSTATE 42849).
- The specified method must have a body before it can be altered (SQLSTATE 42704).

Example:

Alter the method DISTANCE() in the structured type ADDRESS_T to use the library newaddresslib.

```
ALTER METHOD DISTANCE()
  FOR TYPE ADDRESS_T
  EXTERNAL NAME 'newaddresslib!distance2'
```

Related reference:

- "CREATE METHOD" on page 583
- "Common syntax elements" in the *SQL Reference, Volume 2*

ALTER PROCEDURE

The ALTER PROCEDURE statement modifies an existing procedure by changing the properties of the procedure.

Invocation:

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization:

The privileges held by the authorization ID of the statement must include at least one of the following:

- SYSADM or DBADM authority
- ALTERIN privilege on the schema of the procedure
- Definer of the procedure, as recorded in the DEFINER column of SYSCAT.ROUTINES

To alter the EXTERNAL NAME of a procedure, the privileges held by the authorization ID of the statement must also include at least one of the following:

- SYSADM or DBADM authority
- CREATE_EXTERNAL_ROUTINE authority on the database

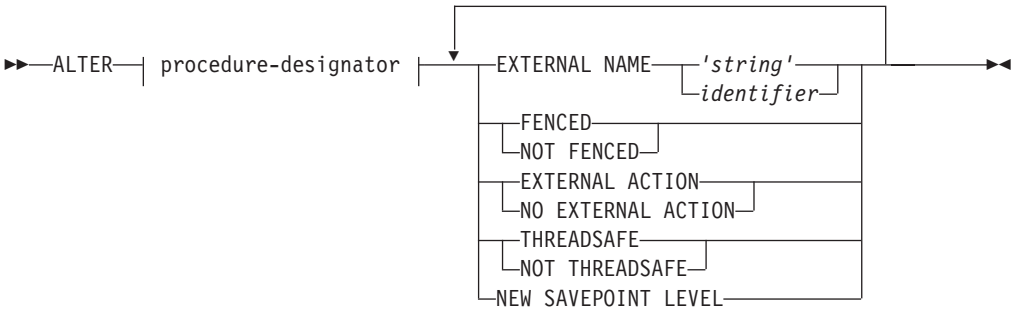
To alter a procedure to be not fenced, the privileges held by the authorization ID of the statement must also include at least one of the following:

- SYSADM or DBADM authority
- CREATE_NOT_FENCED_ROUTINE authority on the database

To alter a procedure to be fenced, no additional authorities or privileges are required.

If the authorization ID has insufficient authority to perform the operation, an error (SQLSTATE 42502) is raised.

Syntax:



Description:

procedure-designator
Uniquely identifies the procedure to be altered. For more information, see Common syntax elements .

EXTERNAL NAME *'string'* **or** *identifier*
Identifies the name of the user-written code that implements the procedure. This option can only be specified when altering external procedures (SQLSTATE 42849). The EXTERNAL NAME clause cannot be altered in procedures that were declared as LANGUAGE SQL (SQLSTATE 42917).

FENCED or **NOT FENCED**
Specifies whether the procedure is considered safe to run in the database manager operating environment's process or address space (NOT FENCED), or not (FENCED). Most procedures have the option of running as FENCED or NOT FENCED.

If a procedure is altered to be FENCED, the database manager insulates its internal resources (for example, data buffers) from access by the procedure. In general, a procedure running as FENCED will not perform as well as a similar one running as NOT FENCED.

CAUTION:
Use of NOT FENCED for procedures that were not adequately coded, reviewed, and tested can compromise the integrity of DB2. DB2 takes some precautions against many of the common types of inadvertent failures that might occur, but cannot guarantee complete integrity when NOT FENCED stored procedures are used.

This option can only be specified when altering external procedures (SQLSTATE 42849).

A procedure declared as NOT THREADSAFE cannot be altered to be NOT FENCED (SQLSTATE 42613).

If a procedure has any parameters defined AS LOCATOR, and was defined with the NO SQL option, the procedure cannot be altered to be FENCED (SQLSTATE 42613).

This option cannot be altered for LANGUAGE OLE or CLR procedures (SQLSTATE 42849).

The FENCED or NOT FENCED clause cannot be altered in procedures that were declared as LANGUAGE SQL (SQLSTATE 42917).

EXTERNAL ACTION or **NO EXTERNAL ACTION**
Specifies whether the procedure takes some action that changes the state of an object not managed by the database manager (EXTERNAL ACTION), or not (NO EXTERNAL ACTION). If NO EXTERNAL ACTION is specified, the system can use certain optimizations that assume the procedure has no external impact.

THREADSAFE or **NOT THREADSAFE**
Specifies whether the procedure is considered safe to run in the same process as other routines (THREADSAFE), or not (NOT THREADSAFE).

If the procedure is defined with LANGUAGE other than OLE:

- If the procedure is defined as THREADSAFE, the database manager can invoke the procedure in the same process as other routines. In general, to be threadsafe, a procedure should not use any global or static data areas. Most programming references include a discussion of writing threadsafe routines. Both FENCED and NOT FENCED procedures can be THREADSAFE.
- If the procedure is defined as NOT THREADSAFE, the database manager will never invoke the procedure in the same process as another routine. Only a fenced procedure can be NOT THREADSAFE (SQLSTATE 42613).

This option can only be specified when altering external procedures (SQLSTATE 42849).

This option cannot be altered for LANGUAGE OLE procedures (SQLSTATE 42849).

The THREADSAFE or NOT THREADSAFE clause cannot be altered in procedures that were declared as LANGUAGE SQL (SQLSTATE 42917).

NEW SAVEPOINT LEVEL
Specifies that a new savepoint level is to be created for the procedure. A

savepoint level refers to the scope of reference for any savepoint-related statement, as well as to the name space used for comparison and reference of any savepoint names.

The savepoint level for a procedure can only be altered to NEW SAVEPOINT LEVEL.

Rules:

- It is not possible to alter a procedure that is in the SYSIBM, SYSFUN, or SYSPROC schema (SQLSTATE 42832).

Example:

Alter the procedure PARTS_ON_HAND() to be not fenced.

```
ALTER PROCEDURE PARTS_ON_HAND() NOT FENCED
```

Related reference:

- "CREATE PROCEDURE" on page 588
- "Common syntax elements" in the *SQL Reference, Volume 2*

ALTER TABLE

The ALTER TABLE statement alters the definition of a table.

Invocation:

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization:

The privileges held by the authorization ID of the statement must include at least one of the following:

- ALTER privilege on the table to be altered
- CONTROL privilege on the table to be altered
- ALTERIN privilege on the schema of the table
- SYSADM or DBADM authority.

To create or drop a foreign key, the privileges held by the authorization ID of the statement must include one of the following on the parent table:

- REFERENCES privilege on the table
- REFERENCES privilege on each column of the specified parent key
- CONTROL privilege on the table
- SYSADM or DBADM authority.

To drop a primary key or unique constraint of table T, the privileges held by the authorization ID of the statement must include at least one of the following on every table that is a dependent of this parent key of T:

- ALTER privilege on the table
- CONTROL privilege on the table

- ALTERIN privilege on the schema of the table
- SYSADM or DBADM authority.

To alter a table to become a materialized query table (using a fullselect), the privileges held by the authorization ID of the statement must include at least one of the following:

- CONTROL on the table
- SYSADM or DBADM authority;

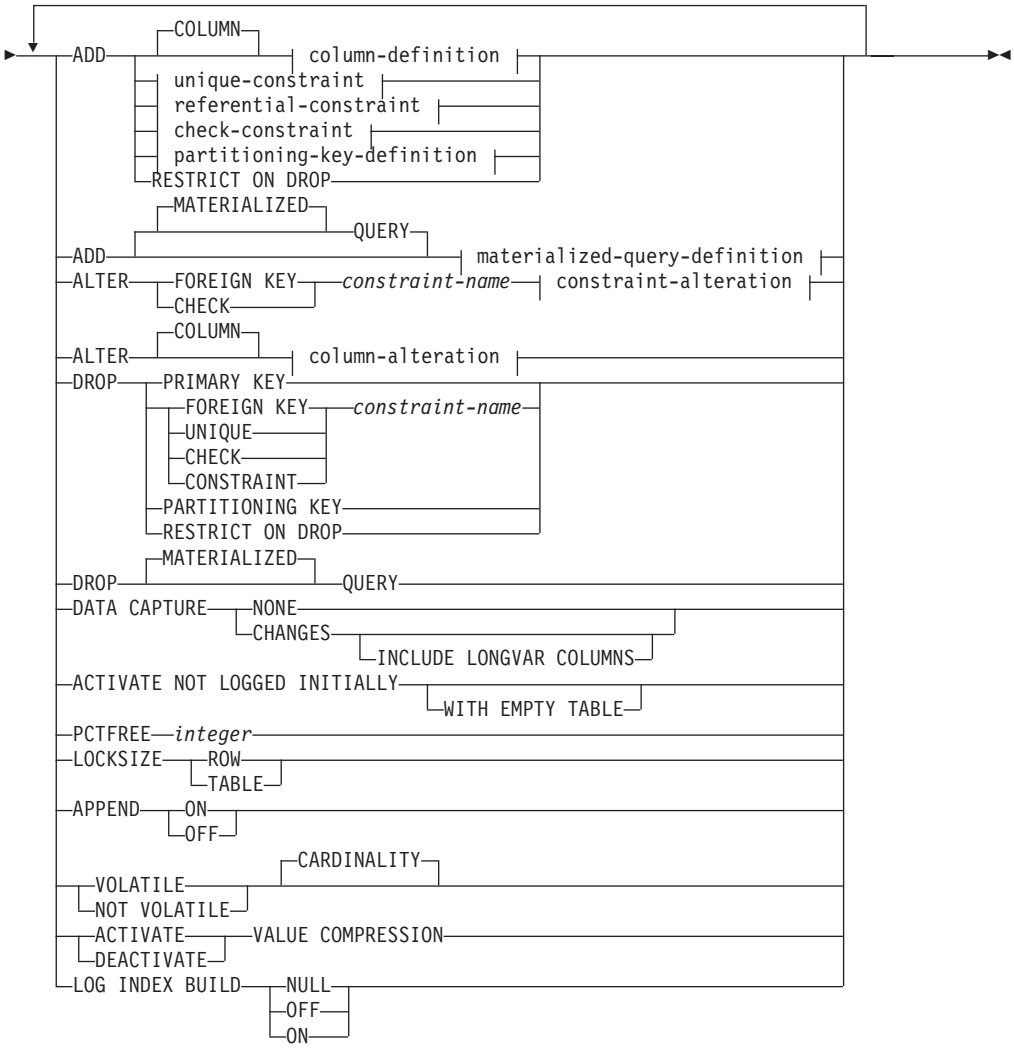
and at least one of the following, on each table or view identified in the fullselect:

- SELECT and ALTER privilege on the table or view
- CONTROL privilege on the table or view
- SELECT privilege on the table or view and ALTERIN privilege on the schema of the table or view
- SYSADM or DBADM authority.

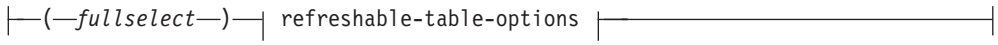
To alter a table so that it is no longer a materialized query table, the privileges held by the authorization ID of the statement must include at least one of the following, on each table or view identified in the fullselect used to define the materialized query table:

- ALTER privilege on the table or view
- CONTROL privilege on the table or view
- ALTERIN privilege on the schema of the table or view
- SYSADM or DBADM authority

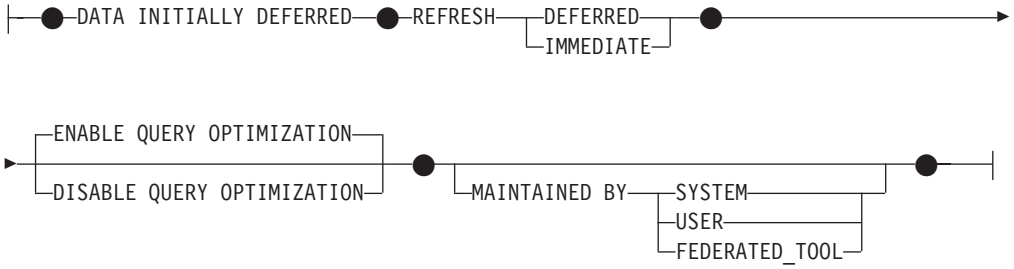
Syntax:



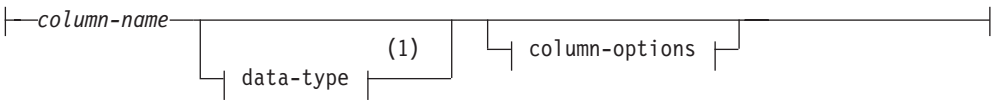
materialized-query-definition:



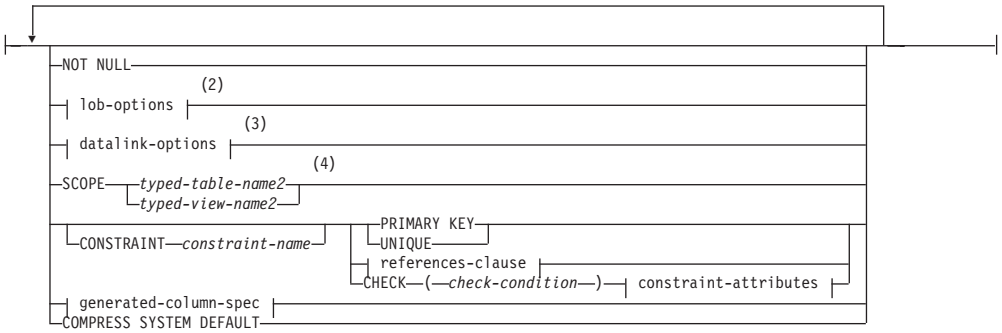
refreshable-table-options:



column-definition:



column-options:



Notes:

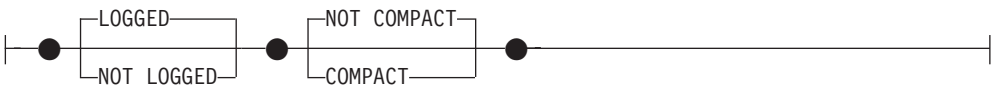
1 If the first column-option chosen is the generated-column-spec, then the data-type can be omitted and computed by the generation-expression.

2 The lob-options clause only applies to large object types (BLOB, CLOB and DBCLOB) and distinct types based on large object types.

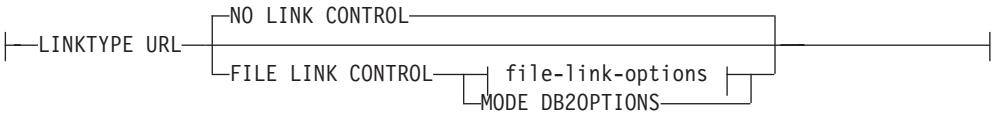
3 The datalink-options clause only applies to the DATALINK type and distinct types based on the DATALINK type.

4 The SCOPE clause only applies to the REF type.

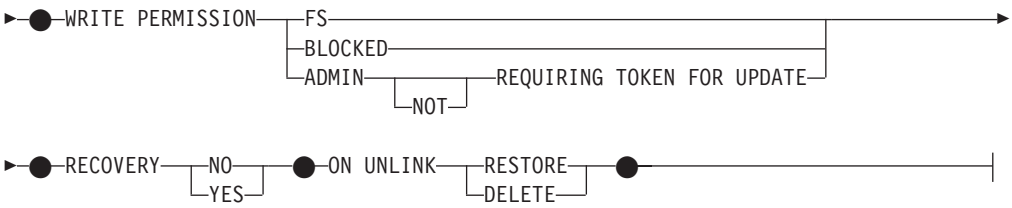
lob-options:



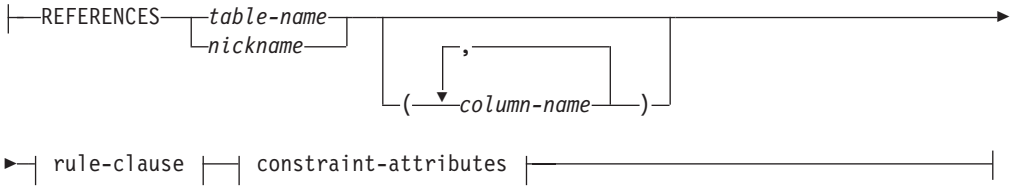
datalink-options:



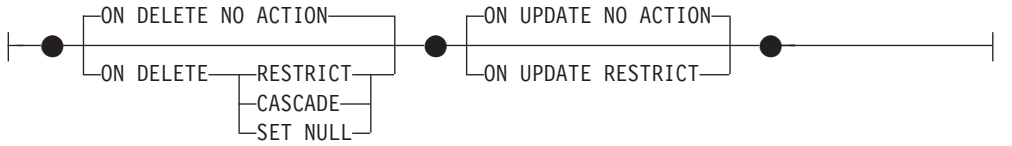
file-link-options:



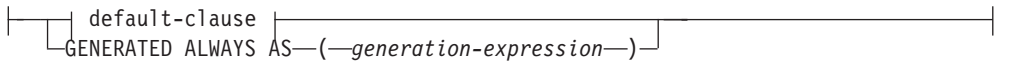
references-clause:



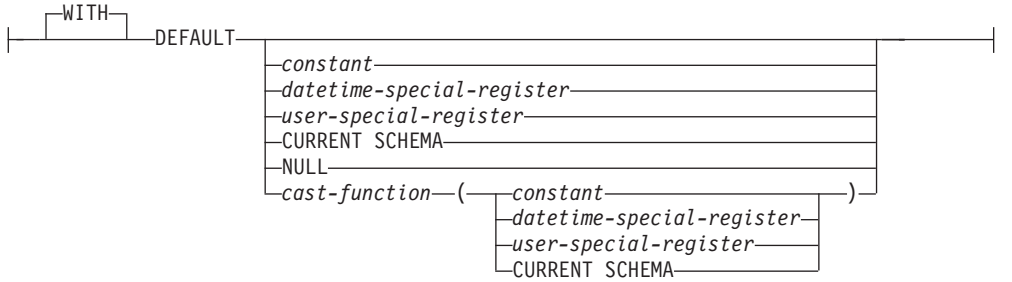
rule-clause:



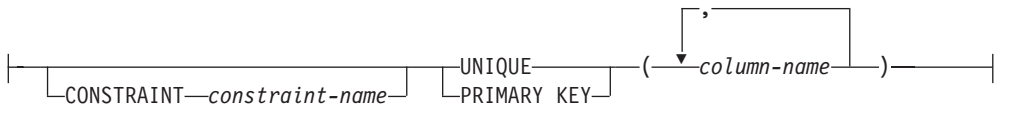
generated-column-spec:



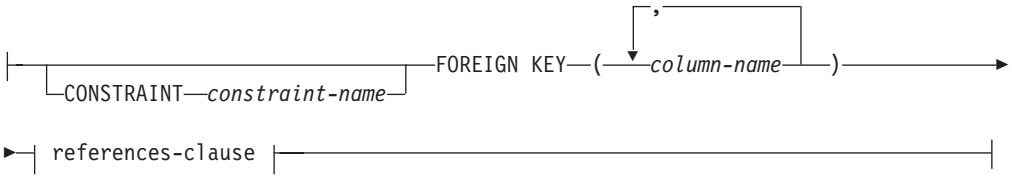
default-clause:



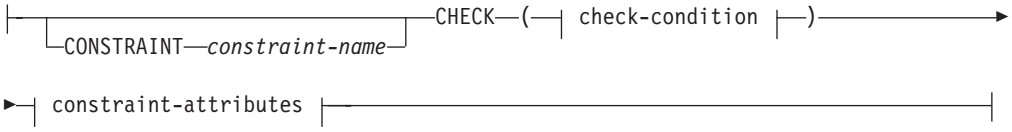
unique-constraint:



referential-constraint:

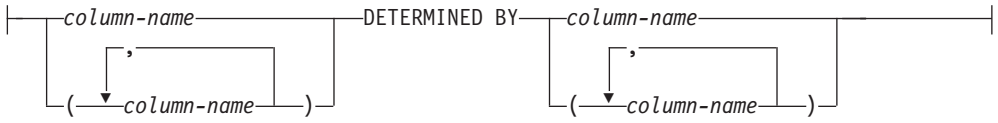


check-constraint:

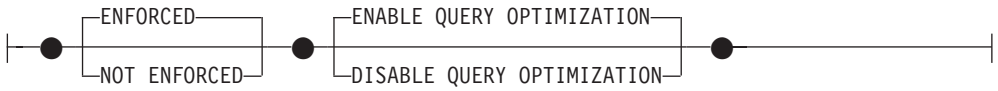


check-condition:

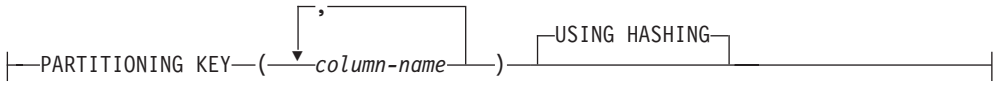
functional-dependency:



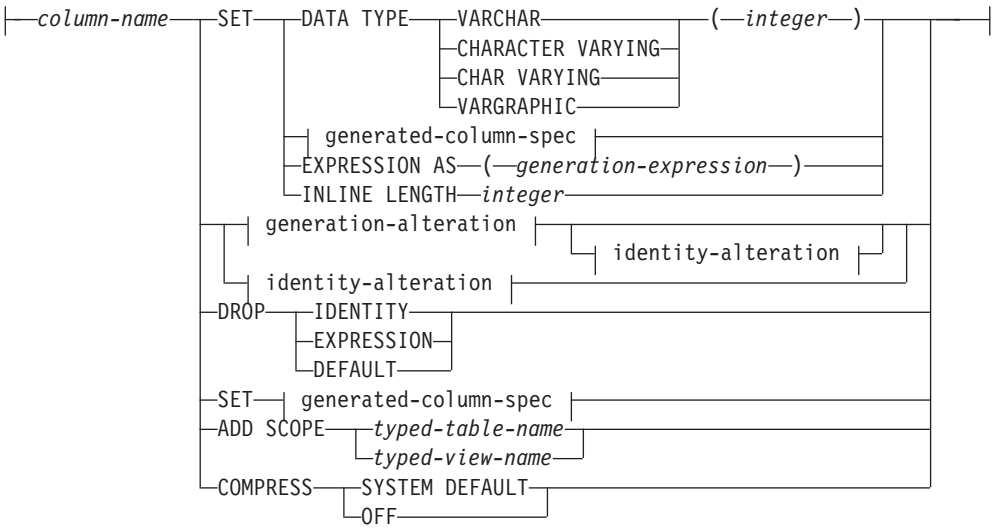
constraint-attributes:



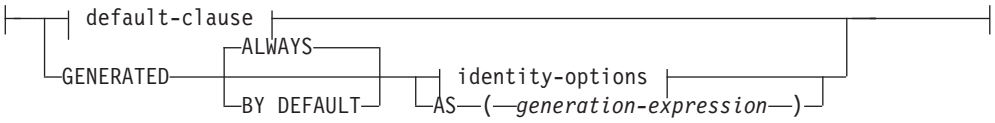
partitioning-key-definition:



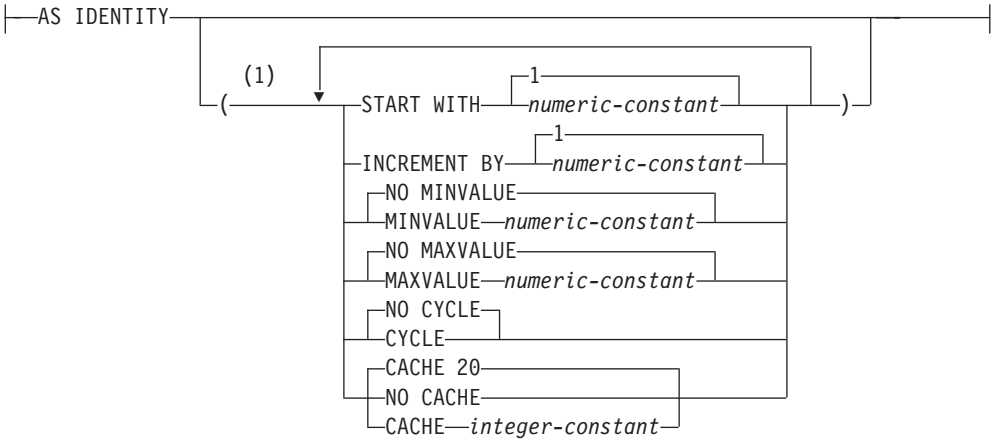
column-alteration:



generated-column-spec:

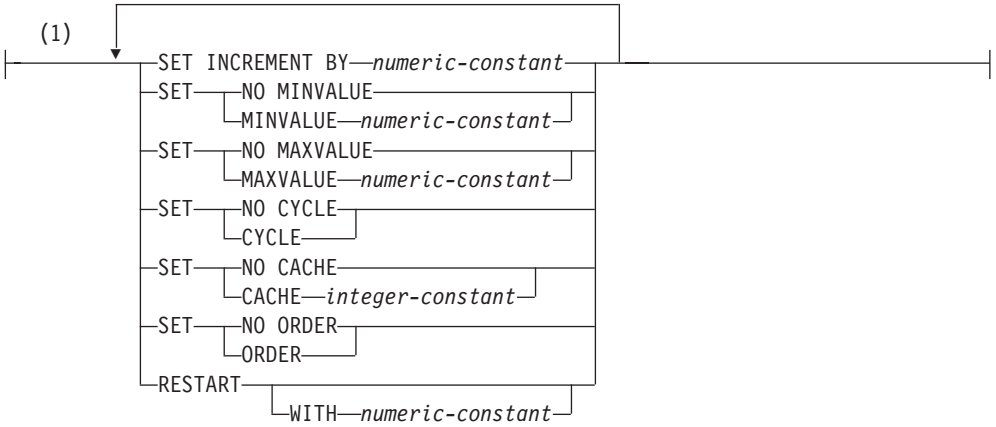


identity-options:

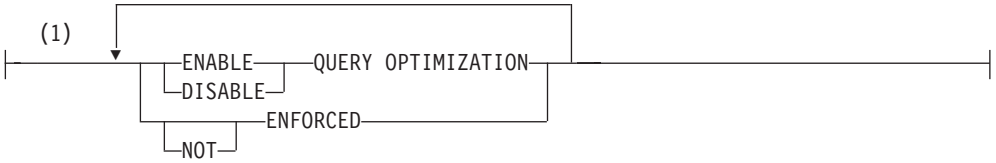


generation-alteration:

identity-alteration:



constraint-alteration:



Notes:

1 The same clause must not be specified more than once.

Description:

table-name
The *table-name* must identify a table that exists at the current server. It cannot be a nickname (SQLSTATE 42809) and must not be a view, a catalog table, or a declared temporary table (SQLSTATE 42995).

If *table-name* identifies a materialized query table, alterations are limited to adding or dropping the materialized query table, activating not logged initially, adding or dropping RESTRICT ON DROP, and changing pctfree, locksize, append, or volatile.

If *table-name* identifies a range-clustered table, alterations are limited to adding, changing, or dropping constraints, activating not logged initially, adding or dropping RESTRICT ON DROP, changing locksize, data capture, or volatile, and setting column default values.

ADD *column-definition*
Adds a column to the table. The table must not be a typed table (SQLSTATE 428DH). For all existing rows in the table, the value of the new column is set to its default value. The new column is the last column of the table; that is, if initially there are *n* columns, the added column is column *n*+1.

Adding the new column must not make the total byte count of all columns exceed the maximum record size.

column-name
Is the name of the column to be added to the table. The name cannot be qualified. Existing column names in the table cannot be used (SQLSTATE 42711).

data-type
Is one of the data types listed under "CREATE TABLE".

NOT NULL
Prevents the column from containing null values. The *default-clause* must also be specified (SQLSTATE 42601).

lob-options
Specifies options for LOB data types. See *lob-options* in "CREATE TABLE".

datalink-options
Specifies options for DATALINK data types. See *datalink-options* in "CREATE TABLE".

SCOPE
Specify a scope for a reference type column.

typed-table-name2
The name of a typed table. The data type of *column-name* must be REF(*S*), where *S* is the type of *typed-table-name2* (SQLSTATE 428DM). No checking is done of the default value for *column-name* to ensure that the value actually references an existing row in *typed-table-name2*.

typed-view-name2
The name of a typed view. The data type of *column-name* must be REF(*S*), where *S* is the type of *typed-view-name2* (SQLSTATE 428DM). No checking is done of the default value for *column-name* to ensure that the values actually references an existing row in *typed-view-name2*.

CONSTRAINT *constraint-name*
Names the constraint. A *constraint-name* must not identify a constraint that was already specified within the same ALTER TABLE statement, or as the name of any other existing constraint on the table (SQLSTATE 42710).

If the constraint name is not specified by the user, an 18-character identifier unique within the identifiers of the existing constraints defined on the table is generated by the system. (The identifier consists of ″SQL″ followed by a sequence of 15 numeric characters that are generated by a timestamp-based function.)

When used with a PRIMARY KEY or UNIQUE constraint, the *constraint-name* may be used as the name of an index that is created to support the constraint. See "Notes" on page 551 for details on index names associated with unique constraints.

PRIMARY KEY
This provides a shorthand method of defining a primary key composed of a single column. Thus, if PRIMARY KEY is specified in the definition of column C, the effect is the same as if the PRIMARY KEY(C) clause were specified as a separate clause. The column cannot contain null values, so the NOT NULL attribute must also be specified (SQLSTATE 42831).

See PRIMARY KEY within the description of the *unique-constraint* below.

UNIQUE
This provides a shorthand method of defining a unique key composed of a single column. Thus, if UNIQUE is specified in the definition of column C, the effect is the same as if the UNIQUE(C) clause were specified as a separate clause.

See UNIQUE within the description of the *unique-constraint* below.

references-clause
This provides a shorthand method of defining a foreign key composed of a single column. Thus, if a references-clause is specified in the definition of column C, the effect is the same as if that references-clause were specified as part of a FOREIGN KEY clause in which C is the only identified column.

See *references-clause* in "CREATE TABLE".

CHECK (*check-condition***)**
This provides a shorthand method of defining a check constraint that applies to a single column. See *check-condition* in "CREATE TABLE".

generated-column-spec
For details on column generation, see "CREATE TABLE".

default-clause
Specifies a default value for the column.

WITH
An optional keyword.

DEFAULT
Provides a default value in the event a value is not supplied on INSERT or is specified as DEFAULT on INSERT or UPDATE. If a specific default value is not specified following the DEFAULT keyword, the default value depends on the data type of the column as shown in Table 57. If a column is defined as a DATALINK or structured type, then a DEFAULT clause cannot be specified.

If a column is defined using a distinct type, then the default value of the column is the default value of the source data type cast to the distinct type.

Table 57. Default Values (when no value specified)

Data Type	Default Value
Numeric	0
Fixed-length character string	Blanks
Varying-length character string	A string of length 0
Fixed-length graphic string	Double-byte blanks
Varying-length graphic string	A string of length 0
Date	For existing rows, a date corresponding to January 1, 0001. For added rows, the current date.
Time	For existing rows, a time corresponding to 0 hours, 0 minutes, and 0 seconds. For added rows, the current time.
Timestamp	For existing rows, a date corresponding to January 1, 0001, and a time corresponding to 0 hours, 0 minutes, 0 seconds and 0 microseconds. For added rows, the current timestamp.
Binary string (blob)	A string of length 0

Omission of DEFAULT from a *column-definition* results in the use of the null value as the default for the column.

Specific types of values that can be specified with the DEFAULT keyword are as follows.

constant
Specifies the constant as the default value for the column. The specified constant must:

- represent a value that could be assigned to the column in accordance with the rules of assignment as described in Chapter 3
- not be a floating-point constant unless the column is defined with a floating-point data type
- not have non-zero digits beyond the scale of the column data type if the constant is a decimal constant (for example, 1.234 cannot be the default for a DECIMAL(5,2) column)
- be expressed with no more than 254 characters including the quote characters, any introducer character such as the X for a hexadecimal constant, and characters from the fully qualified function name and parentheses when the constant is the argument of a *cast-function*.

datetime-special-register
Specifies the value of the datetime special register (CURRENT DATE, CURRENT TIME, or CURRENT TIMESTAMP) at the time of INSERT, UPDATE, or LOAD as the default for the column. The data type of the column must be the data type that corresponds to the special register specified (for example, data type must be DATE when CURRENT DATE is specified). For existing rows, the value is the current date, current time or current timestamp when the ALTER TABLE statement is processed.

user-special-register
Specifies the value of the user special register (CURRENT USER, SESSION_USER, SYSTEM_USER) at the time of INSERT, UPDATE, or LOAD as the default for the column. The data type of the column must be a character string with a length not less than the length attribute of a user special register. Note that USER can be specified in place of SESSION_USER and CURRENT_USER can be specified in place of CURRENT USER. For existing rows, the value is the CURRENT USER, SESSION_USER, or SYSTEM_USER of the ALTER TABLE statement.

CURRENT SCHEMA
Specifies the value of the CURRENT SCHEMA special register at the time of INSERT, UPDATE, or LOAD as the default for the column. If CURRENT SCHEMA is specified, the data type of the column must be a character string with a length greater than or equal to the length attribute of the CURRENT SCHEMA special register. For existing rows, the value of the CURRENT SCHEMA special register at the time the ALTER TABLE statement is processed.

NULL
Specifies NULL as the default for the column. If NOT NULL was specified, DEFAULT NULL must not be specified within the same column definition.

cast-function
This form of a default value can only be used with columns defined as a distinct type, BLOB or datetime (DATE, TIME or TIMESTAMP) data type. For distinct type, with the exception of distinct types based on BLOB or datetime types, the name of the function must match the name of the distinct type for the column. If qualified with a schema name, it must be the same as the schema name for the distinct type. If not qualified, the schema name from function resolution must be the same as the schema name for the distinct type. For a distinct type based on a datetime type, where the default value is a constant, a function must be used and the name of the function must match the name of the source type of the distinct type with an implicit or explicit schema name of SYSIBM. For other datetime columns, the corresponding datetime function may also be used. For a BLOB or a distinct type based on BLOB, a function must be used and the name of the function must be BLOB with an implicit or explicit schema name of SYSIBM.

constant
Specifies a constant as the argument. The constant must conform to the rules of a constant for the source type of the distinct type or for the data type if not a distinct type. If the cast-function is BLOB, the constant must be a string constant.

datetime-special-register
Specifies CURRENT DATE, CURRENT TIME, or CURRENT TIMESTAMP. The source type of the distinct type of the column must be the data type that corresponds to the specified special register.

user-special-register
Specifies CURRENT USER, SESSION_USER, or SYSTEM_USER. The data type of the source type of the distinct type of the column must be a string data type with a length of at least 8 bytes. If the *cast-function* is BLOB, the length attribute must be at least 8 bytes.

CURRENT SCHEMA
Specifies the value of the CURRENT SCHEMA special register. The data type of the source type of the distinct type of the column must be a character string with a length greater than or equal to the length attribute of the CURRENT SCHEMA special register. If the cast-function is BLOB, the length attribute must be at least 8 bytes.

If the value specified is not valid, an error (SQLSTATE 42894) is returned.

GENERATED
Specifies that DB2 generates values for the column.

ALWAYS
Specifies that DB2 will always generate a value for the column when a row is inserted into the table, or whenever the result value of the *generation-expression* might change. The result of the expression is stored in the table. GENERATED ALWAYS is the recommended option

unless data propagation or unload and reload operations are being performed. GENERATED ALWAYS is the required option for generated columns.

BY DEFAULT
Specifies that DB2 will generate a value for the column when a row is inserted into the table, or updated, specifying DEFAULT for the column, unless an explicit value is specified. BY DEFAULT is the recommended option when using data propagation or performing unload and reload operations.

identity-options
This clause cannot be specified when adding a column to an existing table.

AS (*generation-expression***)**
Specifies that the definition of the column is based on an expression. Requires that the table be put in check pending state, using the SET INTEGRITY statement. After the ALTER TABLE statement, the SET INTEGRITY statement with FORCE GENERATED must be used to update and check all the values in that column against the new expression. For details on specifying a column with a *generation-expression*, see "CREATE TABLE".

COMPRESS SYSTEM DEFAULT
Specifies that system default values (that is, the default values used for the data types when no specific values are specified) are to be stored using minimal space. If the VALUE COMPRESSION clause is not specified, a warning is returned (SQLSTATE 01648) and system default values are not stored using minimal space.

Allowing system default values to be stored in this manner causes a slight performance penalty during insert and update operations on the column because of extra checking that is done.

The base data type must not be DATE, TIME, or TIMESTAMP (SQLSTATE 42842). If the base data type is a varying-length string, this clause is ignored. String values of length 0 are automatically compressed if a table has been set with VALUE COMPRESSION.

ADD *unique-constraint*
Defines a unique or primary key constraint. A primary key or unique constraint cannot be added to a table that is a subtable (SQLSTATE 429B3). If the table is a supertable at the top of the hierarchy, the constraint applies to the table and all its subtables.

CONSTRAINT *constraint-name*
Names the primary key or unique constraint. For more information, see *constraint-name* in "CREATE TABLE".

UNIQUE (*column-name***...,)**
Defines a unique key composed of the identified columns. The identified columns must be defined as NOT NULL. Each *column-name* must identify a column of the table and the same column must not be identified more than once. The name cannot be qualified. The number of identified columns must not exceed 16, and the sum of their stored lengths must not exceed 1024. No LOB, LONG VARCHAR, LONG VARGRAPHIC, DATALINK, distinct type based on any of these types, or structured type may be used as part of a unique key, even if the length attribute of the column is small enough to fit within the 1024-byte limit (SQLSTATE 54008). The set of columns in the unique key cannot be the same as the set of columns of the

primary key or another unique key (SQLSTATE 01543). (If LANGLEVEL is SQL92E or MIA, an error is returned, SQLSTATE 42891.) Any existing values in the set of identified columns must be unique (SQLSTATE 23515).

A check is performed to determine if an existing index matches the unique key definition (ignoring any INCLUDE columns in the index). An index definition matches if it identifies the same set of columns without regard to the order of the columns or the direction (ASC/DESC) specifications. If a matching index definition is found, the description of the index is changed to indicate that it is required by the system and it is changed to unique (after ensuring uniqueness) if it was a non-unique index. If the table has more than one matching index, an existing unique index is selected (the selection is arbitrary). If no matching index is found, a unique index will automatically be created for the columns, as described in CREATE TABLE. See "Notes" on page 551 for details on index names associated with unique constraints.

PRIMARY KEY *...(column-name,)*
Defines a primary key composed of the identified columns. Each *column-name* must identify a column of the table, and the same column must not be identified more than once. The name cannot be qualified. The number of identified columns must not exceed 16 and the sum of their stored lengths must not exceed 1024. The table must not have a primary key and the identified columns must be defined as NOT NULL. No LOB, LONG VARCHAR, LONG VARGRAPHIC, DATALINK, distinct type based on any of these types, or structured type may be used as part of a primary key, even if the length attribute of the column is small enough to fit within the 1024-byte limit (SQLSTATE 54008). The set of columns in the primary key cannot be the same as the set of columns in a unique key (SQLSTATE 01543). (If LANGLEVEL is SQL92E or MIA, an error is returned, SQLSTATE 42891.) Any existing values in the set of identified columns must be unique (SQLSTATE 23515).

A check is performed to determine if an existing index matches the primary key definition (ignoring any INCLUDE columns in the index). An index definition matches if it identifies the same set of columns without regard to the order of the columns or the direction (ASC/DESC) specifications. If a matching index definition is found, the description of the index is changed to indicate that it is the primary index, as required by the system, and it is changed to unique (after ensuring uniqueness) if it was a non-unique index. If the table has more than one matching index, an existing unique index is selected (the selection is arbitrary). If no matching index is found, a unique index will automatically be created for the columns, as described in CREATE TABLE. See "Notes" on page 551 for details on index names associated with unique constraints.

Only one primary key can be defined on a table.

ADD *referential-constraint*
Defines a referential constraint. See *referential-constraint* in "CREATE TABLE".

ADD *check-constraint*
Defines a check constraint or functional dependency. See *check-constraint* in "CREATE TABLE".

ADD *partitioning-key-definition*
Defines a partitioning key. The table must be defined in a table space on a single-partition database partition group and must not already have a

partitioning key. If a partitioning key already exists for the table, the existing key must be dropped before adding the new partitioning key.

A partitioning key cannot be added to a table that is a subtable (SQLSTATE 428DH).

PARTITIONING KEY (*column-name***...)**
Defines a partitioning key using the specified columns. Each *column-name* must identify a column of the table, and the same column must not be identified more than once. The name cannot be qualified. A column cannot be used as part of a partitioning key if the data type of the column is a LONG VARCHAR, LONG VARGRAPHIC, BLOB, CLOB, DBCLOB, DATALINK, distinct type on any of these types, or structured type.

USING HASHING
Specifies the use of the hashing function as the partitioning method for data distribution. This is the only partitioning method supported.

ADD RESTRICT ON DROP
Specifies that the table cannot be dropped, and that the table space that contains the table cannot be dropped.

ADD MATERIALIZED QUERY

materialized-query-definition
Changes a regular table to a materialized query table for use during query optimization. The table specified by *table-name* must not:

- be previously defined as a materialized query table
- be a typed table
- have any constraints, unique indexes, or triggers defined
- be referenced in the definition of another materialized query table.

If *table-name* does not meet these criteria, an error is returned (SQLSTATE 428EW).

fullselect
Defines the query in which the table is based. The columns of the existing table must:

- have the same number of columns
- have exactly the same data types
- have the same column names in the same ordinal positions

as the result columns of *fullselect* (SQLSTATE 428EW). For details about specifying the *fullselect* for a materialized query table, see "CREATE TABLE". One additional restriction is that *table-name* cannot be directly or indirectly referenced in the fullselect.

refreshable-table-options
Specifies the refreshable options for altering a materialized query table.

DATA INITIALLY DEFERRED
The data in the table must be validated using the REFRESH TABLE or SET INTEGRITY statement.

REFRESH
Indicates how the data in the table is maintained.

DEFERRED
The data in the table can be refreshed at any time using the REFRESH TABLE statement. The data in the table only reflects the result of the query as a snapshot at the time the REFRESH

TABLE statement is processed. Materialized query tables defined with this attribute do not allow INSERT, UPDATE, or DELETE statements (SQLSTATE 42807).

IMMEDIATE
The changes made to the underlying tables as part of a DELETE, INSERT, or UPDATE are cascaded to the materialized query table. In this case, the content of the table, at any point-in-time, is the same as if the specified subselect is processed. Materialized query tables defined with this attribute do not allow INSERT, UPDATE, or DELETE statements (SQLSTATE 42807).

ENABLE QUERY OPTIMIZATION
The materialized query table can be used for query optimization.

DISABLE QUERY OPTIMIZATION
The materialized query table will not be used for query optimization. The table can still be queried directly.

MAINTAINED BY
Specifies whether the data in the materialized query table is maintained by the system, user, or replication tool.

SYSTEM
Specifies that the data in the materialized query table is maintained by the system.

USER
Specifies that the data in the materialized query table is maintained by the user. The user is allowed to perform update, delete, or insert operations against user-maintained materialized query tables. The REFRESH TABLE statement, used for system-maintained materialized query tables, cannot be invoked against user-maintained materialized query tables. Only a REFRESH DEFERRED materialized query table can be defined as MAINTAINED BY USER.

FEDERATED_TOOL
Specifies that the data in the materialized query table is maintained by the replication tool. The REFRESH TABLE statement, used for system-maintained materialized query tables, cannot be invoked against federated_tool-maintained materialized query tables. Only a REFRESH DEFERRED materialized query table can be defined as MAINTAINED BY FEDERATED_TOOL.

ALTER FOREIGN KEY *constraint-name*
Alters the constraint attributes of the referential constraint *constraint-name*. The *constraint-name* must identify an existing referential constraint (SQLSTATE 42704).

ALTER CHECK *constraint-name*
Alters the constraint attributes of the check constraint or functional dependency *constraint-name*. The *constraint-name* must identify an existing check constraint or functional dependency (SQLSTATE 42704).

constraint-alteration
Options for changing attributes associated with referential or check constraints.

ENFORCED or **NOT ENFORCED**
Specifies whether the constraint is enforced by the database manager during normal operations such as insert, update, or delete.

ENFORCED
Change the constraint to ENFORCED. ENFORCED cannot be specified for a functional dependency (SQLSTATE 42621).

NOT ENFORCED
Change the constraint to NOT ENFORCED. This should only be specified if the table data is independently known to conform to the constraint.

ENABLE QUERY OPTIMIZATION or **DISABLE QUERY OPTIMIZATION**
Specifies whether the constraint or functional dependency can be used for query optimization under appropriate circumstances.

ENABLE QUERY OPTIMIZATION
The constraint is assumed to be true and can be used for query optimization.

DISABLE QUERY OPTIMIZATION
The constraint cannot be used for query optimization.

ALTER *column-alteration*
Alters the definition of a column. Only the specified attributes will be altered; others will remain unchanged.

column-name
Specifies the name of the column that is to be altered. The *column-name* must identify an existing column of the table (SQLSTATE 42703). The name cannot be qualified. The name must not identify a column that is being added, dropped, or altered in the same ALTER TABLE statement (SQLSTATE 42711).

SET DATA TYPE
Sets the data type of a column. The table must not be a typed table (SQLSTATE 428DH).

Altering the column must not make the total byte count of all columns exceed the maximum record size (SQLSTATE 54010). If the column is used in a unique constraint or an index, the new length must not cause the sum of the stored lengths for the unique constraint or index to exceed 1024 (SQLSTATE 54008).

VARCHAR *integer*
Increases the length of an existing VARCHAR column. CHARACTER VARYING or CHAR VARYING can be used as synonyms for the VARCHAR keyword. The data type of *column-name* must be VARCHAR, and the current maximum length defined for the column must not be greater than the value for *integer* (SQLSTATE 42837). The value of *integer* can range up to 32 672.

VARGRAPHIC *integer*
Increases the length of an existing VARGRAPHIC column. The data type of *column-name* must be VARGRAPHIC, and the current maximum length defined for the column must not be greater than the value for *integer* (SQLSTATE 42837). The value of *integer* can range up to 16 336.

SET EXPRESSION AS (*generation-expression***)**
Changes the expression for the column to the specified

generation-expression. SET EXPRESSION AS requires the table to be put in check pending state, using the SET INTEGRITY statement. After the ALTER TABLE statement, the SET INTEGRITY statement must be used to update and check all the values in that column against the new expression. The column must already be defined as a generated column based on an expression (SQLSTATE 42837), and must not have appeared in the DIMENSIONS clause of the table (SQLSTATE 42997). The generation-expression must conform to the same rules that apply when defining a generated column. The result data type of the generation-expression must be assignable to the data type of the column (SQLSTATE 42821).

SET *generated-column-spec*
Specifies the technique used to generate a value for the column. This can be in the form of a specific default value, an expression, or defining the column as an identity column. If an existing default for the column results from a different generation technique, that default must be dropped, which can be done in the same *column-alteration* using one of the DROP clauses.

default-clause
Specifies a new default value for the column that is to be altered. The column must not already be defined as the identity column or have a generation expression defined (SQLSTATE 42837). The specified default value must represent a value that could be assigned to the column in accordance with the rules for assignment as described in "Assignments and comparisons". Altering the default value does not change the value that is associated with this column for existing rows.

GENERATED ALWAYS or **GENERATED BY DEFAULT**
Specifies when the database manager is to generate values for the column. GENERATED BY DEFAULT specifies that a value is only to be generated when a value is not provided, or the DEFAULT keyword is used in an assignment to the column. GENERATED ALWAYS specifies that the database manager is to always generate a value for the column. GENERATED BY DEFAULT cannot be specified with a *generation-expression*.

identity-options
Specifies that the column is the identity column for the table. The column must not already be defined as the identity column, cannot have a generation expression, or cannot have an explicit default (SQLSTATE 42837). A table can only have a single identity column (SQLSTATE 428C1). The column must be specified as not nullable (SQLSTATE 42997), and the data type associated with the column must be an exact numeric data type with a scale of zero (SQLSTATE 42815). An exact numeric data type is one of: SMALLINT, INTEGER, BIGINT, DECIMAL, or NUMERIC with a scale of zero, or a distinct type based on one of these types. For details on identity options, see "CREATE TABLE".

AS (*generation-expression***)**
Specifies that the definition of the column is based on an expression. The column must not already be defined with a generation expression, cannot be the identity column, or cannot have an explicit default (SQLSTATE 42837). The *generation-expression* must conform to the same rules that apply when defining a generated column. The result data type of the *generation-expression* must be assignable to the data type of the

column (SQLSTATE 42821). The column must not be referenced in the partitioning key column or in the ORGANIZE BY clause (SQLSTATE 42997).

SET INLINE LENGTH *integer*
Changes the inline length of an existing structured type column. The inline length indicates the maximum byte size of an instance of a structured type to store inline with the rest of the values in the row. Instances of structured types that cannot be stored inline are stored separately from the base table row, similar to the way that LOB values are handled.

The data type of *column-name* must be a structured type (SQLSTATE 42842).

The default inline length for a structured-type column is the inline length of its type (specified explicitly or by default in the CREATE TYPE statement). If the inline length of a structured type is less than 292, the value 292 is used for the inline length of the column.

The explicit inline length value can only be increased (SQLSTATE -1); must be at least 292; and cannot exceed 32672 (SQLSTATE 54010).

Altering the column must not make the total byte count of all columns exceed the maximum record size (SQLSTATE 54010).

Data that is already stored separately from the rest of the row will not be moved inline by this statement. To take advantage of the altered inline length of a structured type column, invoke the REORG command against the specified table after altering the inline length of its column.

SET GENERATED ALWAYS or **GENERATED BY DEFAULT**
Specifies when the database manager is to generate values for the column. GENERATED BY DEFAULT specifies that a value is only to be generated when a value is not provided or the DEFAULT keyword is used in an assignment to the column. GENERATED ALWAYS specifies that the database manager is to always generate a value for the column. The column must already be defined as a generated column based on an identity column; that is, defined with the AS IDENTITY clause (SQLSTATE 42837).

DROP DEFAULT
Drops the current default for the column. The specified column must have a default value (SQLSTATE 42837).

DROP EXPRESSION
Drops the generated expression attributes of the column, making the column a non-generated column. DROP EXPRESSION is not allowed if the column is not a generated expression column (SQLSTATE 42837).

DROP IDENTITY
Drops the identity attributes of the column, making the column a simple numeric data type column. DROP IDENTITY is not allowed if the column is not an identity column (SQLSTATE 42837).

ADD SCOPE
Add a scope to an existing reference type column that does not already have a scope defined (SQLSTATE 428DK). If the table being altered is a typed table, the column must not be inherited from a supertable (SQLSTATE 428DJ).

typed-table-name
The name of a typed table. The data type of *column-name* must be

REF(*S*), where *S* is the type of *typed-table-name* (SQLSTATE 428DM). No checking is done of any existing values in *column-name* to ensure that the values actually reference existing rows in *typed-table-name*.

typed-view-name
The name of a typed view. The data type of *column-name* must be REF(*S*), where *S* is the type of *typed-view-name* (SQLSTATE 428DM). No checking is done of any existing values in *column-name* to ensure that the values actually reference existing rows in *typed-view-name*.

COMPRESS
Specifies whether or not default values for this column are to be stored more efficiently.

SYSTEM DEFAULT
Specifies that system default values (that is, the default values used for the data types when no specific values are specified) are to be stored using minimal space. If the table is not already set with the VALUE COMPRESSION attribute activated, a warning is returned (SQLSTATE 01648), and system default values are not stored using minimal space.

Allowing system default values to be stored in this manner causes a slight performance penalty during insert and update operations on the column because of the extra checking that is done.

Existing data in the column is not changed. Consider offline table reorganization to enable existing data to take advantage of storing system default values using minimal space.

OFF
Specifies that system default values are to be stored in the column as regular values. Existing data in the column is not changed. Offline reorganization is recommended to change existing data.

The base data type must not be DATE, TIME or TIMESTAMP (SQLSTATE 42842). If the base data type is a varying-length string, this clause is ignored. String values of length 0 are automatically compressed if a table has been set with VALUE COMPRESSION.

If the table being altered is a typed table, the column must not be inherited from a supertable (SQLSTATE 428DJ).

identity-alteration
Alters the identity attributes of the column. The column must be an identity column.

SET INCREMENT BY *numeric-constant*
Specifies the interval between consecutive values of the identity column. The next value to be generated for the identity column will be determined from the last assigned value with the increment applied. The column must already be defined with the IDENTITY attribute (SQLSTATE 42837).

This value can be any positive or negative value that could be assigned to this column (SQLSTATE 42815), and does not exceed the value of a large integer constant (SQLSTATE 42820), without non-zero digits existing to the right of the decimal point (SQLSTATE 428FA).

If this value is negative, this is a descending sequence after the ALTER statement. If this value is 0 or positive, this is an ascending sequence after the ALTER statement.

SET NO MINVALUE or **MINVALUE** *numeric-constant*
Specifies the minimum value at which a descending identity column either cycles or stops generating values, or the value to which an ascending identity column cycles after reaching the maximum value. The column must exist in the specified table (SQLSTATE 42703), and must already be defined with the IDENTITY attribute (SQLSTATE 42837).

NO MINVALUE
For an ascending sequence, the value is the original starting value. For a descending sequence, the value is the minimum value of the data type of the column.

MINVALUE *numeric-constant*
Specifies the numeric constant that is the minimum value. This value can be any positive or negative value that could be assigned to this column (SQLSTATE 42815), without non-zero digits existing to the right of the decimal point (SQLSTATE 428FA), but the value must be less than or equal to the maximum value (SQLSTATE 42815).

SET NO MAXVALUE or **MAXVALUE** *numeric-constant*
Specifies the maximum value at which an ascending identity column either cycles or stops generating values, or the value to which a descending identity column cycles after reaching the minimum value. The column must exist in the specified table (SQLSTATE 42703), and must already be defined with the IDENTITY attribute (SQLSTATE 42837).

NO MAXVALUE
For an ascending sequence, the value is the maximum value of the data type of the column. For a descending sequence, the value is the original starting value.

MAXVALUE *numeric-constant*
Specifies the numeric constant that is the maximum value. This value can be any positive or negative value that could be assigned to this column (SQLSTATE 42815), without non-zero digits existing to the right of the decimal point (SQLSTATE 428FA), but the value must be greater than or equal to the minimum value (SQLSTATE 42815).

SET NO CYCLE or **CYCLE**
Specifies whether this identity column should continue to generate values after generating either its maximum or minimum value. The column must exist in the specified table (SQLSTATE 42703), and must already be defined with the IDENTITY attribute (SQLSTATE 42837).

NO CYCLE
Specifies that values will not be generated for the identity column once the maximum or minimum value has been reached.

CYCLE
Specifies that values continue to be generated for this column after the maximum or minimum value has been reached. If this option is used, then after an ascending identity column reaches the maximum value, it generates its minimum value; or after a descending sequence reaches the minimum value, it generates its maximum value. The maximum and minimum values for the identity column determine the range that is used for cycling.

When CYCLE is in effect, duplicate values can be generated for an identity column. Although not required, if unique values are desired, a single-column unique index defined using the identity column will ensure uniqueness. If a unique index exists on such an identity column and a non-unique value is generated, an error occurs (SQLSTATE 23505).

SET NO CACHE or **CACHE** *integer-constant*
Specifies whether to keep some pre-allocated values in memory for faster access. This is a performance and tuning option. The column must already be defined with the IDENTITY attribute (SQLSTATE 42837).

NO CACHE
Specifies that values for the identity column are not to be pre-allocated. In a data sharing environment, if the identity values *must* be generated in order of request, the NO CACHE option must be used.

When this option is specified, the values of the identity column are not stored in the cache. In this case, every request for a new identity value results in synchronous I/O to the log.

CACHE *integer-constant*
Specifies how many values of the identity sequence are pre-allocated and kept in memory. When values are generated for the identity column, pre-allocating and storing values in the cache reduces synchronous I/O to the log.

If a new value is needed for the identity column and there are no unused values available in the cache, the allocation of the value requires waiting for I/O to the log. However, when a new value is needed for the identity column and there is an unused value in the cache, the allocation of that identity value can happen more quickly by avoiding the I/O to the log.

In the event of a database deactivation, either normally or due to a system failure, all cached sequence values that have not been used in committed statements are lost (that is, they will never be used). The value specified for the CACHE option is the maximum number of values for the identity column that could be lost in case of system failure.

The minimum value is 2 (SQLSTATE 42815).

SET NO ORDER or **ORDER**
Specifies whether the identity column values must be generated in order of request. The column must exist in the specified table (SQLSTATE 42703), and must already be defined with the IDENTITY attribute (SQLSTATE 42837).

NO ORDER
Specifies that the identity column values do not need to be generated in order of request.

ORDER
Specifies that the identity column values must be generated in order of request.

RESTART or **RESTART WITH** *numeric-constant*
Resets the state of the sequence associated with the identity column. If

WITH *numeric-constant* is not specified, the sequence for the identity column is restarted at the value that was specified, either implicitly or explicitly, as the starting value when the identity column was originally created.

The column must exist in the specified table (SQLSTATE 42703), and must already be defined with the IDENTITY attribute (SQLSTATE 42837). RESTART does *not* change the original START WITH value.

The *numeric-constant* is an exact numeric constant that can be any positive or negative value that could be assigned to this column (SQLSTATE 42815), without non-zero digits existing to the right of the decimal point (SQLSTATE 428FA). The *numeric-constant* will be used as the next value for the column.

DROP PRIMARY KEY
Drops the definition of the primary key and all referential constraints dependent on this primary key. The table must have a primary key.

DROP FOREIGN KEY *constraint-name*
Drops the referential constraint *constraint-name*. The *constraint-name* must identify a referential constraint. For information on implications of dropping a referential constraint see "Notes" on page 551.

DROP UNIQUE *constraint-name*
Drops the definition of the unique constraint *constraint-name* and all referential constraints dependent on this unique constraint. The *constraint-name* must identify an existing UNIQUE constraint. For information on implications of dropping a unique constraint, see "Notes" on page 551.

DROP CHECK *constraint-name*
Drops the check constraint *constraint-name*. The *constraint-name* must identify an existing check constraint defined on the table.

DROP CONSTRAINT *constraint-name*
Drops the constraint *constraint-name*. The *constraint-name* must identify an existing check constraint, referential constraint, primary key or unique constraint defined on the table. For information on implications of dropping a constraint, see "Notes" on page 551.

DROP PARTITIONING KEY
Drops the partitioning key. The table must have a partitioning key and must be in a table space defined on a single-partition database partition group.

DROP RESTRICT ON DROP
Removes the restriction on dropping the table, and the table space that contains the table.

DROP MATERIALIZED QUERY
Changes a materialized query table so that it is no longer considered a materialized query table. The table specified by *table-name* must be defined as a materialized query table that is not replicated (SQLSTATE 428EW). The definition of the columns of *table-name* is not changed but the table can no longer be used for query optimization and the REFRESH TABLE statement can no longer be used.

DATA CAPTURE
Indicates whether extra information for data replication is to be written to the log.

If the table is a typed table, then this option is not supported (SQLSTATE 428DH for root tables or 428DR for other subtables).

NONE
Indicates that no extra information will be logged.

CHANGES
Indicates that extra information regarding SQL changes to this table will be written to the log. This option is required if this table will be replicated and the Capture program is used to capture changes for this table from the log.

If the table is defined to allow data on a partition other than the catalog partition (multiple partition database partition group or database partition group with partition other than the catalog partition), then this option is not supported (SQLSTATE 42997).

If the schema name (implicit or explicit) of the table is longer than 18 bytes, this option is not supported (SQLSTATE 42997).

INCLUDE LONGVAR COLUMNS
Allows data replication utilities to capture changes made to LONG VARCHAR or LONG VARGRAPHIC columns. The clause may be specified for tables that do not have any LONG VARCHAR or LONG VARGRAPHIC columns since it is possible to ALTER the table to include such columns.

ACTIVATE NOT LOGGED INITIALLY
Activates the NOT LOGGED INITIALLY attribute of the table for this current unit of work.

Any changes made to the table by an INSERT, DELETE, UPDATE, CREATE INDEX, DROP INDEX, or ALTER TABLE in the same unit of work after the table is altered by this statement are not logged. Any changes made to the system catalog by the ALTER statement in which the NOT LOGGED INITIALLY attribute is activated are logged. Any subsequent changes made in the same unit of work to the system catalog information are logged.

At the completion of the current unit of work, the NOT LOGGED INITIALLY attribute is deactivated and all operations that are done on the table in subsequent units of work are logged.

If using this feature to avoid locks on the catalog tables while inserting data, it is important that only this clause be specified on the ALTER TABLE statement. Use of any other clause in the ALTER TABLE statement will result in catalog locks. If no other clauses are specified for the ALTER TABLE statement, then only a SHARE lock will be acquired on the system catalog tables. This can greatly reduce the possibility of concurrency conflicts for the duration of time between when this statement is executed and when the unit of work in which it was executed is ended.

If the table is a typed table, this option is only supported on the root table of the typed table hierarchy (SQLSTATE 428DR).

For more information about the NOT LOGGED INITIALLY attribute, see the description of this attribute in “CREATE TABLE”.

Note: If non-logged activity occurs against a table that has the NOT LOGGED INITIALLY attribute activated, and if a statement fails (causing a rollback), or a ROLLBACK TO SAVEPOINT is executed, the entire unit of work is rolled back (SQL1476N). Furthermore, the table for which the NOT LOGGED INITIALLY attribute was activated is marked inaccessible after the rollback has occurred and can only be dropped.

Therefore, the opportunity for errors within the unit of work in which the NOT LOGGED INITIALLY attribute is activated should be minimized.

WITH EMPTY TABLE
Causes all data currently in table to be removed. Once the data has been removed, it cannot be recovered except through use of the RESTORE facility. If the unit of work in which this Alter statement was issued is rolled back, the table data will NOT be returned to its original state.

When this action is requested, no DELETE triggers defined on the affected table are fired. Any indexes that exist on the table are also emptied.

PCTFREE *integer*
Specifies the percentage of each page that is to be left as free space during a load or a table reorganization operation. The first row on each page is added without restriction. When additional rows are added to a page, at least *integer* percent of the page is left as free space. The PCTFREE value is considered only by the load and table reorg utilities. The value of *integer* can range from 0 to 99. A PCTFREE value of -1 in the catalog (SYSCAT.TABLES) is interpreted as the default value. The default PCTFREE value for a table page is 0. If the table is a typed table, this option is only supported on the root table of the typed table hierarchy (SQLSTATE 428DR).

LOCKSIZE
Indicates the size (granularity) of locks used when the table is accessed. Use of this option in the table definition will not prevent normal lock escalation from occurring. If the table is a typed table, this option is only supported on the root table of the typed table hierarchy (SQLSTATE 428DR).

ROW
Indicates the use of row locks. This is the default lock size when a table is created.

TABLE
Indicates the use of table locks. This means that the appropriate share or exclusive lock is acquired on the table and intent locks (except intent none) are not used. Use of this value may improve the performance of queries by limiting the number of locks that need to be acquired. However, concurrency is also reduced since all locks are held over the complete table.

APPEND
Indicates whether data is appended to the end of the table data or placed where free space is available in data pages. If the table is a typed table, this option is only supported on the root table of the typed table hierarchy (SQLSTATE 428DR).

ON
Indicates that table data will be appended and information about free space on pages will not be kept. The table must not have a clustered index (SQLSTATE 428CA).

OFF
Indicates that table data will be placed where there is available space. This is the default when a table is created.

The table should be reorganized after setting APPEND OFF since the information about available free space is not accurate and may result in poor performance during insert.

VOLATILE CARDINALITY or **NOT VOLATILE CARDINALITY**
Indicates to the optimizer whether or not the cardinality of table *table-name* can vary significantly at run time. Volatility applies to the number of rows in the table, not to the table itself. CARDINALITY is an optional keyword. The default is NOT VOLATILE.

VOLATILE
Specifies that the cardinality of table *table-name* can vary significantly at run time, from empty to large. To access the table, the optimizer will use an index scan (rather than a table scan, regardless of the statistics) if that index is index-only (all referenced columns are in the index), or that index is able to apply a predicate in the index scan. The list prefetch access method will not be used to access the table. If the table is a typed table, this option is only supported on the root table of the typed table hierarchy (SQLSTATE 428DR).

NOT VOLATILE
Specifies that the cardinality of *table-name* is not volatile. Access plans to this table will continue to be based on existing statistics and on the current optimization level.

VALUE COMPRESSION
Specifies whether or not NULL and 0-length data values are to be stored more efficiently for most data types. This also determines the row format that is to be used. If the table is a typed table, this option is only supported on the root table of the typed table hierarchy (SQLSTATE 428DR).

ACTIVATE
Specifies that 0-length data values for columns whose data type is BLOB, CLOB, DBCLOB, LONG VARCHAR, or LONG VARGRAPHIC are to be stored using minimal space. Each NULL value is stored without using an additional byte. The row format that is used to support this determines the byte counts for each data type, and tends to cause data fragmentation during updates. The new row format (specified for a column through the COMPRESS SYSTEM DEFAULT option) also allows system default values for the column to be stored more efficiently.

DEACTIVATE
Specifies that NULL values are to be stored with space set aside for possible future updates. This space is not set aside for varying-length columns. The row format used determines the byte counts for each data type. It also does not support efficient storage of system default values for a column. If columns already exist with the COMPRESS SYSTEM DEFAULT attribute, a warning is returned (SQLSTATE 01648).

An update operation will cause an existing row to be changed to the new row format. Offline table reorganization is recommended to improve the performance of update operations on existing rows.

LOG INDEX BUILD
Specifies the level of logging that is to be performed during create, recreate, or reorganize index operations on this table.

NULL
Specifies that the value of the *logindexbuild* database configuration parameter will be used to determine whether or not index build operations are to be completely logged. This is the default when the table is created.

OFF
Specifies that any index build operations on this table will be logged minimally. This value overrides the setting of the *logindexbuild* database configuration parameter.

ON
Specifies that any index build operations on this table will be logged completely. This value overrides the setting of the *logindexbuild* database configuration parameter.

Rules:

- Any unique or primary key constraint defined on the table must be a superset of the partitioning key, if there is one (SQLSTATE 42997).
- Primary or unique keys cannot be subsets of dimensions (SQLSTATE 429BE).
- A column can only be referenced in one ADD or ALTER COLUMN clause in a single ALTER TABLE statement (SQLSTATE 42711).
- A column length cannot be altered if the table has any materialized query tables that are dependent on the table (SQLSTATE 42997).
- The table must be set to check pending state, using the SET INTEGRITY statement (SQLSTATE 55019), before:
  - Adding a column with a generation expression
  - Altering the generated expression of a column
  - Changing a column to have a generated expression

Notes:

- Altering a table to a materialized query table will put the table in check-pending state. If the table is defined as REFRESH IMMEDIATE, the table must be taken out of check-pending state before INSERT, DELETE, or UPDATE commands can be invoked on the table referenced by the fullselect. The table can be taken out of check-pending state by using REFRESH TABLE or SET INTEGRITY, with the IMMEDIATE CHECKED option, to completely refresh the data in the table based on the fullselect. If the data in the table accurately reflects the result of the fullselect, the IMMEDIATE UNCHECKED option of SET INTEGRITY can be used to take the table out of check-pending state.
- Altering a table to change it to a REFRESH IMMEDIATE materialized query table will cause any packages with INSERT, DELETE, or UPDATE usage on the table referenced by the fullselect to be invalidated.
- Altering a table to change from a materialized query table to a regular table will cause any packages dependent on the table to be invalidated.
- Altering a table to change from a MAINTAINED BY FEDERATED_TOOL materialized query table to a regular table will not cause any change in the subscription setup of the replication tool. Because a subsequent change to a MAINTAINED BY SYSTEM materialized query table will cause the replication tool to fail, you must change the subscription setting when changing a MAINTAINED BY FEDERATED_TOOL materialized query table.
- If a deferred materialized query table is associated with a staging table, the staging table will be dropped if the materialized query table is altered to a regular table.
- ADD column clauses are processed prior to all other clauses. Other clauses are processed in the order that they are specified.
- Any columns added via ALTER TABLE will not automatically be added to any existing view of the table.

- When an index is automatically created for a unique or primary key constraint, the database manager will try to use the specified constraint name as the index name with a schema name that matches the schema name of the table. If this matches an existing index name or no name for the constraint was specified, the index is created in the SYSIBM schema with a system-generated name formed of "SQL" followed by a sequence of 15 numeric characters generated by a timestamp based function.
- Any table that may be involved in a DELETE operation on table T is said to be *delete-connected* to T. Thus, a table is delete-connected to T if it is a dependent of T or it is a dependent of a table in which deletes from T cascade.
- A package has an insert (update/delete) usage on table T if records are inserted into (updated in/deleted from) T either directly by a statement in the package, or indirectly through constraints or triggers executed by the package on behalf of one of its statements. Similarly, a package has an update usage on a column if the column is modified directly by a statement in the package, or indirectly through constraints or triggers executed by the package on behalf of one of its statements.
- In a federated system, a remote base table that was created using transparent DDL can be altered. However, transparent DDL does impose some limitations on the modifications that can be made:
  - A remote base table can only be altered by adding new columns or specifying a primary key.
  - You cannot specify a comment on an existing column in a remote base table.
  - An existing primary key in a remote base table cannot be altered or dropped.
  - Altering a remote base table invalidates any packages that are dependent on the nickname associated with that remote base table.
  - The remote data source must support the changes being requested through the ALTER TABLE statement. Depending on how the data source responds to requests it does not support, an error might be returned or the request might be ignored.
  - An attempt to alter a remote base table that was not created using transparent DDL returns an error.
- Any changes to primary key, unique keys, or foreign keys may have the following effect on packages, indexes, and other foreign keys.
  - If a primary key or unique key is added:
    - There is no effect on packages, foreign keys, or existing unique keys. (If the primary or unique key uses an existing unique index that was created in a previous version and has not been converted to support deferred uniqueness, the index is converted, and packages with update usage on the associated table are invalidated.)
  - If a primary key or unique key is dropped:
    - The index is dropped if it was automatically created for the constraint. Any packages dependent on the index are invalidated.
    - The index is set back to non-unique if it was converted to unique for the constraint and it is no longer system-required. Any packages dependent on the index are invalidated.
    - The index is set to no longer system required if it was an existing unique index used for the constraint. There is no effect on packages.
    - All dependent foreign keys are dropped. Further action is taken for each dependent foreign key, as specified in the next item.

- If a foreign key is added, dropped, or altered from NOT ENFORCED to ENFORCED (or ENFORCED to NOT ENFORCED):
  - All packages with an insert usage on the object table are invalidated.
  - All packages with an update usage on at least one column in the foreign key are invalidated.
  - All packages with a delete usage on the parent table are invalidated.
  - All packages with an update usage on at least one column in the parent key are invalidated.
- If a foreign key or a functional dependency is altered from ENABLE QUERY OPTIMIZATION to DISABLE QUERY OPTIMIZATION:
  - All packages with dependencies on the constraint for optimization purposes are invalidated.
- Adding a column to a table will result in invalidation of all packages with insert usage on the altered table. If the added column is the first user-defined structured type column in the table, packages with DELETE usage on the altered table will also be invalidated.
- Adding a check or referential constraint to a table that already exists and that is not in check pending state, or altering the existing check or referential constraint from NOT ENFORCED to ENFORCED on an existing table that is not in check pending state will cause the existing rows in the table to be immediately evaluated against the constraint. If the verification fails, an error (SQLSTATE 23512) is raised. If a table is in check pending state, adding a check or referential constraint, or altering a constraint from NOT ENFORCED to ENFORCED will not immediately lead to the enforcement of the constraint. Instead, the corresponding constraint type flags used in the check pending operation will be updated. Issue the SET INTEGRITY statement to begin enforcing the constraint.
- Adding, altering, or dropping a check constraint will result in invalidation of all packages with either an insert usage on the object table, an update usage on at least one of the columns involved in the constraint, or a select usage exploiting the constraint to improve performance.
- Adding a partitioning key will result in invalidation of all packages with an update usage on at least one of the columns of the partitioning key.
- A partitioning key that was defined by default as the first column of the primary key is not affected by dropping the primary key and adding a different primary key.
- Altering a column to increase the length will invalidate all packages that reference the table (directly or indirectly through a referential constraint or trigger) with the altered column.
- Altering a column to increase the length will regenerate views (except typed views) that are dependent on the table. If an error occurs while regenerating a view, an error is returned (SQLSTATE 56098). Any typed views that are dependent on the table are marked inoperative.
- Altering a column to increase the length may cause errors (SQLSTATE 54010) in processing triggers when a statement that would involve the trigger is prepared or bound. This may occur when row length based on the sum of the lengths of the transition variables and transition table columns is too long. If such a trigger were dropped a subsequent attempt to create it would result in an error (SQLSTATE 54040).
- VARCHAR and VARGRAPHIC columns that have been altered to be greater than 4000 and 2000 respectively should not be used as input parameters in functions in the SYSFUN schema (SQLSTATE 22001).

- Altering a structured type column to increase the inline length will invalidate all packages that reference the table, either directly or indirectly through a referential constraint or trigger.
- Altering a structured type column to increase the inline length will regenerate views that are dependent on the table.
- Changing the LOCKSIZE for a table will result in invalidation of all packages that have a dependency on the altered table.
- The ACTIVATE NOT LOGGED INITIALLY clause cannot be used when DATALINK columns with the FILE LINK CONTROL attribute are being added to the table (SQLSTATE 42613).
- Changing VOLATILE or NOT VOLATILE CARDINALITY will result in invalidation of all packages that have a dependency on the altered table.
- Replication customers should take caution when increasing the length of VARCHAR columns. The change data table associated with an application table might already be at or near the DB2 rowsize limit. The change data table should be altered before the application table, or the two should be altered within the same unit of work, to ensure that the alteration can be completed for both tables. Consideration should be given for copies, which may also be at or near the rowsize limit, or reside on platforms which lack the feature to increase the length of an existing column.

  If the change data table is not altered before the Capture program processes log records with the increased VARCHAR column length, the Capture program will likely fail. If a copy containing the VARCHAR column is not altered before the subscription maintaining the copy runs, the subscription will likely fail.
- ***Compatibilities***
  - For compatibility with previous versions of DB2:
    - The ADD keyword is optional for:
      - Unnamed PRIMARY KEY constraints
      - Unnamed referential constraints
      - Referential constraints whose name follows the phrase FOREIGN KEY
    - The CONSTRAINT keyword can be omitted from a *column-definition* defining a references-clause
    - *constraint-name* can be specified following FOREIGN KEY (without the CONSTRAINT keyword)
    - SET SUMMARY AS can be specified in place of SET MATERIALIZED QUERY AS
    - SET MATERIALIZED QUERY AS DEFINITION ONLY can be specified in place of DROP MATERIALIZED QUERY
    - SET MATERIALIZED QUERY AS (fullselect) can be specified in place of ADD MATERIALIZED QUERY (fullselect)
  - For compatibility with previous versions of DB2 and for consistency:
    - A comma can be used to separate multiple options in the *identity-alteration* clause.
  - The following syntax is also supported:
    - NOMINVALUE, NOMAXVALUE, NOCYCLE, NOCACHE, and NOORDER

Examples:

Example 1: Add a new column named RATING, which is one character long, to the DEPARTMENT table.

```
ALTER TABLE DEPARTMENT
   ADD RATING CHAR(1)
```

Example 2: Add a new column named SITE_NOTES to the PROJECT table. Create SITE_NOTES as a varying-length column with a maximum length of 1000 characters. The values of the column do not have an associated character set and therefore should not be translated.

```
ALTER TABLE PROJECT
   ADD SITE_NOTES  VARCHAR(1000) FOR BIT DATA
```

Example 3: Assume a table called EQUIPMENT exists defined with the following columns:

```
Column Name        Data Type
EQUIP_NO           INT
EQUIP_DESC         VARCHAR(50)
LOCATION           VARCHAR(50)
EQUIP_OWNER        CHAR(3)
```

Add a referential constraint to the EQUIPMENT table so that the owner (EQUIP_OWNER) must be a department number (DEPTNO) that is present in the DEPARTMENT table. DEPTNO is the primary key of the DEPARTMENT table. If a department is removed from the DEPARTMENT table, the owner (EQUIP_OWNER) values for all equipment owned by that department should become unassigned (or set to null). Give the constraint the name DEPTQUIP.

```
ALTER TABLE EQUIPMENT
     ADD CONSTRAINT DEPTQUIP
      FOREIGN KEY (EQUIP_OWNER)
        REFERENCES DEPARTMENT
           ON DELETE SET NULL
```

Also, an additional column is needed to allow the recording of the quantity associated with this equipment record. Unless otherwise specified, the EQUIP_QTY column should have a value of 1 and must never be null.

```
ALTER TABLE EQUIPMENT
  ADD COLUMN EQUIP_QTY
  SMALLINT NOT NULL DEFAULT 1
```

Example 4: Alter table EMPLOYEE. Add the check constraint named REVENUE defined so that each employee must make a total of salary and commission greater than $30,000.

```
ALTER TABLE EMPLOYEE
  ADD CONSTRAINT REVENUE
  CHECK (SALARY + COMM > 30000)
```

Example 5: Alter table EMPLOYEE. Drop the constraint REVENUE which was previously defined.

```
ALTER TABLE EMPLOYEE
  DROP CONSTRAINT REVENUE
```

Example 6: Alter a table to log SQL changes in the default format.

```
ALTER TABLE SALARY1
  DATA CAPTURE NONE
```

Example 7: Alter a table to log SQL changes in an expanded format.

```
ALTER TABLE SALARY2
  DATA CAPTURE CHANGES
```

Example 8: Alter the EMPLOYEE table to add 4 new columns with default values.

```
ALTER TABLE EMPLOYEE
   ADD COLUMN HEIGHT MEASURE   DEFAULT MEASURE(1)
   ADD COLUMN BIRTHDAY BIRTHDATE DEFAULT DATE('01-01-1850')
   ADD COLUMN FLAGS BLOB(1M)  DEFAULT BLOB(X'01')
   ADD COLUMN PHOTO PICTURE   DEFAULT BLOB(X'00')
```

The default values use various function names when specifying the default. Since MEASURE is a distinct type based on INTEGER, the MEASURE function is used. The HEIGHT column default could have been specified without the function since the source type of MEASURE is not BLOB or a datetime data type. Since BIRTHDATE is a distinct type based on DATE, the DATE function is used (BIRTHDATE cannot be used here). For the FLAGS and PHOTO columns the default is specified using the BLOB function even though PHOTO is a distinct type. To specify a default for BIRTHDAY, FLAGS and PHOTO columns, a function must be used because the type is a BLOB or a distinct type sourced on a BLOB or datetime data type.

Example 9: A table called CUSTOMERS is defined with the following columns:

```
Column Name        Data Type
BRANCH_NO          SMALLINT
CUSTOMER_NO        DECIMAL(7)
CUSTOMER_NAME      VARCHAR(50)
```

In this table, the primary key is made up of the BRANCH_NO and CUSTOMER_NO columns. To partition the table, you will need to create a partitioning key for the table. The table must be defined in a table space on a single-node database partition group. The primary key must be a superset of the partitioning columns: at least one of the columns of the primary key must be used as the partitioning key. Make BRANCH_NO the partitioning key as follows:

```
ALTER TABLE CUSTOMERS
  ADD PARTITIONING KEY (BRANCH_NO)
```

Example 10: A remote table EMPLOYEE was created in a federated system using transparent DDL. Alter the remote table EMPLOYEE to add the columns PHONE_NO and WORK_DEPT; also add a primary key on the existing column EMP_NO and the new column WORK_DEPT.

```
ALTER TABLE EMPLOYEE
  ADD COLUMN PHONE_NO CHAR(4) NOT NULL
  ADD COLUMN WORK_DEPT CHAR(3)
  ADD PRIMARY KEY (EMP_NO, WORK_DEPT)
```

Example 11: Alter the DEPARTMENT table to add a functional dependency FD1, then drop the functional dependency FD1 from the DEPARTMENT table.

```
ALTER TABLE DEPARTMENT
  ADD COSTRAINT FD1
    CHECK ( DEPTNAME DETERMINED BY DEPTNO) NOT ENFORCED

ALTER TABLE DEPARTMENT
  DROP CHECK FD1
```

Example 12: Change the default value for the WORKDEPT column in the EMPLOYEE table to 123.

```
ALTER TABLE EMPLOYEE
  ALTER COLUMN WORKDEPT
    SET DEFAULT '123'
```

Related concepts:

- "What is transparent DDL?" in the *Federated Systems Guide*

Related tasks:

- "Altering remote tables using transparent DDL" in the *Federated Systems Guide*

Related reference:

- "ALTER TYPE (Structured) statement" in the *SQL Reference, Volume 2*
- "CREATE TABLE" on page 591
- "Assignments and comparisons" in the *SQL Reference, Volume 1*
- "ALTOBJ procedure" in the *SQL Administrative Routines*

Related samples:

- "dbrecov.sqc -- How to recover a database (C)"
- "tbconstr.sqc -- How to create, use, and drop constraints (C)"
- "dbrecov.sqC -- How to recover a database (C++)"
- "dtstruct.sqC -- Create, use, drop a hierarchy of structured types and typed tables (C++)"
- "tbconstr.sqC -- How to create, use, and drop constraints (C++)"
- "TbGenCol.java -- How to use generated columns (JDBC)"

ALTER TABLESPACE

The ALTER TABLESPACE statement is used to modify an existing table space in the following ways:

- Add a container to, or drop a container from a DMS table space; that is, a table space created with the MANAGED BY DATABASE option.
- Modify the size of a container in a DMS table space.
- Add a container to an SMS table space on a partition that currently has no containers.
- Modify the PREFETCHSIZE setting for a table space.
- Modify the BUFFERPOOL used for tables in the table space.
- Modify the OVERHEAD setting for a table space.
- Modify the TRANSFERRATE setting for a table space.
- Modify the file system caching policy for a table space.

Invocation:

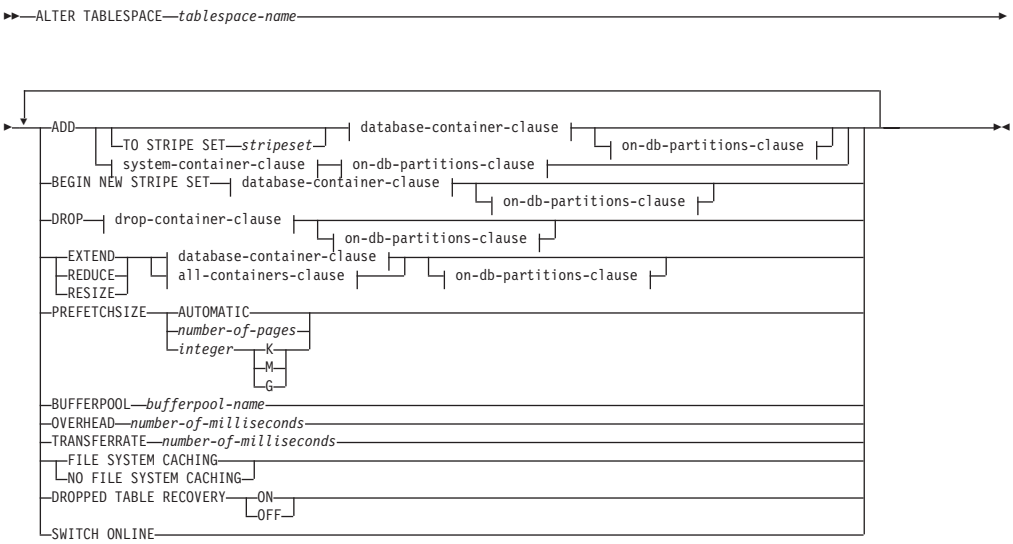
This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization:

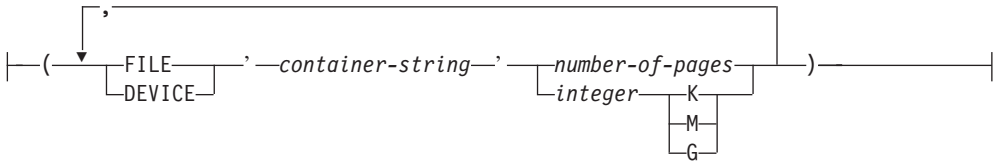
The authorization ID of the statement must have SYSCTRL or SYSADM authority.

Syntax:

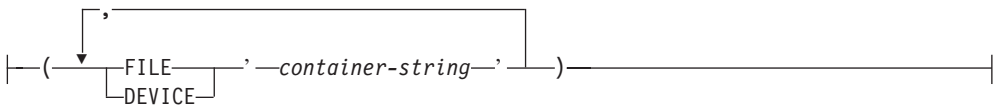
ALTER TABLESPACE



database-container-clause:

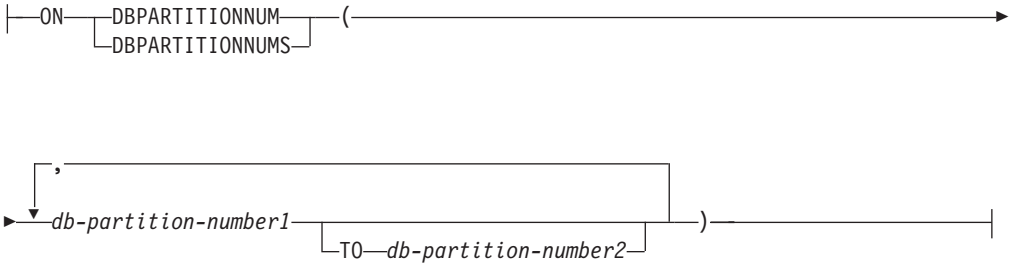


drop-container-clause:

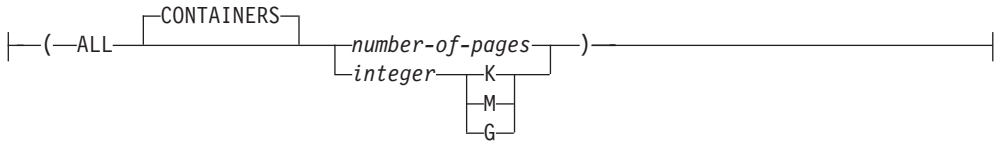


system-container-clause:

on-db-partitions-clause:



all-containers-clause:



Description:

tablespace-name
Names the table space. This is a one-part name. It is a long SQL identifier (either ordinary or delimited).

ADD
Specifies that one or more new containers are to be added to the table space.

TO STRIPE SET *stripeset*
Specifies that one or more new containers are to be added to the table space, and that they will be placed into the given stripe set.

BEGIN NEW STRIPE SET
Specifies that a new stripe set is to be created in the table space, and that one or more containers are to be added to this new stripe set. Containers that are subsequently added using the ADD option will be added to this new stripe set unless TO STRIPE SET is specified.

DROP
Specifies that one or more containers are to be dropped from the table space.

EXTEND
Specifies that existing containers are to be increased in size. The size specified is the size by which the existing container is increased. If the *all-containers-clause* is specified, all containers in the table space will increase by this size.

REDUCE
Specifies that existing containers are to be reduced in size. The size specified is the size by which the existing container is decreased. If the *all-containers-clause* is specified, all containers in the table space will decrease by this size.

RESIZE
Specifies that the size of existing containers is to be changed. The size specified is the new size for the container. If the *all-containers-clause* is specified, all containers in the table space will be changed to this size. If the operation affects more than one container, these containers must all either increase in size, or decrease in size. It is not possible to increase some while decreasing others (SQLSTATE 429BC).

database-container-clause
Adds one or more containers to a DMS table space. The table space must identify a DMS table space that already exists at the application server.

drop-container-clause
Drops one or more containers from a DMS table space. The table space must identify a DMS table space that already exists at the application server.

system-container-clause
Adds one or more containers to an SMS table space on the specified partitions. The table space must identify an SMS table space that already exists at the application server. There must not be any containers on the specified partitions for the table space. (SQLSTATE 42921).

on-db-partitions-clause
Specifies one or more partitions for the corresponding container operations.

all-containers-clause
Extends, reduces, or resizes all of the containers in a DMS table space. The table space must identify a DMS table space that already exists at the application server.

PREFETCHSIZE
Prefetching reads in data needed by a query prior to it being referenced by the query, so that the query need not wait for I/O to be performed.

AUTOMATIC
Specifies that the prefetch size of a table space is to be updated automatically; that is, the prefetch size will be managed by DB2, using the following formula:

```
Prefetch size =
 (number of containers) *
 (number of physical disks per container) *
 (extent size)
```

The number of physical disks per container defaults to 1, unless a value is specified through the DB2_PARALLEL_IO registry variable.

DB2 will update the prefetch size automatically whenever the number of containers in a table space changes (following successful execution of an ALTER TABLESPACE statement that adds or drops one or more containers). The prefetch size is updated at database start-up.

Automatic updating of the prefetch size can be turned off by specifying a numeric value in the PREFETCHSIZE clause.

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.

BUFFERPOOL *bufferpool-name*
The name of the buffer pool used for tables in this table space. The buffer pool must currently exist in the database (SQLSTATE 42704). The database partition group of the table space must be defined for the bufferpool (SQLSTATE 42735).

OVERHEAD *number-of-milliseconds*
Any numeric literal (integer, decimal, or floating point) that specifies the I/O controller overhead and disk seek and latency time, in milliseconds. The number should be an average for all containers that belong to the table space, if not the same for all containers. This value is used to determine the cost of I/O during query optimization.

TRANSFERRATE *number-of-milliseconds*
Any numeric literal (integer, decimal, or floating point) that specifies the time to read one page (4K or 8K) into memory, in milliseconds. The number should be an average for all containers that belong to the table space, if not the same for all containers. This value is used to determine the cost of I/O during query optimization.

FILE SYSTEM CACHING or **NO FILE SYSTEM CACHING**
Specifies whether or not I/O operations will be cached at the file system level. Connections to the database must be terminated before a new caching policy takes effect.

FILE SYSTEM CACHING
All I/O operations in the target table space will be cached at the file system level.

NO FILE SYSTEM CACHING
All I/O operations will bypass the file system level cache.

DROPPED TABLE RECOVERY
Dropped tables in the specified table space may be recovered using the RECOVER DROPPED TABLE ON option of the ROLLFORWARD command.

SWITCH ONLINE
table spaces in OFFLINE state are brought online if the containers have become accessible. If the containers are not accessible an error is returned (SQLSTATE 57048).

Notes:

- *Compatibilities*
  - For compatibility with versions earlier than Version 8, the keyword:
    - NODE can be substituted for DBPARTITIONNUM
    - NODES can be substituted for DBPARTITIONNUMS
- Each container definition requires 53 bytes plus the number of bytes necessary to store the container name. The combined length of all container names for the table space cannot exceed 20 480 bytes (SQLSTATE 54034).
- Default container operations are container operations that are specified in the ALTER TABLESPACE statement, but that are not explicitly directed to a specific database partition. These container operations are sent to any database partition that is not listed in the statement. If these default container operations are not sent to any database partition, because all database partitions are explicitly mentioned for a container operation, a warning is returned (SQLSTATE 1758W).
- Once space has been added or removed from a table space, and the transaction is committed, the contents of the table space may be rebalanced across the containers. Access to the table space is not restricted during rebalancing.
- If the table space is in OFFLINE state and the containers have become accessible, the user can disconnect all applications and connect to the database again to bring the table space out of OFFLINE state. Alternatively, SWITCH ONLINE option can bring the table space up (out of OFFLINE) while the rest of the database is still up and being used.
- If adding more than one container to a table space, it is recommended that they be added in the same statement so that the cost of rebalancing is incurred only once. An attempt to add containers to the same table space in separate ALTER TABLESPACE statements within a single transaction will result in an error (SQLSTATE 55041).
- Any attempts to extend, reduce, resize, or drop containers that do not exist will raise an error (SQLSTATE 428B2).
- When extending, reducing, or resizing a container, the container type must match the type that was used when the container was created (SQLSTATE 428B2).

- An attempt to change container sizes in the same table space, using separate ALTER TABLESPACE statements but within a single transaction, will raise an error (SQLSTATE 55041).
- In a partitioned database if more than one database partition resides on the same physical node, the same device or specific path cannot be specified for such database partitions (SQLSTATE 42730). For this environment, either specify a unique *container-string* for each database partition or use a relative path name.
- Although the table space definition is transactional and the changes to the table space definition are reflected in the catalog tables on commit, the buffer pool with the new definition cannot be used until the next time the database is started. The buffer pool in use, when the ALTER TABLESPACE statement was issued, will continue to be used in the interim.

Rules:

- The BEGIN NEW STRIPE SET clause cannot be specified in the same statement as ADD, DROP, EXTEND, REDUCE, and RESIZE, unless those clauses are being directed to different partitions (SQLSTATE 429BC).
- The stripe set value specified with the TO STRIPE SET clause must be within the valid range for the table space being altered (SQLSTATE 42615).
- When adding or removing space from the table space, the following rules must be followed:
  - EXTEND and RESIZE can be used in the same statement, provided that the size of each container is increasing (SQLSTATE 429BC).
  - REDUCE and RESIZE can be used in the same statement, provided that the size of each container is decreasing (SQLSTATE 429BC).
  - EXTEND and REDUCE cannot be used in the same statement, unless they are being directed to different partitions (SQLSTATE 429BC).
  - ADD cannot be used with REDUCE or DROP in the same statement, unless they are being directed to different partitions (SQLSTATE 429BC).
  - DROP cannot be used with EXTEND or ADD in the same statement, unless they are being directed to different partitions (SQLSTATE 429BC).

Examples:

Example 1: Add a device to the PAYROLL table space.

```
ALTER TABLESPACE PAYROLL
  ADD (DEVICE '/dev/rhdisk9' 10000)
```

Example 2: Change the prefetch size and I/O overhead for the ACCOUNTING table space.

```
ALTER TABLESPACE ACCOUNTING
  PREFETCHSIZE 64
  OVERHEAD 19.3
```

Example 3: Create a table space TS1, then resize the containers so that all of the containers have 2000 pages. (Three different ALTER TABLESPACE statements that will accomplish this resizing are shown.)

```
CREATE TABLESPACE TS1
  MANAGED BY DATABASE
  USING (FILE '/conts/cont0' 1000,
         DEVICE '/dev/rcont1' 500,
         FILE 'cont2' 700)
```

```
ALTER TABLESPACE TS1
  RESIZE (FILE '/conts/cont0' 2000,
          DEVICE '/dev/rcont1' 2000,
          FILE 'cont2' 2000)
```

OR

```
ALTER TABLESPACE TS1
  RESIZE (ALL 2000)
```

OR

```
ALTER TABLESPACE TS1
  EXTEND (FILE '/conts/cont0' 1000,
          DEVICE '/dev/rcont1' 1500,
          FILE 'cont2' 1300)
```

Example 4: Extend all of the containers in the DATA_TS table space by 1000 pages.

```
ALTER TABLESPACE DATA_TS
  EXTEND (ALL 1000)
```

Example 5: Resize all of the containers in the INDEX_TS table space to 100 megabytes (MB).

```
ALTER TABLESPACE INDEX_TS
  RESIZE (ALL 100 M)
```

Example 6: Add three new containers. Extend the first container, and resize the second.

```
ALTER TABLESPACE TS0
  ADD (FILE 'cont2' 2000, FILE 'cont3' 2000)
  ADD (FILE 'cont4' 2000)
  EXTEND (FILE 'cont0' 100)
  RESIZE (FILE 'cont1' 3000)
```

Example 7: Table space TSO exists on partitions 0, 1 and 2. Add a new container to database partition 0. Extend all of the containers on database partition 1. Resize a container on all database partitions other than the ones that were explicitly specified (that is, database partitions 0 and 1).

```
ALTER TABLESPACE TS0
  ADD (FILE 'A' 200) ON DBPARTITIONNUM (0)
  EXTEND (ALL 200) ON DBPARTITIONNUM (1)
  RESIZE (FILE 'B' 500)
```

The RESIZE clause is the default container clause in this example, and will be executed on database partition 2, because other operations are being explicitly sent to database partitions 0 and 1. If, however, there had only been these two database partitions, the statement would have succeeded, but returned a warning (SQL1758W) that default containers had been specified but not used.

Related reference:

- "CREATE TABLESPACE" on page 648
- "System environment variables" in the *Administration Guide: Performance*

ALTER VIEW

The ALTER VIEW statement modifies an existing view by altering a reference type column to add a scope.

Invocation:

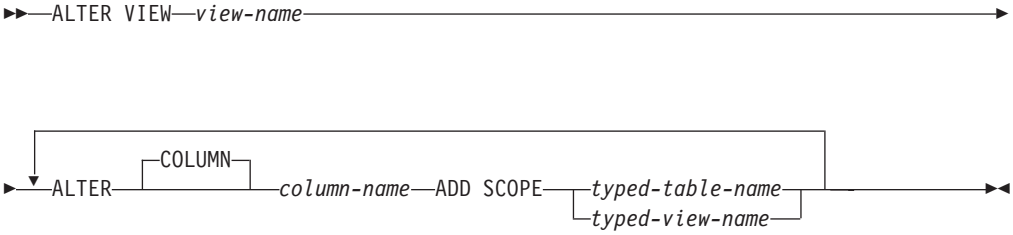
This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization:

The privileges held by the authorization ID of the statement must include at least one of the following:

- SYSADM or DBADM authority
- ALTERIN privilege on the schema of the view
- Definer of the view to be altered
- CONTROL privilege on the view to be altered.

Syntax:



Description:

view-name
Identifies the view to be changed. It must be a view described in the catalog.

ALTER COLUMN *column-name*
Is the name of the column to be altered in the view. The *column-name* must identify an existing column of the view (SQLSTATE 42703). The name cannot be qualified.

ADD SCOPE
Add a scope to an existing reference type column that does not already have a scope defined (SQLSTATE 428DK). The column must not be inherited from a superview (SQLSTATE 428DJ).

typed-table-name
The name of a typed table. The data type of *column-name* must be REF(*S*), where *S* is the type of *typed-table-name* (SQLSTATE 428DM). No checking is done of any existing values in *column-name* to ensure that the values actually reference existing rows in *typed-table-name*.

typed-view-name
The name of a typed view. The data type of *column-name* must be REF(*S*), where *S* is the type of *typed-view-name* (SQLSTATE 428DM). No checking is done of any existing values in *column-name* to ensure that the values actually reference existing rows in *typed-view-name*.

COMMENT

The COMMENT statement adds or replaces comments in the catalog descriptions of various objects.

Invocation:

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

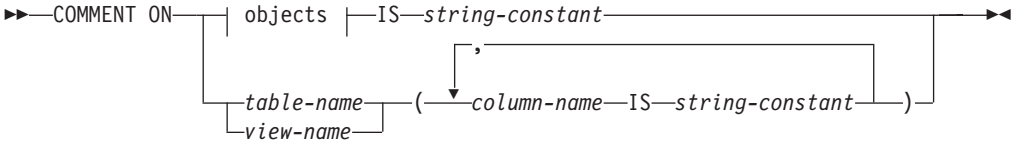
Authorization:

The privileges that must be held by the authorization ID of the COMMENT statement must include one of the following:

- SYSADM or DBADM
- definer of the object (underlying table for column or constraint) as recorded in the DEFINER column of the catalog view for the object (OWNER column for a schema)
- ALTERIN privilege on the schema (applicable only to objects allowing more than one-part names)
- CONTROL privilege on the object (applicable to index, package, table and view objects only)
- ALTER privilege on the object (applicable to table objects only)

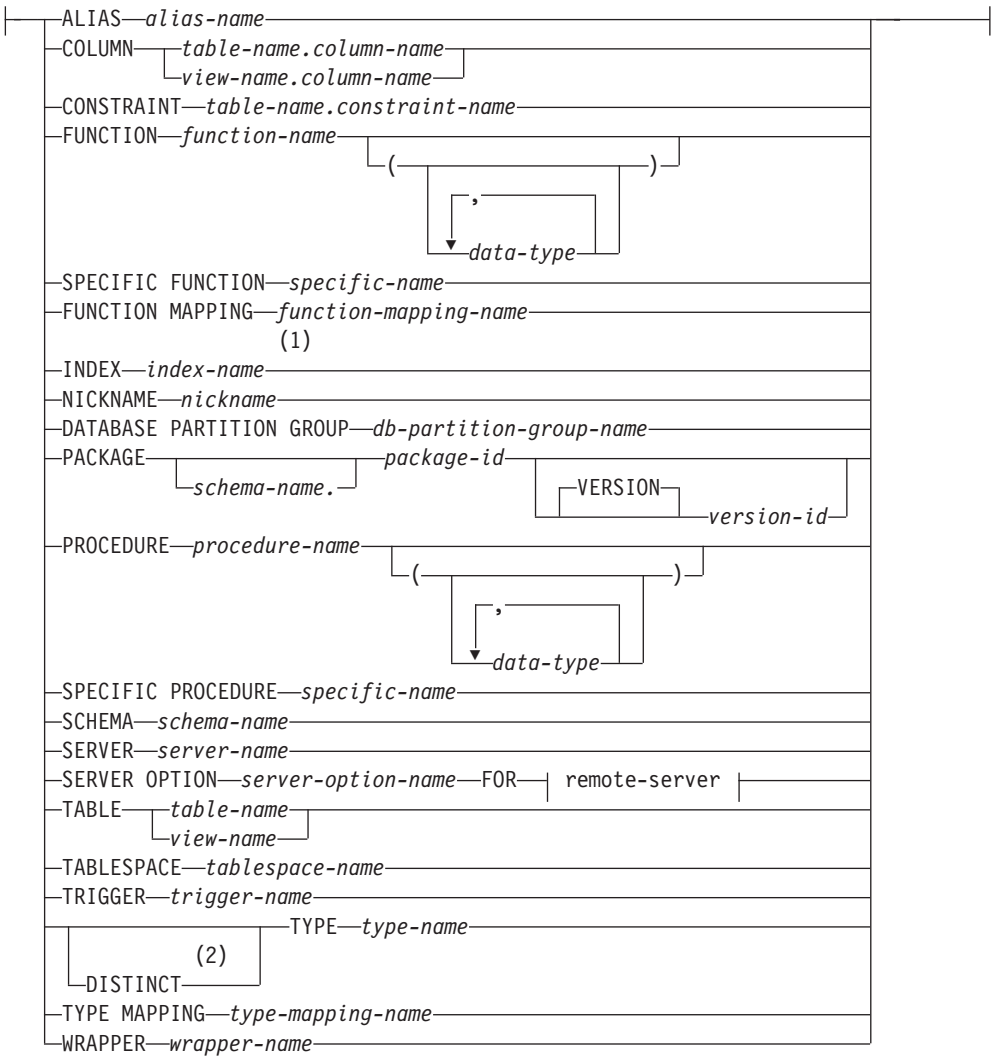
Note that for table space or database partition group, the authorization ID must have SYSADM or SYSCTRL authority.

Syntax:



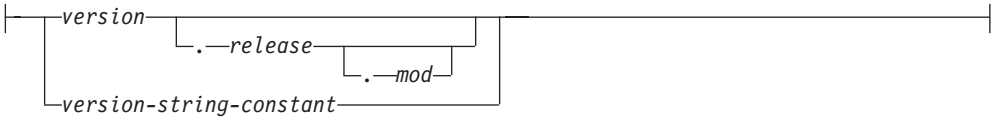
objects:

COMMENT



remote-server:

server-version:



Notes:

1 *Index-name* can be the name of either an index or an index specification.

2 The keyword DATA can be used as a synonym for DISTINCT.

Description:

ALIAS *alias-name*
Indicates a comment will be added or replaced for an alias. The *alias-name* must identify an alias that is described in the catalog (SQLSTATE 42704). The comment replaces the value of the REMARKS column of the SYSCAT.TABLES catalog view for the row that describes the alias.

COLUMN *table-name.column-name* or *view-name.column-name*
Indicates a comment will be added or replaced for a column. The *table-name.column-name* or *view-name.column-name* combination must identify a column and table combination that is described in the catalog (SQLSTATE 42704). The comment replaces the value of the REMARKS column of the SYSCAT.COLUMNS catalog view for the row that describes the column.

A comment cannot be made on a column of an inoperative view. (SQLSTATE 51024).

CONSTRAINT *table-name.constraint-name*
Indicates a comment will be added or replaced for a constraint. The *table-name.constraint-name* combination must identify a constraint and the table that it constrains; they must be described in the catalog (SQLSTATE 42704). The comment replaces the value of the REMARKS column of the SYSCAT.TABCONST catalog view for the row that describes the constraint.

FUNCTION
Indicates a comment will be added or replaced for a function. The function instance specified must be a user-defined function or function template described in the catalog.

There are several different ways available to identify the function instance:

FUNCTION *function-name*
Identifies the particular function, and is valid only if there is exactly one function with the *function-name*. The function thus identified may have any number of parameters defined for it. In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified object names. If no function by this name exists in the named or implied schema, an error (SQLSTATE 42704) is raised. If there is more than one specific instance of the function in the named or implied schema, an error (SQLSTATE 42725) is raised.

FUNCTION *function-name* **(***data-type***,...)**
Provides the function signature, which uniquely identifies the function to be commented upon. The function selection algorithm is *not* used.

function-name
Gives the function name of the function to be commented upon. In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified object names.

(*data-type***,...)**
Must match the data types that were specified on the CREATE FUNCTION statement in the corresponding position. The number of data types, and the logical concatenation of the data types is used to identify the specific function for which to add or replace the comment.

If the *data-type* is unqualified, the type name is resolved by searching the schemas on the SQL path. This also applies to data type names specified for a REFERENCE type.

It is not necessary to specify the length, precision or scale for the parameterized data types. Instead an empty set of parentheses may be coded to indicate that these attributes are to be ignored when looking for a data type match.

FLOAT() cannot be used (SQLSTATE 42601) since the parameter value indicates different data types (REAL or DOUBLE).

However, if length, precision, or scale is coded, the value must exactly match that specified in the CREATE FUNCTION statement.

A type of FLOAT(n) does not need to match the defined value for n since 0 <n<25 means REAL and 24<n<54 means DOUBLE. Matching occurs based on whether the type is REAL or DOUBLE.

(Note that the FOR BIT DATA attribute is not considered part of the signature for matching purposes. So, for example, a CHAR FOR BIT DATA specified in the signature would match a function defined with CHAR only, and vice versa.)

If no function with the specified signature exists in the named or implied schema, an error (SQLSTATE 42883) is raised.

SPECIFIC FUNCTION *specific-name*
Indicates that comments will be added or replaced for a function (see FUNCTION for other methods of identifying a function). Identifies the particular user-defined function that is to be commented upon, using the specific name either specified or defaulted to at function creation time. In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified object names. The *specific-name* must identify a specific function instance in the named or implied schema; otherwise, an error (SQLSTATE 42704) is raised.

It is not possible to comment on a function that is in the SYSIBM, SYSFUN, or SYSPROC schema (SQLSTATE 42832).

The comment replaces the value of the REMARKS column of the SYSCAT.ROUTINES catalog view for the row that describes the function.

FUNCTION MAPPING *function-mapping-name*
Indicates a comment will be added or replaced for a function mapping. The *function-mapping-name* must identify a function mapping that is described in the catalog (SQLSTATE 42704). The comment replaces the value for the REMARKS column of the SYSCAT.FUNCMAPPINGS catalog view for the row that describes the function mapping.

INDEX *index-name*
Indicates a comment will be added or replaced for an index or index specification. The *index-name* must identify either a distinct index or an index specification that is described in the catalog (SQLSTATE 42704). The comment replaces the value for the REMARKS column of the SYSCAT.INDEXES catalog view for the row that describes the index or index specification.

NICKNAME *nickname*
Indicates a comment will be added or replaced for a nickname. The *nickname* must be a nickname that is described in the catalog (SQLSTATE 42704). The comment replaces the value for the REMARKS column of the SYSCAT.TABLES catalog view for the row that describes the nickname.

DATABASE PARTITION GROUP *db-partition-group-name*
Indicates a comment will be added or replaced for a database partition group. The *db-partition-group-name* must identify a distinct database partition group that is described in the catalog (SQLSTATE 42704). The comment replaces the value for the REMARKS column of the SYSCAT.DBPARTITIONGROUPS catalog view for the row that describes the database partition group.

PACKAGE *schema-name.package-id*
Indicates that a comment will be added or replaced for a package. If a schema name is not specified, the package ID is implicitly qualified by the default schema. The schema name and package ID, together with the implicitly or explicitly specified version ID, must identify a package that is described in the catalog (SQLSTATE 42704). The comment replaces the value for the REMARKS column of the SYSCAT.PACKAGES catalog view for the row that describes the package.

VERSION *version-id*
Identifies which package version is to be commented on. If a value is not specified, the version defaults to the empty string. If multiple packages with the same package name but different versions exist, only one package version can be commented on in one invocation of the COMMENT statement. Delimit the version identifier with double quotation marks when it:

- Is generated by the VERSION(AUTO) precompiler option
- Begins with a digit
- Contains lowercase or mixed-case letters

If the statement is invoked from an operating system command prompt, precede each double quotation mark delimiter with a back slash character to ensure that the operating system does not strip the delimiters.

PROCEDURE
Indicates a comment will be added or replaced for a procedure. The procedure instance specified must be a stored procedure described in the catalog.

There are several different ways available to identify the procedure instance:

PROCEDURE *procedure-name*
Identifies the particular procedure, and is valid only if there is exactly one procedure with the *procedure-name* in the schema. The procedure thus identified may have any number of parameters defined for it. In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified object names. If no procedure by this name exists in the named or implied schema, an error (SQLSTATE 42704) is raised. If there is more than one specific instance of the procedure in the named or implied schema, an error (SQLSTATE 42725) is raised.

PROCEDURE *procedure-name* **(***data-type***,...)**
This is used to provide the procedure signature, which uniquely identifies the procedure to be commented upon.

procedure-name
Gives the procedure name of the procedure to be commented upon. In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified object names.

(*data-type***,...)**
Must match the data types that were specified on the CREATE PROCEDURE statement in the corresponding position. The number of data types, and the logical concatenation of the data types is used to identify the specific procedure for which to add or replace the comment.

If the *data-type* is unqualified, the type name is resolved by searching the schemas on the SQL path. This also applies to data type names specified for a REFERENCE type.

It is not necessary to specify the length, precision or scale for the parameterized data types. Instead an empty set of parentheses may be coded to indicate that these attributes are to be ignored when looking for a data type match.

FLOAT() cannot be used (SQLSTATE 42601) since the parameter value indicates different data types (REAL or DOUBLE).

However, if length, precision, or scale is coded, the value must exactly match that specified in the CREATE PROCEDURE statement.

A type of FLOAT(n) does not need to match the defined value for n since 0<n<25 means REAL and 24<n<54 means DOUBLE. Matching occurs based on whether the type is REAL or DOUBLE.

If no procedure with the specified signature exists in the named or implied schema, an error (SQLSTATE 42883) is raised.

SPECIFIC PROCEDURE *specific-name*
Indicates that comments will be added or replaced for a procedure (see PROCEDURE for other methods of identifying a procedure). Identifies the particular stored procedure that is to be commented upon, using the specific name either specified or defaulted to at procedure creation time. In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified object names. The *specific-name* must identify a specific procedure instance in the named or implied schema; otherwise, an error (SQLSTATE 42704) is raised.

It is not possible to comment on a procedure that is in the SYSIBM, SYSFUN, or SYSPROC schema (SQLSTATE 42832).

The comment replaces the value of the REMARKS column of the SYSCAT.ROUTINES catalog view for the row that describes the procedure.

SCHEMA *schema-name*
Indicates a comment will be added or replaced for a schema. The *schema-name* must identify a schema that is described in the catalog (SQLSTATE 42704). The comment replaces the value of the REMARKS column of the SYSCAT.SCHEMATA catalog view for the row that describes the schema.

SERVER *server-name*
Indicates a comment will be added or replaced for a data source. The *server-name* must identify a data source that is described in the catalog (SQLSTATE 42704). The comment replaces the value for the REMARKS column of the SYSCAT.SERVERS catalog view for the row that describes the data source.

SERVER OPTION *server-option-name* **FOR** *remote-server*
Indicates a comment will be added or replaced for a server option.

server-option-name
Identifies a server option. This option must be one that is described in the catalog (SQLSTATE 42704). The comment replaces the value for the REMARKS column of the SYSCAT.SERVEROPTIONS catalog view for the row that describes the server option.

remote-server
Describes the data source to which the *server-option* applies.

SERVER *server-name*
Names the data source to which the *server-option* applies. The *server-name* must identify a data source that is described in the catalog.

TYPE *server-type*
Specifies the type of data source—for example, DB2 Universal Database for OS/390 or Oracle—to which the *server-option* applies. The *server-type* can be specified in either lower- or uppercase; it will be stored in uppercase in the catalog.

VERSION
Specifies the version of the data source identified by *server-name*.

version
Specifies the version number. *version* must be an integer.

release
Specifies the number of the release of the version denoted by *version. release* must be an integer.

mod
Specifies the number of the modification of the release denoted by *release. mod* must be an integer.

version-string-constant
Specifies the complete designation of the version. The *version-string-constant* can be a single value (for example, '8i'); or it can be the concatenated values of *version*, *release*, and, if applicable, *mod* (for example, '8.0.3').

WRAPPER *wrapper-name*
Identifies the wrapper that is used to access the data source referenced by *server-name.*

TABLE *table-name* or *view-name*
Indicates a comment will be added or replaced for a table or view. The *table-name* or *view-name* must identify a table or view (not an alias or nickname) that is described in the catalog (SQLSTATE 42704) and must not identify a declared temporary table (SQLSTATE 42995). The comment replaces the value for the REMARKS column of the SYSCAT.TABLES catalog view for the row that describes the table or view.

TABLESPACE *tablespace-name*
Indicates a comment will be added or replaced for a table space. The *tablespace-name* must identify a distinct table space that is described in the catalog (SQLSTATE 42704). The comment replaces the value for the REMARKS column of the SYSCAT.TABLESPACES catalog view for the row that describes the tablespace.

TRIGGER *trigger-name*
Indicates a comment will be added or replaced for a trigger. The *trigger-name* must identify a distinct trigger that is described in the catalog (SQLSTATE 42704). The comment replaces the value for the REMARKS column of the SYSCAT.TRIGGERS catalog view for the row that describes the trigger.

TYPE *type-name*
Indicates a comment will be added or replaced for a user-defined type. The *type-name* must identify a user-defined type that is described in the catalog (SQLSTATE 42704). If DISTINCT is specified, *type-name* must identify a distinct type that is described in the catalog (SQLSTATE 42704). The comment replaces the value of the REMARKS column of the SYSCAT.DATATYPES catalog view for the row that describes the user-defined type.

In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified object names.

TYPE MAPPING *type-mapping-name*
Indicates a comment will be added or replaced for a user-defined data type mapping. The *type-mapping-name* must identify a data type mapping that is described in the catalog (SQLSTATE 42704). The comment replaces the value for the REMARKS column of the SYSCAT.TYPEMAPPINGS catalog view for the row that describes the mapping.

WRAPPER *wrapper-name*
Indicates a comment will be added or replaced for a wrapper. The *wrapper-name* must identify a wrapper that is described in the catalog (SQLSTATE 42704). The comment replaces the value for the REMARKS column of the SYSCAT.WRAPPERS catalog view for the row that describes the wrapper.

IS *string-constant*
Specifies the comment to be added or replaced. The *string-constant* can be any character string constant of up to 254 bytes. (Carriage return and line feed each count as 1 byte.)

table-name | *view-name* **({** *column-name* **IS** *string-constant* **} ...)**
This form of the COMMENT statement provides the ability to specify comments for multiple columns of a table or view. The column names must not be qualified, each name must identify a column of the specified table or view, and the table or view must be described in the catalog. The *table-name* cannot be a declared temporary table (SQLSTATE 42995).

A comment cannot be made on a column of an inoperative view (SQLSTATE 51024).

Notes:

- ***Compatibilities***
  - For compatibility with previous versions of DB2:

- NODEGROUP can be specified in place of DATABASE PARTITION GROUP

Examples:

Example 1: Add a comment for the EMPLOYEE table.

```
COMMENT ON TABLE EMPLOYEE
  IS 'Reflects first quarter reorganization'
```

Example 2: Add a comment for the EMP_VIEW1 view.

```
COMMENT ON TABLE EMP_VIEW1
  IS 'View of the EMPLOYEE table without salary information'
```

Example 3: Add a comment for the EDLEVEL column of the EMPLOYEE table.

```
COMMENT ON COLUMN EMPLOYEE.EDLEVEL
  IS 'highest grade level passed in school'
```

Example 4: Add comments for two different columns of the EMPLOYEE table.

```
COMMENT ON EMPLOYEE
  (WORKDEPT IS 'see DEPARTMENT table for names',
  EDLEVEL IS 'highest grade level passed in school' )
```

Example 5: Pellow wants to comment on the CENTRE function, which he created in his PELLOW schema, using the signature to identify the specific function to be commented on.

```
COMMENT ON FUNCTION CENTRE (INT,FLOAT)
  IS 'Frank''s CENTRE fctn, uses Chebychev method'
```

Example 6: McBride wants to comment on another CENTRE function, which she created in the PELLOW schema, using the specific name to identify the function instance to be commented on:

```
COMMENT ON SPECIFIC FUNCTION PELLOW.FOCUS92 IS
  'Louise''s most triumphant CENTRE function, uses the
       Brownian fuzzy-focus technique'
```

Example 7: Comment on the function ATOMIC_WEIGHT in the CHEM schema, where it is known that there is only one function with that name:

```
COMMENT ON FUNCTION CHEM.ATOMIC_WEIGHT
  IS 'takes atomic nbr, gives atomic weight'
```

Example 8: Eigler wants to comment on the SEARCH procedure, which he created in his EIGLER schema, using the signature to identify the specific procedure to be commented on.

```
COMMENT ON PROCEDURE SEARCH (CHAR,INT)
  IS 'Frank''s mass search and replace algorithm'
```

Example 9: Macdonald wants to comment on another SEARCH function, which he created in the EIGLER schema, using the specific name to identify the procedure instance to be commented on:

```
COMMENT ON SPECIFIC PROCEDURE EIGLER.DESTROY IS
  'Patrick''s mass search and destroy algorithm'
```

Example 10: Comment on the procedure OSMOSIS in the BIOLOGY schema, where it is known that there is only one procedure with that name:

```
COMMENT ON PROCEDURE BIOLOGY.OSMOSIS
  IS 'Calculations modelling osmosis'
```

Example 11: Comment on an index specification named INDEXSPEC.

```
COMMENT ON INDEX INDEXSPEC
  IS 'An index specification that indicates to the optimizer
  that the table referenced by nickname NICK1 has an index.'
```

Example 12: Comment on the wrapper whose default name is NET8.

```
COMMENT ON WRAPPER NET8
  IS 'The wrapper for data sources associated with
      Oracle's Net8 client software.'
```

CREATE FUNCTION

This statement is used to register or define a user-defined function or function template with an application server.

There are five different types of functions that can be created using this statement. Each of these is described separately.

- External Scalar. The function is written in a programming language and returns a scalar value. The external executable is registered in the database, along with various attributes of the function.
- External Table. The function is written in a programming language and returns a complete table. The external executable is registered in the database along with various attributes of the function.
- OLE DB External Table. A user-defined OLE DB external table function is registered in the database to access data from an OLE DB provider.
- Sourced or Template. A source function is implemented by invoking another function (either built-in, external, SQL, or source) that is already registered in the database.

  It is possible to create a partial function, called a *function template*, which defines what types of values are to be returned, but which contains no executable code. The user maps it to a data source function within a federated system, so that the data source function can be invoked from a federated database. A function template can be registered only with an application server that is designated as a federated server.
- SQL Scalar, Table or Row. The function body is written in SQL and defined together with the registration in the database. It returns a scalar value, a table, or a single row.

Related reference:

- "CREATE FUNCTION (OLE DB External Table) statement" in the *SQL Reference, Volume 2*
- "CREATE FUNCTION (SQL Scalar, Table, or Row) statement" in the *SQL Reference, Volume 2*
- "CREATE FUNCTION (External Scalar) statement" in the *SQL Reference, Volume 2*
- "CREATE FUNCTION (External Table) statement" in the *SQL Reference, Volume 2*
- "CREATE FUNCTION (Sourced or Template) statement" in the *SQL Reference, Volume 2*

Related samples:

- "dbinline.sqc -- How to use inline SQL Procedure Language (C)"
- "udfcli.sqc -- Call a variety of types of user-defined functions (C)"

- "udfemcli.sqc -- Call a variety of types of embedded SQL user-defined functions. (C)"
- "udfcli.c -- How to work with different types of user-defined functions (UDFs)"
- "udfcli.sqC -- Call a variety of types of user-defined functions (C++)"
- "udfemcli.sqC -- Call a variety of types of embedded SQL user-defined functions. (C++)"
- "UDFCreate.db2 -- How to catalog the Java UDFs contained in UDFsrv.java "
- "UDFjCreate.db2 -- How to catalog the Java UDFs contained in UDFjsrv.java "

CREATE INDEX

The CREATE INDEX statement is used to:

- Create an index on a DB2 table
- Create an index specification (metadata that indicates to the optimizer that a data source table has an index)

Invocation:

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization:

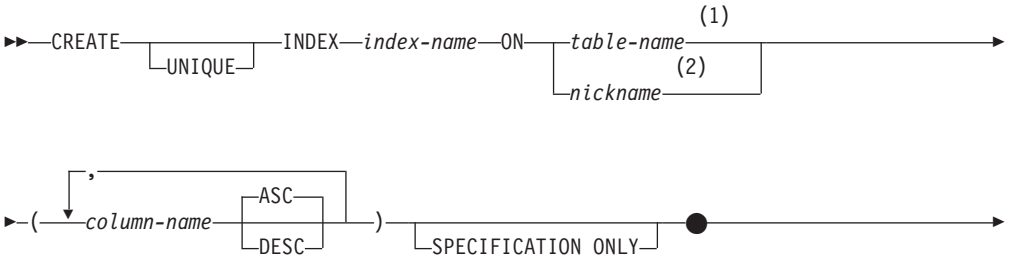
The privileges held by the authorization ID of the statement must include at least one of the following:

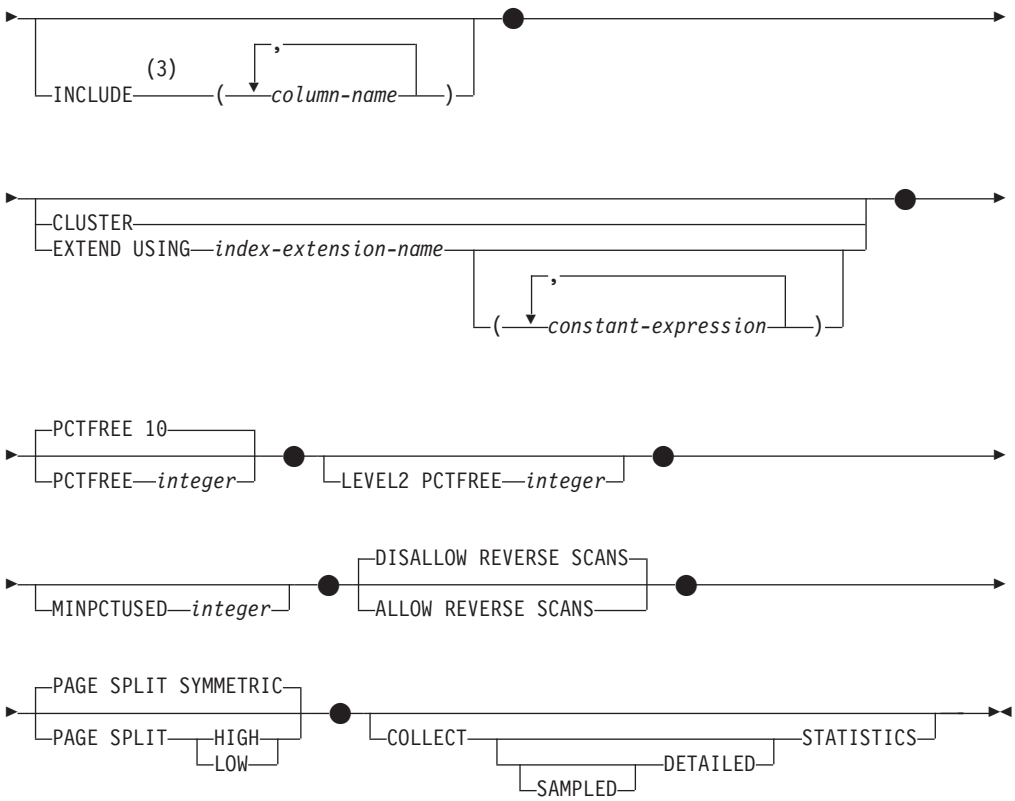
- SYSADM or DBADM authority.
- One of:
  - CONTROL privilege on the table or nickname on which the index is defined
  - INDEX privilege on the table or nickname on which the index is defined.

  and one of:
  - IMPLICIT_SCHEMA authority on the database, if the implicit or explicit schema name of the index does not exist
  - CREATEIN privilege on the schema, if the schema name of the index refers to an existing schema.

No explicit privilege is required to create an index on a declared temporary table.

Syntax:





Notes:

1 In a federated system, *table-name* must identify a table in the federated database. It cannot identify a data source table.

2 If *nickname* is specified, the CREATE INDEX statement creates an index specification. In this case, INCLUDE, CLUSTER, EXTEND USING, PCTFREE, MINPCTUSED, DISALLOW REVERSE SCANS, ALLOW REVERSE SCANS, PAGE SPLIT, or COLLECT STATISTICS cannot be specified.

3 The INCLUDE clause can only be specified if UNIQUE is specified.

Description:

UNIQUE

If ON *table-name* is specified, UNIQUE prevents the table from containing two or more rows with the same value of the index key. The uniqueness is enforced at the end of the SQL statement that updates rows or inserts new rows.

The uniqueness is also checked during the execution of the CREATE INDEX statement. If the table already contains rows with duplicate key values, the index is not created.

When UNIQUE is used, null values are treated as any other values. For example, if the key is a single column that may contain null values, that column may contain no more than one null value.

If the UNIQUE option is specified, and the table has a partitioning key, the columns in the index key must be a superset of the partitioning key. That is, the columns specified for a unique index key must include all the columns of the partitioning key (SQLSTATE 42997).

Primary or unique keys cannot be subsets of dimensions (SQLSTATE 429BE).

If ON *nickname* is specified, UNIQUE should be specified only if the data for the index key contains unique values for every row of the data source table. The uniqueness will not be checked.

INDEX *index-name*
Names the index or index specification. The name, including the implicit or explicit qualifier, must not identify an index or index specification that is described in the catalog, or an existing index on a declared temporary table (SQLSTATE 42704). The qualifier must not be SYSIBM, SYSCAT, SYSFUN, or SYSSTAT (SQLSTATE 42939).

The implicit or explicit qualifier for indexes on declared global temporary tables must be SESSION (SQLSTATE 428EK).

ON *table-name* **or** *nickname*
The *table-name* identifies a table on which an index is to be created. The table must be a base table (not a view), a materialized query table described in the catalog, or a declared temporary table. The name of a declared temporary table must be qualified with SESSION. The *table-name* must not identify a catalog table (SQLSTATE 42832). If UNIQUE is specified and *table-name* is a typed table, it must not be a subtable (SQLSTATE 429B3).

nickname is the nickname on which an index specification is to be created. The *nickname* references either a data source table whose index is described by the index specification, or a data source view that is based on such a table. The *nickname* must be listed in the catalog.

column-name
For an index, *column-name* identifies a column that is to be part of the index key. For an index specification, *column-name* is the name by which the federated server references a column of a data source table.

Each *column-name* must be an unqualified name that identifies a column of the table. Up to 16 columns can be specified. If *table-name* is a typed table, up to 15 columns can be specified. If *table-name* is a subtable, at least one *column-name* must be introduced in the subtable; that is, not inherited from a supertable (SQLSTATE 428DS). No *column-name* can be repeated (SQLSTATE 42711).

The sum of the stored lengths of the specified columns must not be greater than 1024. If *table-name* is a typed table, the index key length limit is further reduced by 4 bytes.

Note that this length can be reduced by system overhead, which varies according to the data type of the column and whether it is nullable. For more information on overhead affecting this limit, see “Bytes Counts” in “CREATE TABLE”.

No LOB column, DATALINK column, or distinct type column based on a LOB or DATALINK may be used as part of an index, even if the length attribute of the column is small enough to fit within the 1024-byte limit (SQLSTATE 54008). A structured type column can only be specified if the EXTEND USING clause is also specified (SQLSTATE 42962). If the EXTEND USING clause is specified, only one column can be specified, and the type of the column must be a structured type or a distinct type that is not based on a LOB, DATALINK, LONG VARCHAR, or LONG VARGRAPHIC (SQLSTATE 42997).

ASC
Specifies that index entries are to be kept in ascending order of the column values; this is the default setting. ASC cannot be specified for indexes that are defined with EXTEND USING (SQLSTATE 42601).

DESC
Specifies that index entries are to be kept in descending order of the column values. DESC cannot be specified for indexes that are defined with EXTEND USING (SQLSTATE 42601).

SPECIFICATION ONLY
Indicates that this statement will be used to create an index specification that applies to the data source table referenced by *nickname*. SPECIFICATION ONLY must be specified if *nickname* is specified (SQLSTATE 42601). It cannot be specified if *table-name* is specified (SQLSTATE 42601).

This clause cannot be used when creating an index on a declared temporary table (SQLSTATE 42995).

INCLUDE
This keyword introduces a clause that 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 may improve the performance of some queries through index only access. The columns must be distinct from the columns used to enforce uniqueness (SQLSTATE 42711). The limits for the number of columns and sum of the length attributes apply to all of the columns in the unique key and in the index.

This clause cannot be used with declared temporary tables (SQLSTATE 42995).

column-name
Identifies a column that is included in the index but not part of the unique index key. The same rules apply as defined for columns of the unique index key. The keywords ASC or DESC may be specified following the column-name but have no effect on the order.

INCLUDE cannot be specified for indexes that are defined with EXTEND USING, or if *nickname* is specified (SQLSTATE 42601).

CLUSTER
Specifies that the index is the clustering index of the table. The cluster factor of a clustering index is maintained or improved dynamically as data is inserted into the associated table, by attempting to insert new rows physically close to the rows for which the key values of this index are in the same range. Only one clustering index may exist for a table so CLUSTER may not be specified if it was used in the definition of any existing index on the table (SQLSTATE 55012). A clustering index may not be created on a table that is defined to use append mode (SQLSTATE 428D8).

CLUSTER is disallowed if *nickname* is specified (SQLSTATE 42601). This clause cannot be used with declared temporary tables (SQLSTATE 42995) or range-clustered tables (SQLSTATE 429BG).

EXTEND USING *index-extension-name*
Names the *index-extension* used to manage this index. If this clause is specified, then there must be only one *column-name* specified and that column must be a structured type or a distinct type (SQLSTATE 42997). The *index-extension-name* must name an index extension described in the catalog (SQLSTATE 42704). For a distinct type, the column must exactly match the type of the corresponding source key parameter in the index extension. For a structured type column, the type of the corresponding source key parameter must be the same type or a supertype of the column type (SQLSTATE 428E0).

This clause cannot be used with declared temporary tables (SQLSTATE 42995).

constant-expression
Identifies values for any required arguments for the index extension. Each expression must be a constant value with a data type that exactly matches the defined data type of the corresponding index extension parameters, including length or precision, and scale (SQLSTATE 428E0). This clause must not exceed 32 768 bytes in length in the database code page (SQLSTATE 22001).

PCTFREE *integer*
Specifies what percentage of each index page to leave as free space when building the index. The first entry in a page is added without restriction. When additional entries are placed in an index page at least *integer* percent of free space is left on each page. The value of *integer* can range from 0 to 99. However, if a value greater than 10 is specified, only 10 percent free space will be left in non-leaf pages. The default is 10.

PCTFREE is disallowed if *nickname* is specified (SQLSTATE 42601). This clause cannot be used with declared temporary tables (SQLSTATE 42995).

LEVEL2 PCTFREE *integer*
Specifies what percentage of each index level 2 page to leave as free space when building the index. The value of *integer* can range from 0 to 99. If LEVEL2 PCTFREE is not set, a minimum of 10 or PCTFREE percent of free space is left on all non-leaf pages. If LEVEL2 PCTFREE is set, *integer* percent of free space is left on level 2 intermediate pages, and a minimum of 10 or *integer* percent of free space is left on level 3 and higher intermediate pages.

LEVEL2 PCTFREE is disallowed if *nickname* is specified (SQLSTATE 42601). This clause cannot be used with declared temporary tables (SQLSTATE 42995).

MINPCTUSED *integer*
Indicates whether index leaf pages are merged online, and the threshold for the minimum percentage of space used on an index leaf page. If, after a key is removed from an index leaf page, the percentage of space used on the page is at or below *integer* percent, an attempt is made to merge the remaining keys on this page with those of a neighboring page. If there is sufficient space on one of these pages, the merge is performed and one of the pages is deleted. The value of *integer* can be from 0 to 99. However, a value of 50 or below is recommended for performance reasons. Specifying this option will have an impact on update and delete performance. For type 2 indexes, merging is only done during update and delete operations when there is an exclusive table lock. If an exclusive table lock does not exist, keys are marked as pseudo deleted during update and delete operations, and no merging is done. Consider using the CLEANUP ONLY ALL option of REORG INDEXES to merge leaf pages instead of using the MINPCTUSED option of CREATE INDEX.

MINPCTUSED is disallowed if *nickname* is specified (SQLSTATE 42601). This clause cannot be used with declared temporary tables (SQLSTATE 42995).

DISALLOW REVERSE SCANS
Specifies that an index only supports forward scans or scanning of the index in the order defined at INDEX CREATE time. This is the default.

DISALLOW REVERSE SCANS is disallowed if *nickname* is specified (SQLSTATE 42601).

ALLOW REVERSE SCANS
Specifies that an index can support both forward and reverse scans; that is, in the order defined at INDEX CREATE time and in the opposite (or reverse) order.

ALLOW REVERSE SCANS is disallowed if *nickname* is specified (SQLSTATE 42601).

PAGE SPLIT
Specifies an index split behavior. The default is SYMMETRIC.

SYMMETRIC
Specifies that pages are to be split roughly in the middle.

HIGH
Specifies an index page split behavior that uses the space on index pages efficiently when the values of the index keys being inserted follow a particular pattern. The index key must contain more than one column. For a subset of index key values, the leftmost column or columns of the index must contain the same value, and the rightmost column or columns of the index must contain values that increase with each insertion. For details, see "Options on the CREATE INDEX statement".

LOW
Specifies an index page split behavior that uses the space on index pages efficiently when the values of the index keys being inserted follow a particular pattern. The index key must contain more than one column. For a subset of index key values, the leftmost column or columns of the index must contain the same value, and the rightmost column or columns of the index must contain values that decrease with each insertion. For details, see "Options on the CREATE INDEX statement".

COLLECT STATISTICS
Specifies that basic index statistics are to be collected during index creation.

DETAILED
Specifies that extended index statistics (CLUSTERFACTOR and PAGE_FETCH_PAIRS) are also to be collected during index creation.

SAMPLED
Specifies that sampling can be used when compiling extended index statistics.

Rules:

- The CREATE INDEX statement will fail (SQLSTATE 01550) if attempting to create an index that matches an existing index. Two index descriptions are considered duplicates if:
  - the set of columns (both key and include columns) and their order in the index is the same as that of an existing index AND
  - the ordering attributes are the same AND
  - both the previously existing index and the one being created are non-unique OR the previously existing index is unique AND
  - if both the previously existing index and the one being created are unique, the key columns of the index being created are the same or a superset of key columns of the previously existing index.

Notes:

- ***Compatibilities***
  - For compatibility with DB2 for OS/390:

  - The following syntax is tolerated and ignored:
    - CLOSE
    - DEFINE
    - FREEPAGE
    - GBPCACHE
    - PIECESIZE
    - TYPE 2
    - using-block
  - The following syntax is accepted as the default behavior:
    - COPY NO
    - DEFER NO
- Concurrent read/write access to the table is permitted while an index is being created. Once the index has been built, changes that were made to the table during index creation time are forward-fitted to the new index. Write access to the table is then briefly blocked while index creation completes, after which the new index becomes available.

  To circumvent this default behavior, use the LOCK TABLE statement to explicitly lock the table before issuing a CREATE INDEX statement. (The table can be locked in either SHARE or EXCLUSIVE mode, depending on whether read access is to be allowed.)
- If the named table already contains data, CREATE INDEX creates the index entries for it. If the table does not yet contain data, CREATE INDEX creates a description of the index; the index entries are created when data is inserted into the table.
- Once the index is created and data is loaded into the table, it is advisable to issue the RUNSTATS command. The RUNSTATS command updates statistics collected on the database tables, columns, and indexes. These statistics are used to determine the optimal access path to the tables. By issuing the RUNSTATS command, the database manager can determine the characteristics of the new index. If data has been loaded before the CREATE INDEX statement is issued, it is recommended that the COLLECT STATISTICS option on the CREATE INDEX statement be used as an alternative to the RUNSTATS command.
- Creating an index with a schema name that does not already exist will result in the implicit creation of that schema provided the authorization ID of the statement has IMPLICIT_SCHEMA authority. The schema owner is SYSIBM. The CREATEIN privilege on the schema is granted to PUBLIC.
- The optimizer can recommend indexes prior to creating the actual index.
- If an index specification is being defined for a data source table that has an index, the name of the index specification does not have to match the name of the index.
- The optimizer uses index specifications to improve access to the data source tables that the specifications apply to.
- The COLLECT STATISTICS options are not supported with declared temporary tables (SQLSTATE 42995).
- The COLLECT STATISTICS options are not supported if a nickname is specified (SQLSTATE 42601).
- When creating a unique index on a materialized query table (MQT), consider the implications of this uniqueness constraint on other processing. For example, if the unique index does not match the uniqueness attributes of the materialized query for a refresh immediate system-maintained MQT, it will be the index over the MQT that catches uniqueness violations during insert or update operations

against the underlying table. In a similar scenario with a refresh deferred system-maintained MQT, the REFRESH TABLE statement would fail. In general, a unique index on an MQT should match uniqueness constraints that already exist for data based on the underlying table or that can be inferred from the query associated with the MQT.

Examples:

Example 1: Create an index named UNIQUE_NAM on the PROJECT table. The purpose of the index is to ensure that there are not two entries in the table with the same value for project name (PROJNAME). The index entries are to be in ascending order.

```
CREATE UNIQUE INDEX UNIQUE_NAM
   ON PROJECT(PROJNAME)
```

Example 2: Create an index named JOB_BY_DPT on the EMPLOYEE table. Arrange the index entries in ascending order by job title (JOB) within each department (WORKDEPT).

```
CREATE INDEX JOB_BY_DPT
   ON EMPLOYEE (WORKDEPT, JOB)
```

Example 3: The nickname EMPLOYEE references a data source table called CURRENT_EMP. After this nickname was created, an index was defined on CURRENT_EMP. The columns chosen for the index key were WORKDEBT and JOB. Create an index specification that describes this index. Through this specification, the optimizer will know that the index exists and what its key is. With this information, the optimizer can improve its strategy to access the table.

```
CREATE UNIQUE INDEX JOB_BY_DEPT
  ON EMPLOYEE (WORKDEPT, JOB)
  SPECIFICATION ONLY
```

Example 4: Create an extended index type named SPATIAL_INDEX on a structured type column location. The description in index extension GRID_EXTENSION is used to maintain SPATIAL_INDEX. The literal is given to GRID_EXTENSION to create the index grid size.

```
CREATE INDEX SPATIAL_INDEX ON CUSTOMER (LOCATION)
  EXTEND USING (GRID_EXTENSION (x'000100100010001000400010'))
```

Example 5: Create an index named IDX1 on a table named TAB1, and collect basic index statistics on index IDX1.

```
CREATE INDEX IDX1 ON TAB1 (col1) COLLECT STATISTICS
```

Example 6: Create an index named IDX2 on a table named TAB1, and collect detailed index statistics on index IDX2.

```
CREATE INDEX IDX2 ON TAB1 (col2) COLLECT DETAILED STATISTICS
```

Example 7: Create an index named IDX3 on a table named TAB1, and collect detailed index statistics on index IDX3 using sampling.

```
CREATE INDEX IDX3 ON TAB1 (col3) COLLECT SAMPLED DETAILED STATISTICS
```

Related concepts:

- "Options on the CREATE INDEX statement" on page 150
- "Index specifications in a federated system" in the *Federated Systems Guide*
- "Index specifications" in the *Federated Systems Guide*

Related reference:

- "CREATE TABLE" on page 591
- "Interaction of triggers and constraints" in the *SQL Reference, Volume 1*
- "CREATE INDEX EXTENSION statement" in the *SQL Reference, Volume 2*

Related samples:

- "dbstat.sqb -- Reorganize table and run statistics (MF COBOL)"
- "TbGenCol.java -- How to use generated columns (JDBC)"

CREATE METHOD

This statement is used to associate a method body with a method specification that is already part of the definition of a user-defined structured type.

Invocation:

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization:

The privileges held by the authorization ID of the statement must include at least one of the following:

- SYSADM or DBADM authority
- CREATEIN privilege on the schema of the structured type referred to in the CREATE METHOD statement
- The DEFINER of the structured type referred to in the CREATE METHOD statement.

To associate an external method body with its method specification, the authorization ID of the statement must also include at least one of the following:

- SYSADM or DBADM authority
- CREATE_EXTERNAL_ROUTINE authority on the database.

When creating an SQL method, the privileges held by the authorization ID of the statement must also include, for each table, view, or nickname identified in any fullselect:

- CONTROL privilege on that table, view, or nickname, or
- SELECT privilege on that table, view, or nickname

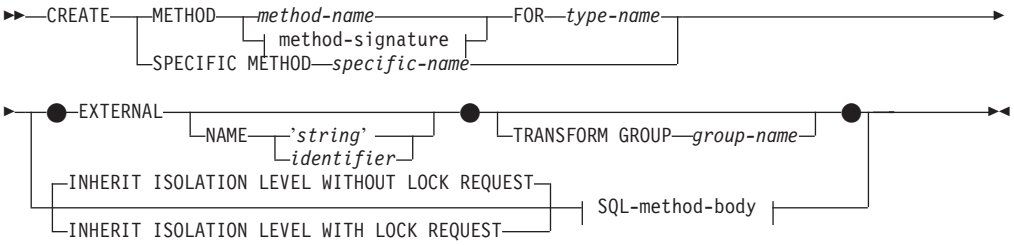
If the definer of an SQL method can only create the method because the definer has SYSADM authority, the definer is granted implicit DBADM authority for the purpose of creating the method.

Group privileges other than PUBLIC are not considered for any table or view specified in the CREATE METHOD statement.

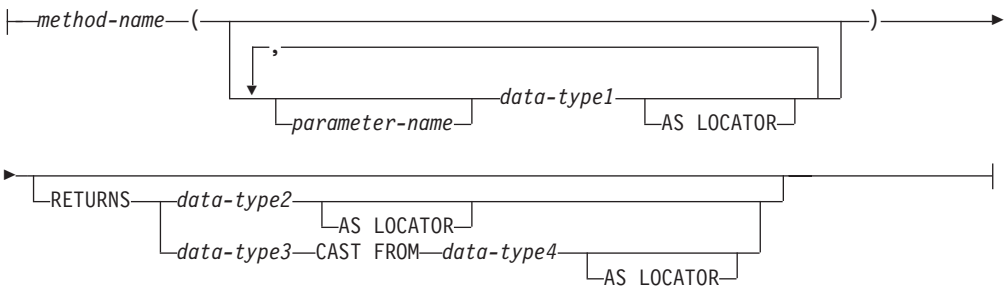
Authorization requirements of the data source for the table or view referenced by the nickname are applied when the method is invoked. The authorization ID of the connection may be mapped to a different remote authorization ID.

If the authorization ID has insufficient authority to perform the operation, an error is raised (SQLSTATE 42502).

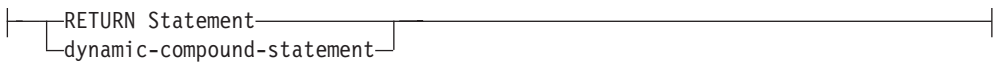
Syntax:



method-signature:



SQL-method-body:



Description:

METHOD
Identifies an existing method specification that is associated with a user-defined structured type. The method-specification can be identified through one of the following means:

method-name
Names the method specification for which a method body is being defined. The implicit schema is the schema of the subject type (*type-name*). There must be only one method specification for *type-name* that has this *method-name* (SQLSTATE 42725).

method-signature
Provides the method signature which uniquely identifies the method to be defined. The method signature must match the method specification that was provided on the CREATE TYPE or ALTER TYPE statement (SQLSTATE 42883).

method-name
Names the method specification for which a method body is being defined. The implicit schema is the schema of the subject type (*type-name*).

parameter-name
Identifies the parameter name. If parameter names are provided in

the method signature, they must be exactly the same as the corresponding parts of the matching method specification. Parameter names are supported in this statement solely for documentation purposes.

data-type1
Specifies the data type of each parameter.

AS LOCATOR
For the LOB types or distinct types which are based on a LOB type, the AS LOCATOR clause can be added.

RETURNS
This clause identifies the output of the method. If a RETURNS clause is provided in the method signature, it must be exactly the same as the corresponding part of the matching method specification on CREATE TYPE. The RETURNS clause is supported in this statement solely for documentation purposes.

data-type2
Specifies the data type of the output.

AS LOCATOR
For LOB types or distinct types which are based on LOB types, the AS LOCATOR clause can be added. This indicates that a LOB locator is to be returned by the method instead of the actual value.

data-type3 **CAST FROM** *data-type4*
This form of the RETURNS clause is used to return a different data type to the invoking statement from the data type that was returned by the function code.

AS LOCATOR
For LOB types or distinct types which are based on LOB types, the AS LOCATOR clause can be used to indicate that a LOB locator is to be returned from the method instead of the actual value.

FOR *type-name*
Names the type for which the specified method is to be associated. The name must identify a type already described in the catalog. (SQLSTATE 42704) In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified object names.

SPECIFIC METHOD *specific-name*
Identifies the particular method, using the specific name either specified or defaulted to at CREATE TYPE time. The specific-name must identify a method specification in the named or implicit schema; otherwise, an error is raised (SQLSTATE 42704).

EXTERNAL
This clause indicates that the CREATE METHOD statement is being used to register a method, based on code written in an external programming language, and adhering to the documented linkage conventions and interface. The matching method-specification in CREATE TYPE must specify a LANGUAGE other than SQL. When the method is invoked, the subject of the method is passed to the implementation as an implicit first parameter.

If the NAME clause is not specified, "NAME *method-name*" is assumed.

NAME
This clause identifies the name of the user-written code which implements the method being defined.

'*string*'
The 'string' option is a string constant with a maximum of 254 characters. The format used for the string is dependent on the LANGUAGE specified. For more information on the specific language conventions, see "CREATE FUNCTION (External Scalar)".

identifier
This identifier specified is an SQL identifier. The SQL identifier is used as the library-id in the string. Unless it is a delimited identifier, the identifier is folded to upper case. If the identifier is qualified with a schema name, the schema name portion is ignored. This form of NAME can only be used with LANGUAGE C (as defined in the method-specification on CREATE TYPE).

TRANSFORM GROUP *group-name*
Indicates the transform group that is used for user-defined structured type transformations when invoking the method. A transform is required since the method definition includes a user-defined structured type.

It is strongly recommended that a transform group name be specified; if this clause is not specified, the default group-name used is DB2_FUNCTION. If the specified (or default) group-name is not defined for a referenced structured type, an error results (SQLSTATE 42741). Likewise, if a required FROM SQL or TO SQL transform function is not defined for the given group-name and structured type, an error results (SQLSTATE 42744).

INHERIT ISOLATION LEVEL WITHOUT LOCK REQUEST or **INHERIT ISOLATION LEVEL WITH LOCK REQUEST**
Specifies whether or not a lock request can be associated with the isolation-clause of the statement when the method inherits the isolation level of the statement that invokes the method. The default is INHERIT ISOLATION LEVEL WITHOUT LOCK REQUEST.

INHERIT ISOLATION LEVEL WITHOUT LOCK REQUEST
Specifies that, as the method inherits the isolation level of the invoking statement, it cannot be invoked in the context of an SQL statement which includes a lock-request-clause as part of a specified isolation-clause (SQLSTATE 42601).

INHERIT ISOLATION LEVEL WITH LOCK REQUEST
Specifies that, as the method inherits the isolation level of the invoking statement, it also inherits the specified lock-request-clause.

SQL-method-body
The SQL-method-body defines how the method is implemented if the method specification in CREATE TYPE is LANGUAGE SQL.

The SQL-method-body must comply with the following parts of method specification:

- DETERMINISTIC or NOT DETERMINISTIC (SQLSTATE 428C2)
- EXTERNAL ACTION or NO EXTERNAL ACTION (SQLSTATE 428C2)
- CONTAINS SQL or READS SQL DATA (SQLSTATE 42985)

Parameter names can be referenced in the SQL-method-body. The subject of the method is passed to the method implementation as an implicit first parameter named SELF.

For additional details, see "Compound SQL (Dynamic)" and "RETURN Statement".

Rules:

- The method specification must be previously defined using the CREATE TYPE or ALTER TYPE statement before CREATE METHOD can be used (SQLSTATE 42723).
- If the method being created is an overriding method, those packages that are dependent on the following methods are invalidated:
  - The original method
  - Other overriding methods that have as their subject a supertype of the method being created

Notes:

- If the method allows SQL, the external program must not attempt to access any federated objects (SQLSTATE 55047).
- ***Privileges***

  The definer of a method always receives the EXECUTE privilege on the method, as well as the right to drop the method.

  If an EXTERNAL method is created, the definer of the method always receives the EXECUTE privilege WITH GRANT OPTION.

  If an SQL method is created, the definer of the method will only be given the EXECUTE privilege WITH GRANT OPTION on the method when the definer has WITH GRANT OPTION on all privileges required to define the method, or if the definer has SYSADM or DBADM authority. The definer of an SQL method only acquires privileges if the privileges from which they are derived exist at the time the method is created. The definer must have these privileges either directly, or because PUBLIC has the privileges. Privileges held by groups of which the method definer is a member are not considered. When using the method, the connected user's authorization ID must have the valid privileges on the table or view that the nickname references at the data source.
- ***Table access restrictions***

  If a method is defined as READS SQL DATA, no statement in the method can access a table that is being modified by the statement which invoked the method (SQLSTATE 57053).

Examples:

Example 1:

```
CREATE METHOD BONUS (RATE DOUBLE)
  FOR EMP
  RETURN SELF..SALARY * RATE
```

Example 2:

```
CREATE METHOD SAMEZIP (addr address_t)
  RETURNS INTEGER
  FOR address_t
  RETURN
    (CASE
```

```
    WHEN (self..zip = addr..zip)
      THEN 1
    ELSE 0
  END)
```

Example 3:

```
CREATE METHOD DISTANCE (address_t)
  FOR address_t
  EXTERNAL NAME 'addresslib!distance'
  TRANSFORM GROUP func_group
```

Related reference:

- "RETURN statement" in the *SQL Reference, Volume 2*
- "Compound SQL (Dynamic) statement" in the *SQL Reference, Volume 2*
- "CREATE FUNCTION (External Scalar) statement" in the *SQL Reference, Volume 2*

CREATE PROCEDURE

The CREATE PROCEDURE statement defines a procedure with an application server.

There are two different types of procedures that can be created using this statement. Each of these is described separately.

- External. The procedure body is written in a programming language. The external executable is referenced by a procedure defined with an application server, along with various attributes of the procedure.
- SQL. The procedure body is written in SQL. The procedure body is defined with an application server along with various attributes of the procedure.

Related reference:

- "CREATE PROCEDURE (External) statement" in the *SQL Reference, Volume 2*
- "CREATE PROCEDURE (SQL) statement" in the *SQL Reference, Volume 2*

CREATE SCHEMA

The CREATE SCHEMA statement defines a schema. It is also possible to create some objects and grant privileges on objects within the statement.

Invocation:

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization:

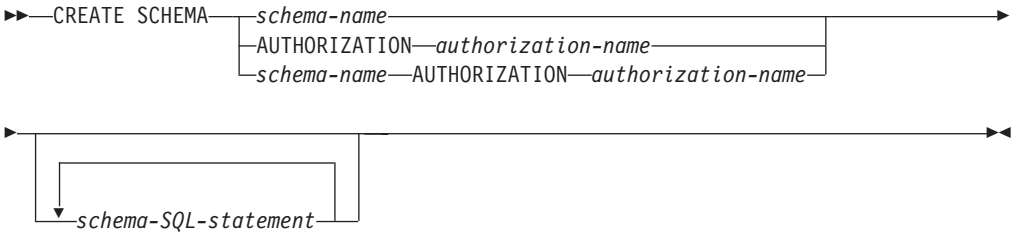
An authorization ID that holds SYSADM or DBADM authority can create a schema with any valid *schema-name* or *authorization-name*.

An authorization ID that does not hold SYSADM or DBADM authority can only create a schema with a *schema-name* or *authorization-name* that matches the authorization ID of the statement.

If the statement includes any *schema-SQL-statement*s the privileges held by the *authorization-name* (if not specified, it defaults to the authorization ID of the statement) must include at least one of the following:

- The privileges required to perform each of the *schema-SQL-statement*s
- SYSADM or DBADM authority.

Syntax:



Description:

schema-name
Names the schema. The name must not identify a schema already described in the catalog (SQLSTATE 42710). The name cannot begin with 'SYS' (SQLSTATE 42939). The owner of the schema is the authorization ID that issued the statement.

AUTHORIZATION *authorization-name*
Identifies the user who is the owner of the schema. The value of *authorization-name* is also used to name the schema. The *authorization-name* must not identify a schema already described in the catalog (SQLSTATE 42710).

schema-name **AUTHORIZATION** *authorization-name*
Identifies a schema called *schema-name*, whose owner is *authorization-name*. The *schema-name* must not identify a schema already described in the catalog (SQLSTATE 42710). The *schema-name* cannot begin with 'SYS' (SQLSTATE 42939).

schema-SQL-statement
SQL statements that can be included as part of the CREATE SCHEMA statement are:

- CREATE TABLE statement, excluding typed tables and materialized query tables
- CREATE VIEW statement, excluding typed views
- CREATE INDEX statement
- COMMENT statement
- GRANT statement

Notes:

- The owner of the schema is determined as follows:
  - If an AUTHORIZATION clause is specified, the specified *authorization-name* is the schema owner
  - If an AUTHORIZATION clause is not specified, the authorization ID that issued the CREATE SCHEMA statement is the schema owner.
- The schema owner is assumed to be a user (not a group).

- When the schema is explicitly created with the CREATE SCHEMA statement, the schema owner is granted CREATEIN, DROPIN, and ALTERIN privileges on the schema with the ability to grant these privileges to other users.
- The definer of any object created as part of the CREATE SCHEMA statement is the schema owner. The schema owner is also the grantor for any privileges granted as part of the CREATE SCHEMA statement.
- Unqualified object names in any SQL statement within the CREATE SCHEMA statement are implicitly qualified by the name of the created schema.
- If the CREATE statement contains a qualified name for the object being created, the schema name specified in the qualified name must be the same as the name of the schema being created (SQLSTATE 42875). Any other objects referenced within the statements may be qualified with any valid schema name.
- It is recommended not to use ″SESSION″ as a schema name. Since declared temporary tables must be qualified by ″SESSION″, it is possible to have an application declare a temporary table with a name identical to that of a persistent table. An SQL statement that references a table with the schema name ″SESSION″ will resolve (at statement compile time) to the declared temporary table rather than a persistent table with the same name. Since an SQL statement is compiled at different times for static embedded and dynamic embedded SQL statements, the results depend on when the declared temporary table is defined. If persistent tables, views or aliases are not defined with a schema name of ″SESSION″, these issues do not require consideration.

Examples:

Example 1: As a user with DBADM authority, create a schema called RICK with the user RICK as the owner.

```
CREATE SCHEMA RICK AUTHORIZATION RICK
```

Example 2: Create a schema that has an inventory part table and an index over the part number. Give authority on the table to user JONES.

```
CREATE SCHEMA INVENTRY

  CREATE TABLE PART (PARTNO   SMALLINT NOT NULL,
                     DESCR    VARCHAR(24),
                     QUANTITY INTEGER)

  CREATE INDEX PARTIND ON PART (PARTNO)

  GRANT ALL ON PART TO JONES
```

Example 3: Create a schema called PERS with two tables that each have a foreign key that references the other table. This is an example of a feature of the CREATE SCHEMA statement that allows such a pair of tables to be created without the use of the ALTER TABLE statement.

```
CREATE SCHEMA PERS

  CREATE TABLE ORG (DEPTNUMB  SMALLINT NOT NULL,
                    DEPTNAME VARCHAR(14),
                    MANAGER  SMALLINT,
                    DIVISION VARCHAR(10),
                    LOCATION VARCHAR(13),
                    CONSTRAINT PKEYDNO
                      PRIMARY KEY (DEPTNUMB),
                    CONSTRAINT FKEYMGR
                      FOREIGN KEY (MANAGER)
                      REFERENCES STAFF (ID) )
```

```
CREATE TABLE STAFF (ID        SMALLINT NOT NULL,
                    NAME      VARCHAR(9),
                    DEPT      SMALLINT,
                    JOB       VARCHAR(5),
                    YEARS     SMALLINT,
                    SALARY    DECIMAL(7,2),
                    COMM      DECIMAL(7,2),
                    CONSTRAINT PKEYID
                      PRIMARY KEY (ID),
                    CONSTRAINT FKEYDNO
                      FOREIGN KEY (DEPT)
                      REFERENCES ORG (DEPTNUMB) )
```

Related reference:

- "COMMENT" on page 565
- "CREATE INDEX" on page 575
- "CREATE TABLE" on page 591
- "CREATE VIEW" on page 656
- "GRANT (Table, View, or Nickname Privileges)" on page 718

CREATE TABLE

The CREATE TABLE statement defines a table. The definition must include its name and the names and attributes of its columns. The definition can include other attributes of the table, such as its primary key or check constraints.

To declare a global temporary table, use the DECLARE GLOBAL TEMPORARY TABLE statement.

Invocation:

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization:

The privileges held by the authorization ID of the statement must include at least one of the following:

- SYSADM or 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.

  If a subtable is being defined, the authorization ID must be the same as the definer of the root table of the table hierarchy.

To define a foreign key, the privileges held by the authorization ID of the statement must include one of the following on the parent table:

- REFERENCES privilege on the table
- REFERENCES privilege on each column of the specified parent key

- CONTROL privilege on the table
- SYSADM or DBADM authority.

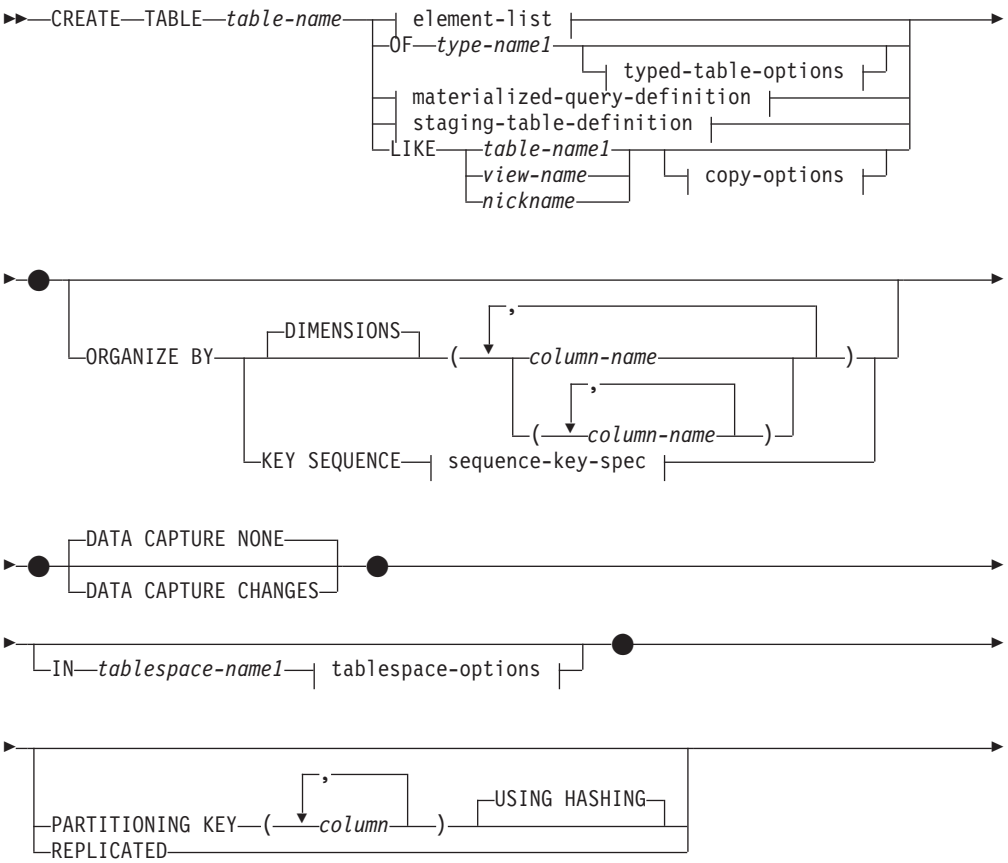
To define a materialized query table (using a fullselect) the privileges held by the authorization ID of the statement must include at least one of the following on each table or view identified in the fullselect:

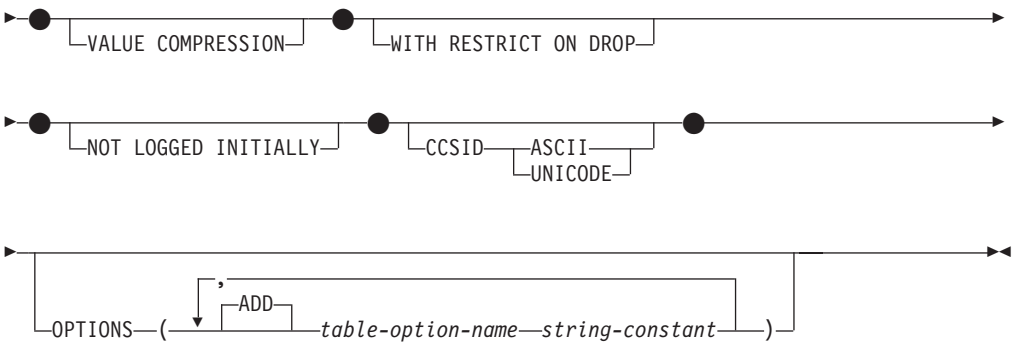
- SELECT privilege on the table or view and ALTER privilege if REFRESH DEFERRED or REFRESH IMMEDIATE is specified
- CONTROL privilege on the table or view
- SYSADM or DBADM authority.

To define a staging table associated with a materialized query table, the privileges held by the authorization ID of the statement must include at least one of the following:

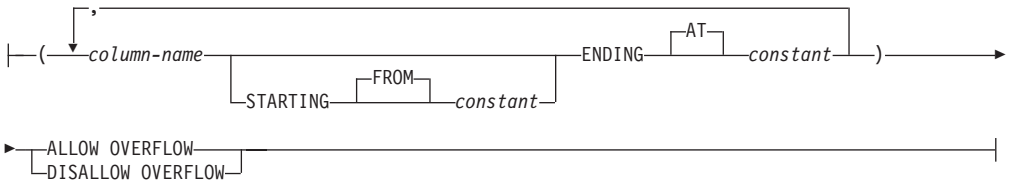
- CONTROL privilege or ALTER privilege on the materialized query table, and at least one of the following on each table or view identified in the fullselect of the materialized query table:
  - SELECT privilege and ALTER privilege on the table or view
  - CONTROL privilege on the table or view
- SYSADM or DBADM authority

Syntax:

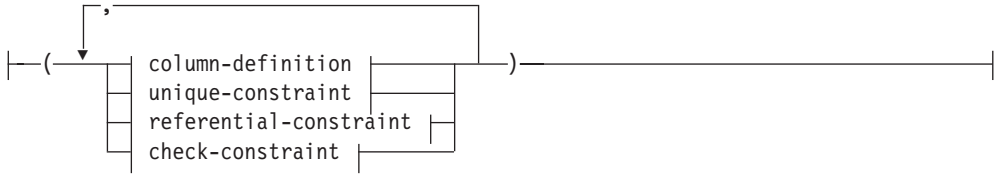




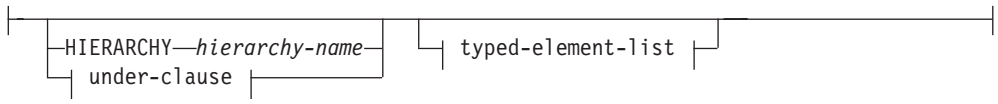
sequence-key-spec:



element-list:

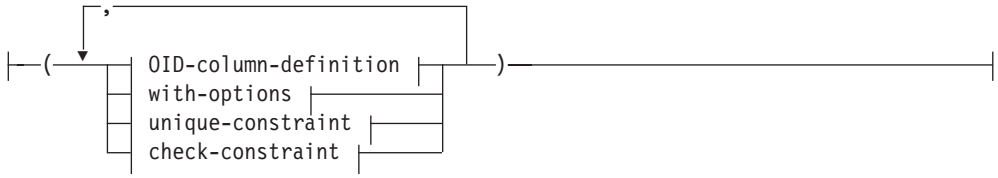


typed-table-options:

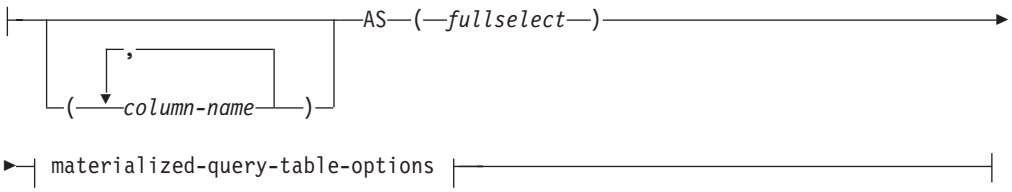


under-clause:

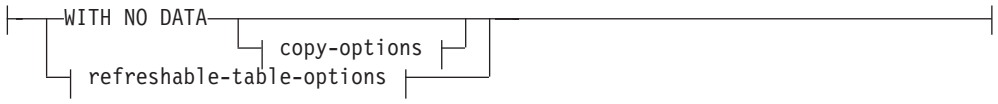
typed-element-list:



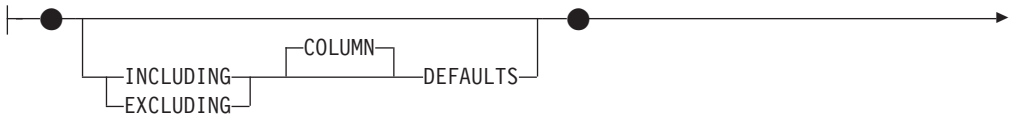
materialized-query-definition:

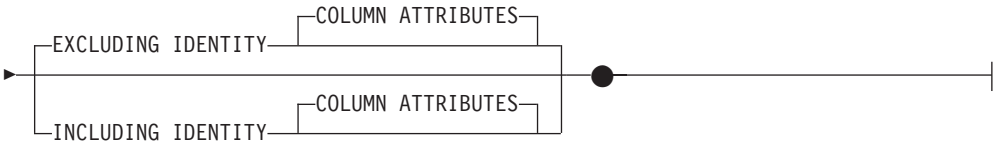


materialized-query-table-options:

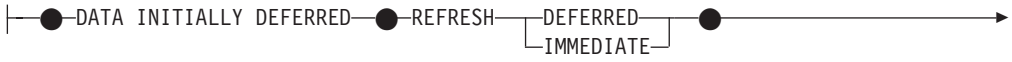


copy-options:

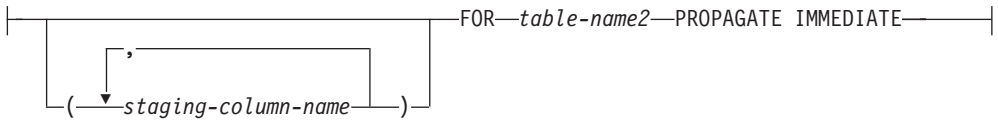




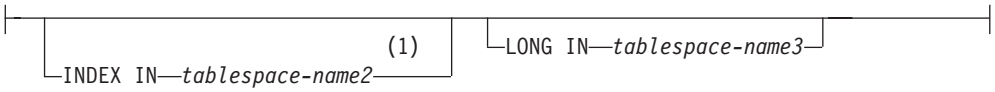
refreshable-table-options:



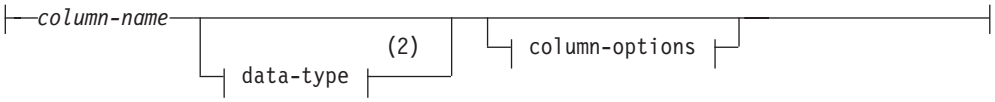
staging-table-definition:



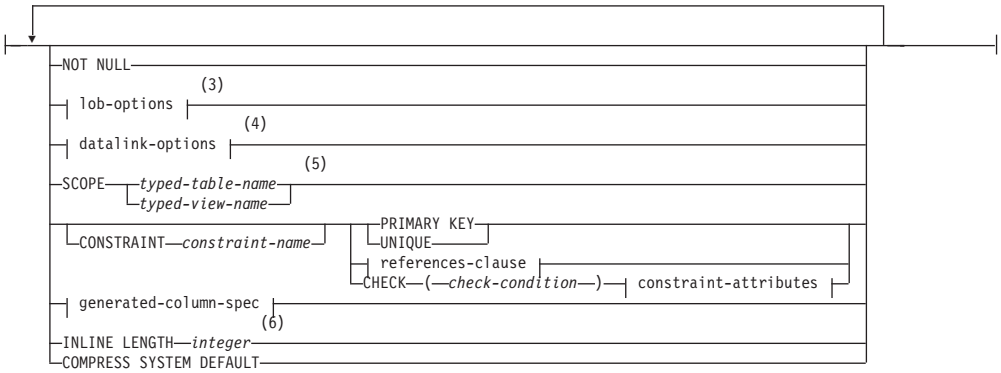
tablespace-options:



column-definition:



column-options:



Notes:

1 Specifying which table space will contain a table's index can only be done when the table is created.

2 If the first column-option chosen is a generated-column-spec with a generation-expression, then the data-type can be omitted. It will be determined from the resulting data type of the generation-expression.

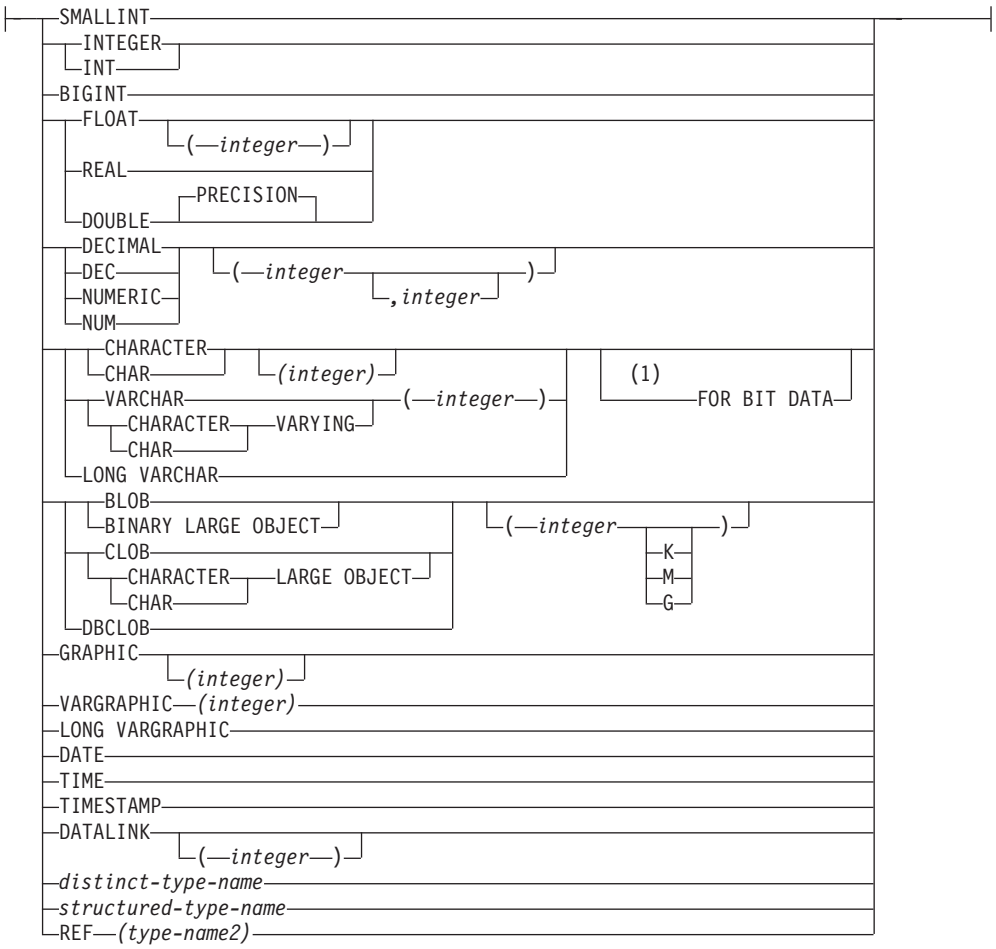
3 The lob-options clause only applies to large object types (BLOB, CLOB and DBCLOB) and distinct types based on large object types.

4 The datalink-options clause only applies to the DATALINK type and distinct types based on the DATALINK type.

5 The SCOPE clause only applies to the REF type.

6 INLINE LENGTH only applies to columns defined as structured types.

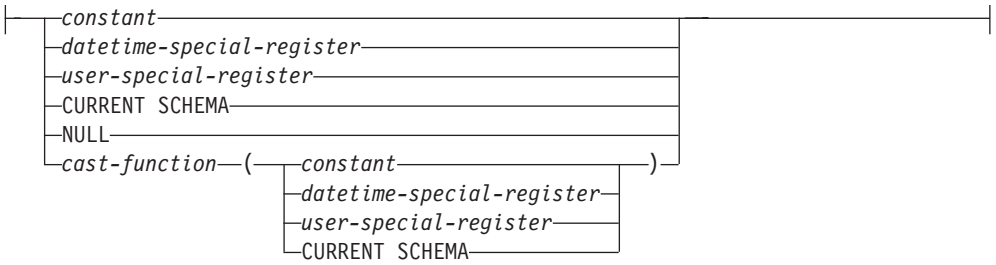
data-type:



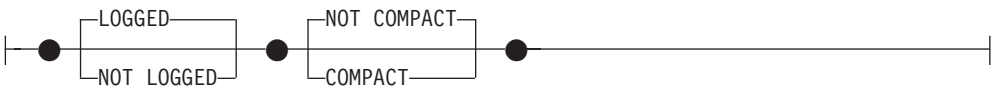
Notes:

1 The FOR BIT DATA clause can be specified in any order with the other column constraints that follow.

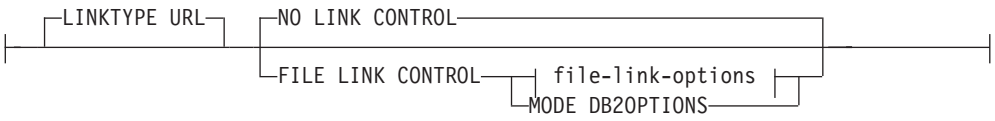
default-values:



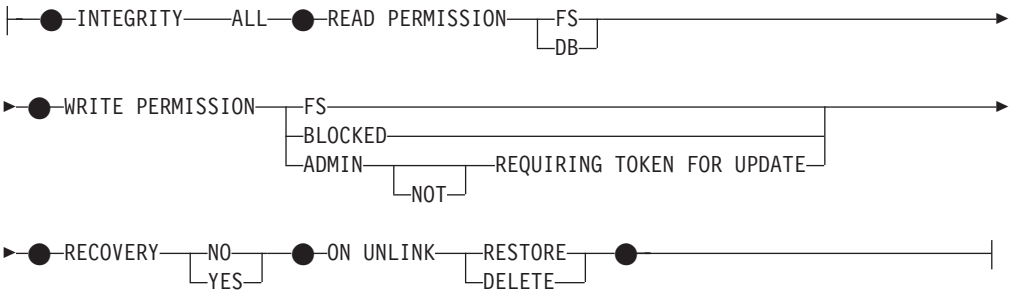
lob-options:



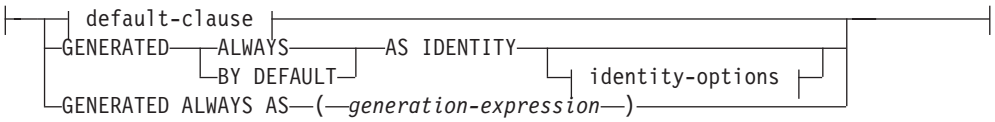
datalink-options:



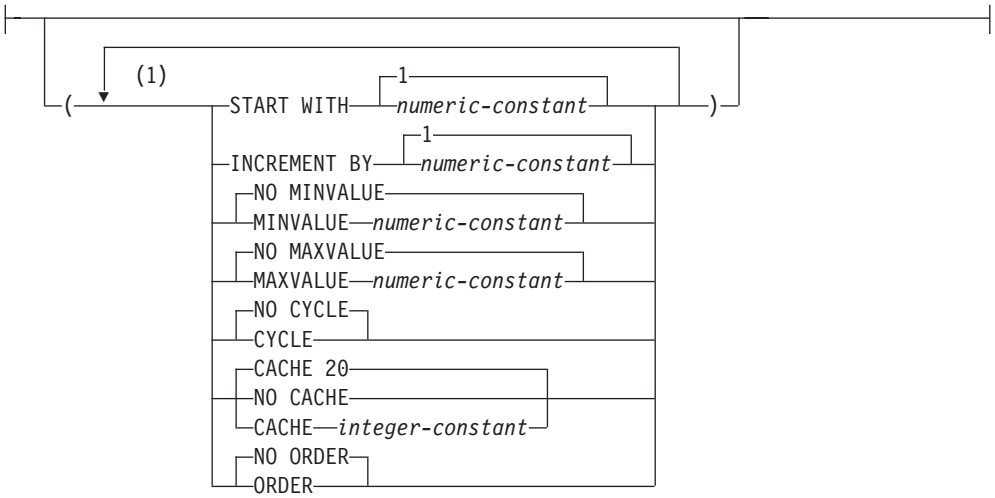
file-link-options:



generated-column-spec:



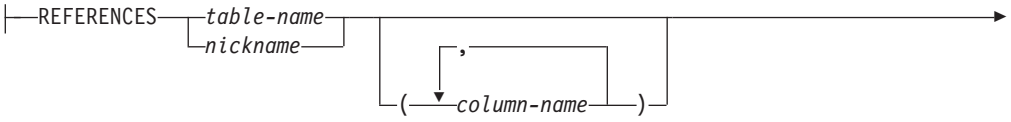
identity-options:

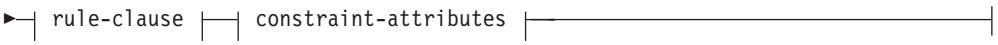


Notes:

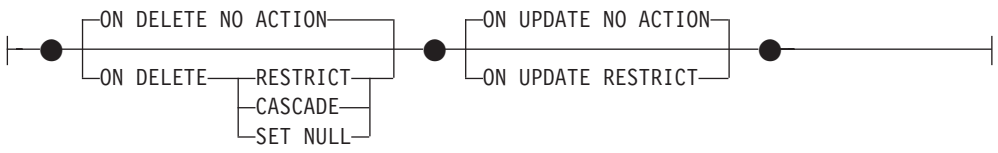
1 The same clause must not be specified more than once.

references-clause:

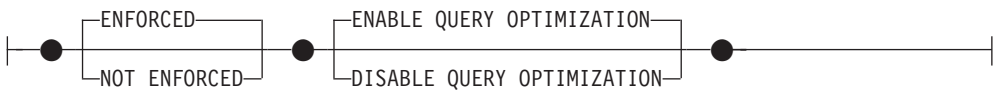




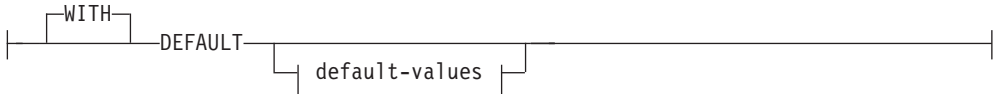
rule-clause:



constraint-attributes:



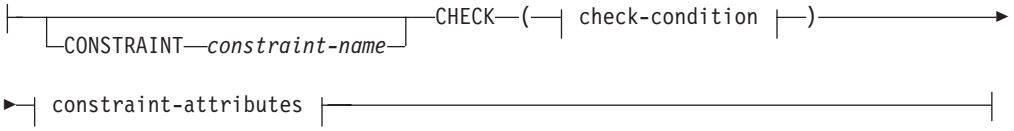
default-clause:



unique-constraint:

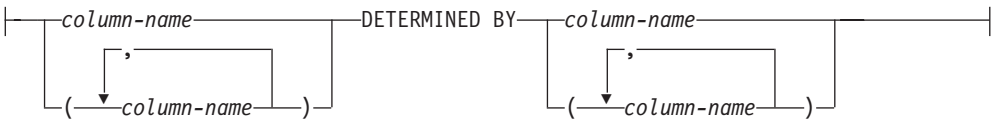
referential-constraint:

check-constraint:



check-condition:

functional-dependency:



OID-column-definition:

with-options:

Description:
System-maintained materialized query tables and user-maintained materialized query tables are referred to by the common term *materialized query table*, unless there is a need to identify each one separately.

table-name
Names the table. The name, including the implicit or explicit qualifier, must not identify a table, view, nickname, or alias described in the catalog. The schema name must not be SYSIBM, SYSCAT, SYSFUN, or SYSSTAT (SQLSTATE 42939).

OF *type-name1*
Specifies that the columns of the table are based on the attributes of the structured type identified by *type-name1*. If *type-name1* is specified without a schema name, the type name is resolved by searching the schemas on the SQL path (defined by the FUNCPATH preprocessing option for static SQL and by the CURRENT PATH register for dynamic SQL). The type name must be the name of an existing user-defined type (SQLSTATE 42704) and it must be an instantiable structured type (SQLSTATE 428DP) with at least one attribute (SQLSTATE 42997).

If UNDER is not specified, an object identifier column must be specified (refer to the *OID-column-definition*). This object identifier column is the first column of the table. The object ID column is followed by columns based on the attributes of *type-name1*.

HIERARCHY *hierarchy-name*
Names the hierarchy table associated with the table hierarchy. It is created at the same time as the root table of the hierarchy. The data for all subtables in the typed table hierarchy is stored in the hierarchy table. A hierarchy table cannot be directly referenced in SQL statements. A *hierarchy-name* is a *table-name*. The *hierarchy-name*, including the implicit or explicit schema name, must not identify a table, nickname, view, or alias described in the catalog. If the schema name is specified, it must be the same as the schema name of the table being created (SQLSTATE 428DQ). If this clause is omitted when defining the root table, a name is generated by the system consisting of the name of the table being created followed by a unique suffix such that the identifier is unique within the identifiers of the existing tables, views, aliases, and nicknames.

UNDER *supertable-name*
Indicates that the table is a subtable of *supertable-name*. The supertable must be

an existing table (SQLSTATE 42704) and the table must be defined using a structured type that is the immediate supertype of *type-name1* (SQLSTATE 428DB). The schema name of *table-name* and *supertable-name* must be the same (SQLSTATE 428DQ). The table identified by *supertable-name* must not have any existing subtable already defined using *type-name1* (SQLSTATE 42742).

The columns of the table include the object identifier column of the supertable with its type modified to be REF(*type-name1*), followed by columns based on the attributes of *type-name1* (remember that the type includes the attributes of its supertype). The attribute names cannot be the same as the OID column name (SQLSTATE 42711).

Other table options including table space, data capture, not logged initially and partitioning key options cannot be specified. These options are inherited from the supertable (SQLSTATE 42613).

INHERIT SELECT PRIVILEGES
Any user or group holding a SELECT privilege on the supertable will be granted an equivalent privilege on the newly created subtable. The subtable definer is considered to be the grantor of this privilege.

element-list
Defines the elements of a table. This includes the definition of columns and constraints on the table.

typed-element-list
Defines the additional elements of a typed table. This includes the additional options for the columns, the addition of an object identifier column (root table only), and constraints on the table.

materialized-query-definition
If the table definition is based on the result of a query, the table is a materialized query table based on the query.

column-name
Names the columns in the table. If a list of column names is specified, it must consist of as many names as there are columns in the result table of the fullselect. Each *column-name* must be unique and unqualified. If a list of column names is not specified, the columns of the table inherit the names of the columns of the result table of the fullselect.

A list of column names must be specified if the result table of the fullselect has duplicate column names of an unnamed column (SQLSTATE 42908). An unnamed column is a column derived from a constant, function, expression, or set operation that is not named using the AS clause of the select list.

AS
Introduces the query that is used for the definition of the table and that determines the data to be included in the table.

fullselect
Defines the query on which the table is based. The resulting column definitions are the same as those for a view defined with the same query.

Every select list element must have a name (use the AS clause for expressions). The *materialized-query-definition* defines attributes of the materialized query table. The option chosen also defines the contents of the fullselect as follows.

When WITH NO DATA is specified, any valid fullselect that does not reference a typed table or a typed view can be specified.

When REFRESH DEFERRED or REFRESH IMMEDIATE is specified, the fullselect cannot include (SQLSTATE 428EC):

- References to a materialized query table, declared temporary table, or typed table in any FROM clause
- References to a view where the fullselect of the view violates any of the listed restrictions on the fullselect of the materialized query table
- Expressions that are a reference type or DATALINK type (or distinct type based on these types)
- Functions that have any of the following attributes:
  - EXTERNAL ACTION
  - LANGUAGE SQL
  - CONTAINS SQL
  - READS SQL DATA
  - MODIFIES SQL DATA
- Functions that depend on physical characteristics (for example, DBPARTITIONNUM, HASHEDVALUE)
- Table or view references to system objects (Explain tables also should not be specified)
- Expressions that are a structured type or LOB type (or a distinct type based on a LOB type)

When REPLICATED is specified, the following restrictions apply:

- The GROUP BY clause is not allowed.
- The materialized query table must only reference a single table; that is, it cannot include a join.

When REFRESH IMMEDIATE is specified:

- The query must be a subselect, with the exception that UNION ALL is supported in the input table expression of a GROUP BY.
- The query cannot be recursive.
- The query cannot include:
  - References to a nickname
  - Functions that are not deterministic
  - Scalar fullselects
  - Predicates with fullselects
  - Special registers
  - SELECT DISTINCT
- If the FROM clause references more than one table or view, it can only define an inner join without using the explicit INNER JOIN syntax.
- When a GROUP BY clause is specified, the following considerations apply:
  - The supported column functions are SUM, COUNT, COUNT_BIG and GROUPING (without DISTINCT). The select list must contain a COUNT(*) or COUNT_BIG(*) column. If the materialized query table select list contains SUM(X), where X is a nullable argument, the materialized query table must also have COUNT(X) in its select list. These column functions cannot be part of any expressions.
  - A HAVING clause is not allowed.

  - If in a multiple partition database partition group, the partitioning key must be a subset of the GROUP BY items.
- The materialized query table must not contain duplicate rows, and the following restrictions specific to this uniqueness requirement apply, depending upon whether or not a GROUP BY clause is specified.
  - When a GROUP BY clause is specified, the following uniqueness-related restrictions apply:
    - All GROUP BY items must be included in the select list.
    - When the GROUP BY contains GROUPING SETS, CUBE, or ROLLUP, the GROUP BY items and associated GROUPING column functions in the select list must form a unique key of the result set. Thus, the following restrictions must be satisfied:
      - No grouping sets can be repeated. For example, `ROLLUP(X,Y),X` is not allowed, because it is equivalent to `GROUPING SETS((X,Y),(X),(X))`.
      - If X is a nullable GROUP BY item that appears within GROUPING SETS, CUBE, or ROLLUP, then GROUPING(X) must appear in the select list.
  - When a GROUP BY clause is not specified, the following uniqueness-related restrictions apply:
    - The materialized query table's uniqueness requirement is achieved by deriving a unique key for the materialized view from one of the unique key constraints defined in each of the underlying tables. Therefore, the underlying tables must have at least one unique key constraint defined on them, and the columns of these keys must appear in the select list of the materialized query table definition.
- When MAINTAINED BY FEDERATED_TOOL is specified, only references to nicknames are allowed in a FROM clause.

When REFRESH DEFERRED is specified, and the materialized query table is created with the intention of providing it with an associated staging table in a later statement, the fullselect of the materialized query table must follow the same restrictions and rules as a fullselect used to create a materialized query table with the REFRESH IMMEDIATE option.

A materialized query table whose fullselect contains a GROUP BY clause is summarizing data from the tables referenced in the fullselect. Such a materialized query table is also known as a *summary table*. A summary table is a specialized type of materialized query table.

WITH NO DATA

The query is used only to define the table. The table is not populated using the results of query and the REFRESH TABLE statement cannot be used. When the CREATE TABLE statement is completed, the table is no longer considered a materialized query table.

The columns of the table are defined based on the definitions of the columns that result from the fullselect. If the fullselect references a single table in the FROM clause, select list items that are columns of that table are defined using the column name, data type, and nullability characteristic of the referenced table.

refreshable-table-options

Define the refreshable options of the materialized query table attributes.

DATA INITIALLY DEFERRED
Data is not inserted into the table as part of the CREATE TABLE statement. A REFRESH TABLE statement specifying the *table-name* is used to insert data into the table.

REFRESH
Indicates how the data in the table is maintained.

DEFERRED
The data in the table can be refreshed at any time using the REFRESH TABLE statement. The data in the table only reflects the result of the query as a snapshot at the time the REFRESH TABLE statement is processed. System-maintained materialized query tables defined with this attribute do not allow INSERT, UPDATE, or DELETE statements (SQLSTATE 42807). User-maintained materialized query tables defined with this attribute do allow INSERT, UPDATE, or DELETE statements.

IMMEDIATE
The changes made to the underlying tables as part of a DELETE, INSERT, or UPDATE are cascaded to the materialized query table. In this case, the content of the table, at any point-in-time, is the same as if the specified *subselect* is processed. Materialized query tables defined with this attribute do not allow INSERT, UPDATE, or DELETE statements (SQLSTATE 42807).

ENABLE QUERY OPTIMIZATION
The materialized query table can be used for query optimization under appropriate circumstances.

DISABLE QUERY OPTIMIZATION
The materialized query table will not be used for query optimization. The table can still be queried directly.

MAINTAINED BY
Specifies whether the data in the materialized query table is maintained by the system, user, or replication tool. The default is SYSTEM.

SYSTEM
Specifies that the data in the materialized query table is maintained by the system.

USER
Specifies that the data in the materialized query table is maintained by the user. The user is allowed to perform update, delete, or insert operations against user-maintained materialized query tables. The REFRESH TABLE statement, used for system-maintained materialized query tables, cannot be invoked against user-maintained materialized query tables. Only a REFRESH DEFERRED materialized query table can be defined as MAINTAINED BY USER.

FEDERATED_TOOL
Specifies that the data in the materialized query table is maintained by the replication tool. The REFRESH TABLE statement, used for system-maintained materialized query tables, cannot be invoked against federated_tool-maintained materialized query tables. Only a REFRESH DEFERRED materialized query table can be defined as MAINTAINED BY FEDERATED_TOOL.

staging-table-definition
Defines the query supported by the staging table indirectly through an associated materialized query table. The underlying tables of the materialized

query table are also the underlying tables for its associated staging table. The staging table collects changes that need to be applied to the materialized query table to synchronize it with the contents of the underlying tables.

staging-column-name
Names the columns in the staging table. If a list of column names is specified, it must consist of *two* more names than there are columns in the materialized query table for which the staging table is defined. If the materialized query table is a replicated materialized query table, or the query defining the materialized query table does not contain a GROUP BY clause, the list of column names must consist of *three* more names than there are columns in the materialized query table for which the staging table is defined. Each column name must be unique and unqualified. If a list of column names is not specified, the columns of the table inherit the names of the columns of the associated materialized query table. The additional columns are named GLOBALTRANSID and GLOBALTRANSTIME, and if a third column is necessary, it is named OPERATIONTYPE.

Table 58. Extra Columns Appended in Staging Tables

Column Name	Data Type	Column Description
GLOBALTRANSID	CHAR(8) FOR BIT DATA	The global transaction ID for each propagated row
GLOBALTRANSTIME	CHAR(13) FOR BIT DATA	The timestamp of the transaction
OPERATIONTYPE	INTEGER	Operation for the propagated row, either insert, update, or delete.

A list of column names must be specified if any of the columns of the associated materialized query table duplicates any of the generated column names (SQLSTATE 42711).

FOR *table-name2*
Specifies the materialized query table that is used for the definition of the staging table. The name, including the implicit or explicit schema, must identify a materialized query table that exists at the current server defined with REFRESH DEFERRED. The fullselect of the associated materialized query table must follow the same restrictions and rules as a fullselect used to create a materialized query table with the REFRESH IMMEDIATE option.

The contents of the staging table can be used to refresh the materialized query table, by invoking the REFRESH TABLE statement, if the contents of the staging table are consistent with the associated materialized query table and the underlying source tables.

PROPAGATE IMMEDIATE
The changes made to the underlying tables as part of a delete, insert, or update operation are cascaded to the staging table in the same delete, insert, or update operation. If the staging table is not marked inconsistent, its content, at any point-in-time, is the delta changes to the underlying table since the last refresh materialized query table.

LIKE *table-name1* **or** *view-name* **or** *nickname*
Specifies that the columns of the table have exactly the same name and description as the columns of the identified table (*table-name1*), view

(*view-name*) or nickname (*nickname*). The name specified after LIKE must identify a table, view or nickname that exists in the catalog, or a declared temporary table. A typed table or typed view cannot be specified (SQLSTATE 428EC).

The use of LIKE is an implicit definition of *n* columns, where *n* is the number of columns in the identified table, view or nickname.

- If a table is identified, then the implicit definition includes the column name, data type and nullability characteristic of each of the columns of *table-name1*. If EXCLUDING COLUMN DEFAULTS is not specified, then the column default is also included.
- If a view is identified, then the implicit definition includes the column name, data type, and nullability characteristic of each of the result columns of the fullselect defined in *view-name*.
- If a nickname is identified, then the implicit definition includes the column name, data type, and nullability characteristic of each column of *nickname*.

Column default and identity column attributes may be included or excluded, based on the copy-attributes clauses. The implicit definition does not include any other attributes of the identified table, view or nickname. Thus the new table does not have any unique constraints, foreign key constraints, triggers, or indexes. The table is created in the table space implicitly or explicitly specified by the IN clause, and the table has any other optional clause only if the optional clause is specified.

copy-options
These options specify whether or not to copy additional attributes of the source result table definition (table, view or fullselect).

INCLUDING COLUMN DEFAULTS
Column defaults for each updatable column of the source result table definition are copied. Columns that are not updatable will not have a default defined in the corresponding column of the created table.

If LIKE *table-name* is specified and *table-name* identifies a base table or declared temporary table, then INCLUDING COLUMN DEFAULTS is the default.

EXCLUDING COLUMN DEFAULTS
Columns defaults are not copied from the source result table definition.

This clause is the default, except when LIKE *table-name* is specified and *table-name* identifies a base table or declared temporary table.

INCLUDING IDENTITY COLUMN ATTRIBUTES
Identity column attributes are copied from the source result table definition, if possible. It is possible to copy the identity column attributes, if the element of the corresponding column in the table, view, or fullselect is the name of a table column, or the name of a view column which directly or indirectly maps to the name of a base table column with the identity property. In all other cases, the columns of the new table will not get the identity property. For example:

- the select-list of the fullselect includes multiple instances of an identity column name (that is, selecting the same column more than once)
- the select list of the fullselect includes multiple identity columns (that is, it involves a join)
- the identity column is included in an expression in the select list
- the fullselect includes a set operation (union, except, or intersect).

EXCLUDING IDENTITY COLUMN ATTRIBUTES
Identity column attributes are not copied from the source result table definition.

ORGANIZE BY DIMENSIONS *(column-name,...)*
Specifies a dimension for each column or group of columns used to cluster the table data. The use of parentheses within the dimension list specifies that a group of columns is to be treated as one dimension. The DIMENSIONS keyword is optional.

A clustering block index is automatically maintained for each specified dimension, and a block index, consisting of all columns used in the clause, is maintained if none of the clustering block indexes includes them all. The set of columns used in the ORGANIZE BY clause must follow the rules for the CREATE INDEX statement.

Each column name specified in the ORGANIZE BY clause must be defined for the table (SQLSTATE 42703), and a dimension cannot occur more than once in the dimension list (SQLSTATE 42709).

Pages of the table are arranged in blocks of equal size, which is the extent size of the tablespace, and all rows of each block contain the same combination of dimension values.

ORGANIZE BY KEY SEQUENCE *sequence-key-spec*
Specifies that the table is organized in ascending key sequence with a fixed size based on the specified range of key sequence values. A table organized in this way is referred to as a *range-clustered table*. Each possible key value in the defined range has a predetermined location in the physical table. The storage required for a range-clustered table must be available when the table is created, and must be sufficient to contain the number of rows in the specified range multiplied by the row size (for details on determining the space requirement, see "Row Size" on page 638 and "Byte Counts" on page 638).

column-name
Specifies a column of the table that is included in the unique key that determines the sequence of the range-clustered table. The data type of the column must be SMALLINT, INTEGER, or BIGINT (SQLSTATE 42611), and the columns must be defined as NOT NULL (SQLSTATE 42831). The same column must not be identified more than once in the sequence key. The number of identified columns must not exceed 16 (SQLSTATE 54008).

A unique index entry will automatically be created in the catalog for the columns in the key sequence specified with ascending order for each column. The name of the index will be SQL, followed by a character timestamp (*yymmddhhmmssxxx*), with SYSIBM as the schema name. An actual index object is not created in storage, because the table organization is ordered by this key. If a primary key or a unique constraint is defined on the same columns as the range-clustered table sequence key, this same index entry is used for the constraint.

For the key sequence specification, a check constraint exists to reflect the column constraints. If the DISALLOW OVERFLOW clause is specified, the name of the check constraint will be RCT, and the check constraint is enforced. If the ALLOW OVERFLOW clause is specified, the name of the check constraint will be RCT_OFLOW, and the check constraint is not enforced.

STARTING FROM *constant*
Specifies the constant value at the low end of the range for *column-name*.

Values less than the specified constant are only allowed if the ALLOW OVERFLOW option is specified. If *column-name* is a SMALLINT or INTEGER column, the constant must be an INTEGER constant. If *column-name* is a BIGINT column, the constant must be an INTEGER or BIGINT constant (SQLSTATE 42821). If a starting constant is not specified, the default value is 1.

ENDING AT *constant*
Specifies the constant value at the high end of the range for *column-name*. Values greater than the specified constant are only allowed if the ALLOW OVERFLOW option is specified. The value of the ending constant must be greater than the starting constant. If *column-name* is a SMALLINT or INTEGER column, the constant must be an INTEGER constant. If *column-name* is a BIGINT column, the constant must be an INTEGER or BIGINT constant (SQLSTATE 42821).

ALLOW OVERFLOW
Specifies that the range-clustered table allows rows with key values that are outside of the defined range of values. When a range-clustered table is created to allow overflows, the rows with key values outside of the range are placed at the end of the defined range without any predetermined order. Operations involving these overflow rows are less efficient than operations on rows having key values within the defined range.

DISALLOW OVERFLOW
Specifies that the range-clustered table does not allow rows with key values that are not within the defined range of values (SQLSTATE 23513). Range-clustered tables that disallow overflows will always maintain all rows in ascending key sequence.

column-definition
Defines the attributes of a column.

column-name
Names a column of the table. The name cannot be qualified, and the same name cannot be used for more than one column of the table (SQLSTATE 42711).

A table may have the following:

- A 4K page size with a maximum of 500 columns, where the byte counts of the columns must not be greater than 4 005.
- An 8K page size with a maximum of 1 012 columns, where the byte counts of the columns must not be greater than 8 101.
- A 16K page size with a maximum of 1 012 columns, where the byte counts of the columns must not be greater than 16 293.
- A 32K page size with a maximum of 1 012 columns, where the byte counts of the columns must not be greater than 32 677.

For more details, see “Row Size” on page 638.

data-type
Is one of the types in the following list. Use:

SMALLINT
For a small integer.

INTEGER or **INT**
For a large integer.

BIGINT
For a big integer.

FLOAT(*integer***)**
For a single or double-precision floating-point number, depending on the value of the *integer*. The value of the integer must be in the range 1 through 53. The values 1 through 24 indicate single precision and the values 25 through 53 indicate double-precision.

You can also specify:

REAL	For single precision floating-point.
DOUBLE	For double-precision floating-point.
DOUBLE PRECISION	For double-precision floating-point.
FLOAT	For double-precision floating-point.

DECIMAL*(precision-integer, scale-integer)* or **DEC***(precision-integer, scale-integer)*
For a decimal number. The first integer is the precision of the number; that is, the total number of digits; it may range from 1 to 31. The second integer is the scale of the number; that is, the number of digits to the right of the decimal point; it may range from 0 to the precision of the number.

If precision and scale are not specified, the default values of 5,0 are used. The words **NUMERIC** and **NUM** can be used as synonyms for **DECIMAL** and **DEC**.

CHARACTER*(integer)* or **CHAR***(integer)* or **CHARACTER** or **CHAR**
For a fixed-length character string of length *integer*, which may range from 1 to 254. If the length specification is omitted, a length of 1 character is assumed.

VARCHAR*(integer)*, or **CHARACTER VARYING***(integer)*, or **CHAR VARYING***(integer)*
For a varying-length character string of maximum length *integer*, which may range from 1 to 32 672.

LONG VARCHAR
For a varying-length character string with a maximum length of 32 700.

FOR BIT DATA
Specifies that the contents of the column are to be treated as bit (binary) data. During data exchange with other systems, code page conversions are not performed. Comparisons are done in binary, irrespective of the database collating sequence.

BLOB or BINARY LARGE OBJECT*(integer [K | M | G])*
For a binary large object string of the specified maximum length in bytes.

The length may be in the range of 1 byte to 2 147 483 647 bytes.

If *integer* by itself is specified, that is the maximum length.

If *integer K* (in either upper- or lowercase) is specified, the maximum length is 1 024 times *integer*. The maximum value for *integer* is 2 097 152. If a multiple of K, M or G that calculates out to 2 147 483 648 is specified, the actual value used is 2 147 483 647 (or 2 gigabytes minus 1 byte), which is the maximum length for a LOB column.

If *integer M* is specified, the maximum length is 1 048 576 times *integer*. The maximum value for *integer* is 2 048.

If *integer G* is specified, the maximum length is 1 073 741 824 times *integer*. The maximum value for *integer* is 2.

If the length specification is omitted, a length of 1 048 576 (1 megabyte) is assumed.

To create BLOB strings greater than 1 gigabyte, you must specify the NOT LOGGED option.

Any number of spaces is allowed between the integer and K, M, or G, and a space is not required. For example, all of the following are valid:

```
BLOB(50K)    BLOB(50 K)    BLOB (50   K)
```

CLOB or CHARACTER (CHAR) LARGE OBJECT*(integer [K | M | G])*
For a character large object string of the specified maximum length in bytes.

The meaning of the *integer K | M | G* is the same as for BLOB.

If the length specification is omitted, a length of 1 048 576 (1 megabyte) is assumed.

To create CLOB strings greater than 1 gigabyte, you must specify the NOT LOGGED option.

It is not possible to specify the FOR BIT DATA clause for CLOB columns. However, a CHAR FOR BIT DATA string can be assigned to a CLOB column, and a CHAR FOR BIT DATA string can be concatenated with a CLOB string.

DBCLOB*(integer [K | M | G])*
For a double-byte character large object string of the specified maximum length in double-byte characters.

The meaning of the *integer K | M | G* is similar to that for BLOB. The differences are that the number specified is the number of double-byte characters, and that the maximum size is 1 073 741 823 double-byte characters.

If the length specification is omitted, a length of 1 048 576 double-byte characters is assumed.

To create DBCLOB strings greater than 1 gigabyte, you must specify the NOT LOGGED option.

GRAPHIC*(integer)*
For a fixed-length graphic string of length *integer* which may range from 1 to 127. If the length specification is omitted, a length of 1 is assumed.

VARGRAPHIC*(integer)*
For a varying-length graphic string of maximum length *integer*, which may range from 1 to 16 336.

LONG VARGRAPHIC
For a varying-length graphic string with a maximum length of 16 350.

DATE
For a date.

TIME
For a time.

TIMESTAMP
For a timestamp.

DATALINK or **DATALINK***(integer)*
For a link to a data file stored outside the database.

The column in the table consists of ″anchor values″ that contain the reference information that is required to establish and maintain the link to the external data as well as an optional comment.

The length of a DATALINK column is 200 bytes. If *integer* is specified, it must be 200. If the length specification is omitted, a length of 200 bytes is assumed.

A DATALINK value is an encapsulated value with a set of built-in scalar functions. There is a function called DLVALUE to create a DATALINK value, and functions called DLNEWCOPY, DLPREVIOUSCOPY, and DLREPLACECONTENT that can also be used to construct a DATALINK value under special circumstances. (DLVALUE should be used to construct a regular DATALINK value.) The following functions can be used to extract attributes from a DATALINK value.

- DLCOMMENT
- DLLINKTYPE
- DLURLCOMPLETE
- DLURLCOMPLETEONLY
- DLURLCOMPLETEWRITE
- DLURLPATH
- DLURLPATHONLY
- DLURLPATHWRITE
- DLURLSCHEME
- DLURLSERVER

A DATALINK column has the following restrictions:

- The column cannot be part of any index. Therefore, it cannot be included as a column of a primary key or unique constraint (SQLSTATE 42962).
- The column cannot be a foreign key of a referential constraint (SQLSTATE 42830).
- A default value (WITH DEFAULT) cannot be specified for the column. If the column is nullable, the default for the column is NULL (SQLSTATE 42894).

distinct-type-name
For a user-defined type that is a distinct type. If a distinct type name is specified without a schema name, the distinct type name is resolved by searching the schemas on the SQL path (defined by the FUNCPATH preprocessing option for static SQL and by the CURRENT PATH register for dynamic SQL).

If a column is defined using a distinct type, then the data type of the column is the distinct type. The length and the scale of the column are respectively the length and the scale of the source type of the distinct type.

If a column defined using a distinct type is a foreign key of a referential constraint, then the data type of the corresponding column of the primary key must have the same distinct type.

structured-type-name
For a user-defined type that is a structured type. If a structured type name is specified without a schema name, the structured type name is resolved by searching the schemas on the SQL path (defined by the FUNCPATH preprocessing option for static SQL, and by the CURRENT PATH register for dynamic SQL).

If a column is defined using a structured type, then the static data type of the column is the structured type. The column may include values with a dynamic type that is a subtype of *structured-type-name*.

A column defined using a structured type cannot be used in a primary key, unique constraint, foreign key, index key or partitioning key (SQLSTATE 42962).

If a column is defined using a structured type, and contains a reference-type attribute at any level of nesting, that reference-type attribute is unscoped. To use such an attribute in a dereference operation, it is necessary to specify a SCOPE explicitly, using a CAST specification.

If a column is defined using a structured type with an attribute of type DATALINK, or a distinct type sourced on DATALINK, this column can only be null. An attempt to use the constructor function for this type will return an error (SQLSTATE 428ED) and so no instance of this type can be inserted into the column.

REF (*type-name2***)**
For a reference to a typed table. If *type-name2* is specified without a schema name, the type name is resolved by searching the schemas on the SQL path (defined by the FUNCPATH preprocessing option for static SQL and by the CURRENT PATH register for dynamic SQL). The underlying data type of the column is based on the representation data type specified in the REF USING clause of the CREATE TYPE statement for *type-name2* or the root type of the data type hierarchy that includes *type-name2*.

column-options
Defines additional options related to columns of the table.

NOT NULL
Prevents the column from containing null values.

If NOT NULL is not specified, the column can contain null values, and its default value is either the null value or the value provided by the WITH DEFAULT clause.

lob-options
Specifies options for LOB data types.

LOGGED
Specifies that changes made to the column are to be written to the log. The data in such columns is then recoverable with database utilities (such as RESTORE DATABASE). LOGGED is the default.

LOBs greater than 1 gigabyte cannot be logged (SQLSTATE 42993).

NOT LOGGED
Specifies that changes made to the column are not to be logged.

NOT LOGGED has no effect on a commit or rollback operation; that is, the database's consistency is maintained even if a transaction is rolled back, regardless of whether or not the LOB value is logged. The implication of not logging is that during a roll forward operation, after a backup or load operation, the LOB data will be replaced by zeros for those LOB values that would have had log records replayed during the roll forward. During crash recovery, all committed changes and changes rolled back will reflect the expected results.

COMPACT
Specifies that the values in the LOB column should take up minimal disk space (free any extra disk pages in the last group used by the LOB value), rather than leave any leftover space at the end of the LOB storage area that might facilitate subsequent append operations. Note that storing data in this way may cause a performance penalty in any append (length-increasing) operations on the column.

NOT COMPACT
Specifies some space for insertions to assist in future changes to the LOB values in the column. This is the default.

datalink-options
Specifies the options associated with a DATALINK data type.

LINKTYPE URL
This defines the type of link as a Uniform Resource Locator (URL).

NO LINK CONTROL
Specifies that there will not be any check made to determine that the file exists. Only the syntax of the URL will be checked. There is no database manager control over the file.

FILE LINK CONTROL
Specifies that a check should be made for the existence of the file. Additional options may be used to give the database manager further control over the file.

file-link-options
Additional options to define the level of database manager control of the file link.

INTEGRITY
Specifies the level of integrity of the link between a DATALINK value and the actual file.

ALL
Any file specified as a DATALINK value is under the control of the database manager and may NOT be deleted or renamed using standard file system programming interfaces.

READ PERMISSION
Specifies how permission to read the file specified in a DATALINK value is determined.

FS The read access permission is determined by the file

system permissions. Such files can be accessed without retrieving the file name from the column.

DB
The read access permission is determined by the database. Access to the file will only be allowed by passing a valid file access token, returned on retrieval of the DATALINK value from the table, in the open operation.

WRITE PERMISSION
Specifies how permission to write to the file specified in a DATALINK value is determined.

FS The write access permission is determined by the file system permissions. Such files can be accessed without retrieving the file name from the column.

BLOCKED
Write access is blocked. The file cannot be directly updated through any interface. An alternative mechanism must be used to cause updates to the information. For example, the file is copied, the copy updated, and then the DATALINK value updated to point to the new copy of the file.

ADMIN
The write permission is determined by the Data Links Manager. Write access to the file will only be allowed by passing a valid write token, returned on retrieval of the DATALINK value from the table, by using the DLURLCOMPLETEWRITE or DLURLPATHWRITE scalar function, in the open operation. This value can be specified only when READ PERMISSION DB is also specified.

The access privilege for a given linked file is defined and maintained in the Data Links Manager.

Once a file is opened for write with a valid write token by a user (the updater), other users can still open the file for read by using a valid read or write token. However, only the same updater can repeatedly open the file for write with the same write token. The same updater also needs the same write token to perform any subsequent read operation.

REQUIRING TOKEN FOR UPDATE
To complete the file update, the write token used to open and modify the file must be contained in the file reference specified during invocation of the scalar functions DLNEWCOPY or DLPREVIOUSCOPY in the SQL UPDATE statement.

NOT REQUIRING TOKEN FOR UPDATE
To complete the file update, a write token is not required in the file reference specified during invocation of the scalar functions DLNEWCOPY or DLPREVIOUSCOPY in the SQL UPDATE statement.

RECOVERY
Specifies whether or not DB2 will support point in time recovery of files referenced by values in this column.

YES
DB2 will support point in time recovery of files referenced by values in this column. This value can only be specified when INTEGRITY ALL and WRITE PERMISSION BLOCKED or WRITE PERMISSION ADMIN are also specified.

NO
Specifies that point in time recovery will not be supported.

ON UNLINK
Specifies the action taken on a file when a DATALINK value is changed or deleted (unlinked). Note that this is not applicable when WRITE PERMISSION FS is used.

RESTORE
Specifies that when a file is unlinked, the Data Links File Manager will attempt to return the file to the owner with the permissions that existed at the time the file was linked. In the case where the user is no longer registered with the file server, the file is assigned to a special predefined "dfmunknown" user ID. This can only be specified when INTEGRITY ALL and WRITE PERMISSION BLOCKED or WRITE PERMISSION ADMIN are also specified.

DELETE
Specifies that the file will be deleted when it is unlinked. This can only be specified when READ PERMISSION DB and WRITE PERMISSION BLOCKED or WRITE PERMISSION ADMIN are also specified.

MODE DB2OPTIONS
This mode defines a set of default file link options. The defaults defined by DB2OPTIONS are:

- INTEGRITY ALL
- READ PERMISSION FS
- WRITE PERMISSION FS
- RECOVERY NO

ON UNLINK is not applicable since WRITE PERMISSION FS is used.

SCOPE
Identifies the scope of the reference type column.

A scope must be specified for any column that is intended to be used as the left operand of a dereference operator or as the argument of the DEREF function. Specifying the scope for a reference type column may be deferred to a subsequent ALTER TABLE statement to allow the target table to be defined, usually in the case of mutually referencing tables.

typed-table-name
The name of a typed table. The table must already exist or be the same as the name of the table being created (SQLSTATE 42704). The data type of *column-name* must be REF(*S*), where *S* is the type of *typed-table-name* (SQLSTATE 428DM). No checking is done of values assigned to *column-name* to ensure that the values actually reference existing rows in *typed-table-name*.

typed-view-name
The name of a typed view. The view must already exist or be the same as the name of the view being created (SQLSTATE 42704). The data type of *column-name* must be REF(*S*), where *S* is the type of *typed-view-name* (SQLSTATE 428DM). No checking is done of values assigned to *column-name* to ensure that the values actually reference existing rows in *typed-view-name*.

CONSTRAINT *constraint-name*
Names the constraint. A *constraint-name* must not identify a constraint that was already specified within the same CREATE TABLE statement. (SQLSTATE 42710).

If this clause is omitted, an 18-character identifier that is unique among the identifiers of existing constraints defined on the table is generated by the system. (The identifier consists of ″SQL″ followed by a sequence of 15 numeric characters generated by a timestamp-based function.)

When used with a PRIMARY KEY or UNIQUE constraint, the *constraint-name* may be used as the name of an index that is created to support the constraint.

PRIMARY KEY
This provides a shorthand method of defining a primary key composed of a single column. Thus, if PRIMARY KEY is specified in the definition of column C, the effect is the same as if the PRIMARY KEY(C) clause is specified as a separate clause.

A primary key cannot be specified if the table is a subtable (SQLSTATE 429B3) since the primary key is inherited from the supertable.

See PRIMARY KEY within the description of the *unique-constraint* below.

UNIQUE
This provides a shorthand method of defining a unique key composed of a single column. Thus, if UNIQUE is specified in the definition of column C, the effect is the same as if the UNIQUE(C) clause is specified as a separate clause.

A unique constraint cannot be specified if the table is a subtable (SQLSTATE 429B3) since unique constraints are inherited from the supertable.

See UNIQUE within the description of the *unique-constraint* below.

references-clause
This provides a shorthand method of defining a foreign key composed of a single column. Thus, if a references-clause is specified in the definition of column C, the effect is the same as if that references-clause were specified as part of a FOREIGN KEY clause in which C is the only identified column.

See *references-clause* under *referential-constraint* below.

CHECK (*check-condition***)**
This provides a shorthand method of defining a check constraint that applies to a single column. See CHECK (*check-condition*) below.

INLINE LENGTH *integer*
This option is only valid for a column defined using a structured type (SQLSTATE 42842) and indicates the maximum byte size of an instance

of a structured type to store inline with the rest of the values in the row. Instances of structured types that cannot be stored inline are stored separately from the base table row, similar to the way that LOB values are handled. This takes place automatically.

The default INLINE LENGTH for a structured-type column is the inline length of its type (specified explicitly or by default in the CREATE TYPE statement). If INLINE LENGTH of the structured type is less than 292, the value 292 is used for the INLINE LENGTH of the column.

Note: The inline lengths of subtypes are not counted in the default inline length, meaning that instances of subtypes may not fit inline unless an explicit INLINE LENGTH is specified at CREATE TABLE time to account for existing and future subtypes.

The explicit INLINE LENGTH value must be at least 292 and cannot exceed 32672 (SQLSTATE 54010).

COMPRESS SYSTEM DEFAULT
Specifies that system default values (that is, the default values used for the data types when no specific values are specified) are to be stored using minimal space. If the VALUE COMPRESSION clause is not specified, a warning is returned (SQLSTATE 01648) and system default values are not stored using minimal space.

Allowing system default values to be stored in this manner causes a slight performance penalty during insert and update operations on the column because of extra checking that is done.

The base data type must not be DATE, TIME, or TIMESTAMP (SQLSTATE 42842). If the base data type is a varying-length string, this clause is ignored. String values of length 0 are automatically compressed if a table has been set with VALUE COMPRESSION.

generated-column-spec

default-clause
Specifies a default value for the column.

WITH
An optional keyword.

DEFAULT
Provides a default value in the event a value is not supplied on INSERT or is specified as DEFAULT on INSERT or UPDATE. If a default value is not specified following the DEFAULT keyword, the default value depends on the data type of the column as shown in “ALTER TABLE”.

If a column is defined as a DATALINK, then a default value cannot be specified (SQLSTATE 42613). The only possible default is NULL.

If the column is based on a column of a typed table, a specific default value must be specified when defining a default. A default value cannot be specified for the object identifier column of a typed table (SQLSTATE 42997).

If a column is defined using a distinct type, then the default value of the column is the default value of the source data type cast to the distinct type.

If a column is defined using a structured type, the *default-clause* cannot be specified (SQLSTATE 42842).

Omission of DEFAULT from a *column-definition* results in the use of the null value as the default for the column. If such a column is defined NOT NULL, then the column does not have a valid default.

default-values
Specific types of default values that can be specified are as follows.

constant
Specifies the constant as the default value for the column. The specified constant must:

- represent a value that could be assigned to the column in accordance with the rules of assignment as described in Chapter 3
- not be a floating-point constant unless the column is defined with a floating-point data type
- not have non-zero digits beyond the scale of the column data type if the constant is a decimal constant (for example, 1.234 cannot be the default for a DECIMAL(5,2) column)
- be expressed with no more than 254 characters including the quote characters, any introducer character such as the X for a hexadecimal constant, and characters from the fully qualified function name and parentheses when the constant is the argument of a *cast-function*.

datetime-special-register
Specifies the value of the datetime special register (CURRENT DATE, CURRENT TIME, or CURRENT TIMESTAMP) at the time of INSERT, UPDATE, or LOAD as the default for the column. The data type of the column must be the data type that corresponds to the special register specified (for example, data type must be DATE when CURRENT DATE is specified).

user-special-register
Specifies the value of the user special register (CURRENT USER, SESSION_USER, SYSTEM_USER) at the time of INSERT, UPDATE, or LOAD as the default for the column. The data type of the column must be a character string with a length not less than the length attribute of a user special register. Note that USER can be specified in place of SESSION_USER and CURRENT_USER can be specified in place of CURRENT USER.

CURRENT SCHEMA
Specifies the value of the CURRENT SCHEMA special register at the time of INSERT, UPDATE, or LOAD as the default for the column. If CURRENT SCHEMA is specified, the data type of the column must be a character string with

a length greater than or equal to the length attribute of the CURRENT SCHEMA special register.

NULL
Specifies NULL as the default for the column. If NOT NULL was specified, DEFAULT NULL may be specified within the same column definition but will result in an error on any attempt to set the column to the default value.

cast-function
This form of a default value can only be used with columns defined as a distinct type, BLOB or datetime (DATE, TIME or TIMESTAMP) data type. For distinct type, with the exception of distinct types based on BLOB or datetime types, the name of the function must match the name of the distinct type for the column. If qualified with a schema name, it must be the same as the schema name for the distinct type. If not qualified, the schema name from function resolution must be the same as the schema name for the distinct type. For a distinct type based on a datetime type, where the default value is a constant, a function must be used and the name of the function must match the name of the source type of the distinct type with an implicit or explicit schema name of SYSIBM. For other datetime columns, the corresponding datetime function may also be used. For a BLOB or a distinct type based on BLOB, a function must be used and the name of the function must be BLOB with an implicit or explicit schema name of SYSIBM.

constant
Specifies a constant as the argument. The constant must conform to the rules of a constant for the source type of the distinct type or for the data type if not a distinct type. If the *cast-function* is BLOB, the constant must be a string constant.

datetime-special-register
Specifies CURRENT DATE, CURRENT TIME, or CURRENT TIMESTAMP. The source type of the distinct type of the column must be the data type that corresponds to the specified special register.

user-special-register
Specifies CURRENT USER, SESSION_USER, or SYSTEM_USER. The data type of the source type of the distinct type of the column must be a string data type with a length of at least 8 bytes. If the *cast-function* is BLOB, the length attribute must be at least 8 bytes.

CURRENT SCHEMA
Specifies the value of the CURRENT SCHEMA special register. The data type of the source type of the distinct type of the column must be a character string with a length greater than or equal to the length attribute of the CURRENT SCHEMA special register. If the cast-function is BLOB, the length attribute must be at least 8 bytes.

If the value specified is not valid, an error is returned (SQLSTATE 42894).

GENERATED
Indicates that DB2 generates values for the column. GENERATED must be specified if the column is to be considered an IDENTITY column.

ALWAYS
Specifies that DB2 will always generate a value for the column when a row is inserted into the table, or whenever the result value of the *generation-expression* changes. The result of the expression is stored in the table. GENERATED ALWAYS is the recommended value unless data propagation or unload and reload operations are being done. GENERATED ALWAYS is the required value for generated columns.

BY DEFAULT
Specifies that DB2 will generate a value for the column when a row is inserted, or updated specifying the DEFAULT clause, unless an explicit value is specified. BY DEFAULT is the recommended value when using data propagation or performing an unload and reload operation.

Although not explicitly required, a unique single-column index should be defined on the generated column to ensure uniqueness of the values.

AS IDENTITY
Specifies that the column is to be the identity column for this table. A table can only have a single IDENTITY column (SQLSTATE 428C1). The IDENTITY keyword can only be specified if the data type associated with the column is an exact numeric type with a scale of zero, or a user-defined distinct type for which the source type is an exact numeric type with a scale of zero (SQLSTATE 42815). SMALLINT, INTEGER, BIGINT, or DECIMAL with a scale of zero, or a distinct type based on one of these types, are considered exact numeric types. By contrast, single- and double-precision floating points are considered approximate numeric data types. Reference types, even if represented by an exact numeric type, cannot be defined as identity columns.

An identity column is implicitly NOT NULL. An identity column cannot have a DEFAULT clause (SQLSTATE 42623).

START WITH *numeric-constant*
Specifies the first value for the identity column. This value can be any positive or negative value that could be assigned to this column (SQLSTATE 42815), without non-zero digits existing to the right of the decimal point (SQLSTATE 428FA). The default is MINVALUE for ascending sequences, and MAXVALUE for descending sequences.

INCREMENT BY *numeric-constant*
Specifies the interval between consecutive values of the identity column. This value can be any positive or negative value that could be assigned to this column (SQLSTATE 42815), and does not exceed the value of a large integer constant (SQLSTATE 42820), without non-zero digits existing to the right of the decimal point (SQLSTATE 428FA).

If this value is negative, this is a descending sequence. If this value is 0, or positive, this is an ascending sequence. The default is 1.

NO MINVALUE or **MINVALUE**
Specifies the minimum value at which a descending identity column either cycles or stops generating values, or an ascending identity column cycles to after reaching the maximum value.

NO MINVALUE
For an ascending sequence, the value is the START WITH value, or 1 if START WITH was not specified. For a descending sequence, the value is the minimum value of the data type of the column. This is the default.

MINVALUE *numeric-constant*
Specifies the numeric constant that is the minimum value. This value can be any positive or negative value that could be assigned to this column (SQLSTATE 42815), without non-zero digits existing to the right of the decimal point (SQLSTATE 428FA), but the value must be less than or equal to the maximum value (SQLSTATE 42815).

NO MAXVALUE or **MAXVALUE**
Specifies the maximum value at which an ascending identity column either cycles or stops generating values, or a descending identity column cycles to after reaching the minimum value.

NO MAXVALUE
For an ascending sequence, the value is the maximum value of the data type of the column. For a descending sequence, the value is the START WITH value, or -1 if START WITH was not specified. This is the default.

MAXVALUE *numeric-constant*
Specifies the numeric constant that is the maximum value. This value can be any positive or negative value that could be assigned to this column (SQLSTATE 42815), without non-zero digits existing to the right of the decimal point (SQLSTATE 428FA), but the value must be greater than or equal to the minimum value (SQLSTATE 42815).

NO CYCLE or **CYCLE**
Specifies whether this identity column should continue to generate values after generating either its maximum or minimum value.

NO CYCLE
Specifies that values will not be generated for the identity column once the maximum or minimum value has been reached. This is the default.

CYCLE
Specifies that values continue to be generated for this column after the maximum or minimum value has been reached. If this option is used, after an ascending identity column reaches the maximum value, it generates its minimum value; or after a descending sequence reaches

the minimum value, it generates its maximum value. The maximum and minimum values for the identity column determine the range that is used for cycling.

When CYCLE is in effect, DB2 may generate duplicate values for an identity column. Although not explicitly required, a unique, single-column index should be defined on the generated column to ensure uniqueness of the values, if unique values are desired. If a unique index exists on such an identity column and a non-unique value is generated, an error occurs (SQLSTATE 23505).

NO CACHE or **CACHE**
Specifies whether to keep some pre-allocated values in memory for faster access. If a new value is needed for the identity column, and there are none available in the cache, then the end of the new cache block must be logged. However, when a new value is needed for the identity column, and there is an unused value in the cache, then the allocation of that identity value is faster, because no logging is necessary. This is a performance and tuning option.

NO CACHE
Specifies that values for the identity column are not to be pre-allocated.

When this option is specified, the values of the identity column are not stored in the cache. In this case, every request for a new identity value results in synchronous I/O to the log.

CACHE *integer-constant*
Specifies how many values of the identity sequence are to be pre-allocated and kept in memory. When values are generated for the identity column, pre-allocating and storing values in the cache reduces synchronous I/O to the log.

If a new value is needed for the identity column and there are no unused values available in the cache, the allocation of the value involves waiting for I/O to the log. However, when a new value is needed for the identity column and there is an unused value in the cache, the allocation of that identity value can happen more quickly by avoiding the I/O to the log.

In the event of a database deactivation, either normally or due to a system failure, all cached sequence values that have not been used in committed statements are *lost*; that is, they will never be used. The value specified for the CACHE option is the maximum number of values for the identity column that could be lost in case of database deactivation. (If a database is not explicitly activated, using the ACTIVATE command or API, when the last application is disconnected from the database, an implicit deactivation occurs.)

The minimum value is 2 (SQLSTATE 42815). The default value is CACHE 20.

NO ORDER or **ORDER**
Specifies whether the identity values must be generated in order of request.

NO ORDER
Specifies that the values do not need to be generated in order of request. This is the default.

ORDER
Specifies that the values must be generated in order of request.

GENERATED ALWAYS AS (*generation-expression***)**
Specifies that the definition of the column is based on an expression. (If the expression for a GENERATED ALWAYS column includes a user-defined external function, changing the executable for the function (such that the results change for given arguments) can result in inconsistent data. This can be avoided by using the SET INTEGRITY statement to force the generation of new values.) The *generation-expression* cannot contain any of the following (SQLSTATE 42621):

- Subqueries
- Column functions
- Dereference operations or DEREF functions
- User-defined or built-in functions that are non-deterministic
- User-defined functions using the EXTERNAL ACTION option
- User-defined functions defined with either CONTAINS SQL or READS SQL DATA
- Host variables or parameter markers
- Special registers
- References to columns defined later in the column list
- References to other generated columns

The data type for the column is based on the result data type of the *generation-expression*. A CAST specification can be used to force a particular data type and to provide a scope (for a reference type only). If *data-type* is specified, values are assigned to the column according to the appropriate assignment rules. A generated column is implicitly considered nullable, unless the NOT NULL column option is used. The data type of a generated column must be one for which equality is defined. This excludes columns of type LONG VARCHAR, LONG VARGRAPHIC, or DATALINK; LOB data types; structured types; and distinct types based on any of these types (SQLSTATE 42962).

OID-column-definition
Defines the object identifier column for the typed table.

REF IS *OID-column-name* **USER GENERATED**
Specifies that an object identifier (OID) column is defined in the table as the first column. An OID is required for the root table of a table hierarchy (SQLSTATE 428DX). The table must be a typed table (the OF clause must be present) that is not a subtable (SQLSTATE 42613). The name for the column is defined as *OID-column-name* and cannot be the same as the name of any attribute of the structured type *type-name1* (SQLSTATE 42711). The

column is defined with type REF(*type-name1*), NOT NULL and a system required unique index (with a default index name) is generated. This column is referred to as the *object identifier column* or *OID column*. The keywords USER GENERATED indicate that the initial value for the OID column must be provided by the user when inserting a row. Once a row is inserted, the OID column cannot be updated (SQLSTATE 42808).

with-options
Defines additional options that apply to columns of a typed table.

column-name
Specifies the name of the column for which additional options are specified. The *column-name* must correspond to the name of a column of the table that is not also a column of a supertable (SQLSTATE 428DJ). A column name can only appear in one WITH OPTIONS clause in the statement (SQLSTATE 42613).

If an option is already specified as part of the type definition (in CREATE TYPE), the options specified here override the options in CREATE TYPE.

WITH OPTIONS *column-options*
Defines options for the specified column. See *column-options* described earlier. If the table is a subtable, primary key or unique constraints cannot be specified (SQLSTATE 429B3).

DATA CAPTURE
Indicates whether extra information for inter-database data replication is to be written to the log. This clause cannot be specified when creating a subtable (SQLSTATE 42613).

If the table is a typed table, then this option is not supported (SQLSTATE 428DH or 42HDR).

NONE
Indicates that no extra information will be logged.

CHANGES
Indicates that extra information regarding SQL changes to this table will be written to the log. This option is required if this table will be replicated and the Capture program is used to capture changes for this table from the log.

If the table is defined to allow data on a partition other than the catalog partition (multiple partition database partition group or database partition group with a partition other than the catalog partition), then this option is not supported (SQLSTATE 42997).

If the schema name (implicit or explicit) of the table is longer than 18 bytes, then this option is not supported (SQLSTATE 42997).

WITH RESTRICT ON DROP
Indicates that the table cannot be dropped, and that the table space that contains the table cannot be dropped.

IN *tablespace-name1*
Identifies the table space in which the table will be created. The table space must exist, and be a REGULAR table space over which the authorization ID of the statement has USE privilege. If no other table space is specified, then all table parts will be stored in this table space. This clause cannot be specified when creating a subtable (SQLSTATE

42613), since the table space is inherited from the root table of the table hierarchy. If this clause is not specified, a table space for the table is determined as follows:

```
IF table space IBMDEFAULTGROUP over which the user
   has USE privilege exists with sufficient page size
     THEN choose it
 ELSE IF a table space over which the user has USE privilege
   exists with sufficient page size
   (see below when multiple table spaces qualify)
     THEN choose it
  ELSE issue an error (SQLSTATE 42727).
```

If more than one table space is identified by the ELSE IF condition, then choose the table space with the smallest sufficient page size over which the authorization ID of the statement has USE privilege. When more than one table space qualifies, preference is given according to who was granted the USE privilege:

1. the authorization ID
2. a group to which the authorization ID belongs
3. PUBLIC

If more than one table space still qualifies, the final choice is made by the database manager.

Determination of the table space may change when:

- table spaces are dropped or created
- USE privileges are granted or revoked.

The sufficient page size of a table is determined by either the byte count of the row or the number of columns. See "Row Size" on page 638 for more information.

tablespace-options
Specifies the table space in which indexes and/or long column values will be stored. For details on types of table spaces, see "CREATE TABLESPACE".

INDEX IN *tablespace-name2*
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, must be a REGULAR or LARGE DMS table space over which the authorization ID of the statement has USE privilege, and must be in the same database partition group as *tablespace-name1* (SQLSTATE 42838).

Note that specifying which table space will contain a table's index can only be done when the table is created. The checking of USE privilege over the table space for the index is only carried out at table creation time. The database manager will not require that the authorization ID of a CREATE INDEX statement have USE privilege on the table space when an index is created later.

LONG IN *tablespace-name3*
Identifies the table space in which the values of any long columns (LONG VARCHAR, LONG VARGRAPHIC, LOB data types, distinct types with any of these as source types, or any columns defined with user-defined structured types with values that cannot

be stored inline) will be stored. This option is allowed only when the primary table space specified in the IN clause is a DMS table space. The table space must exist, must be a LARGE DMS table space over which the authorization ID of the statement has USE privilege, and must be in the same database partition group as *tablespace-name1* (SQLSTATE 42838).

Note that specifying which table space will contain a table's long and LOB columns can only be done when the table is created. The checking of USE privilege over the table space for the long and LOB columns is only carried out at table creation time. The database manager will not require that the authorization ID of an ALTER TABLE statement have USE privilege on the table space when a long or LOB column is added later.

PARTITIONING KEY (*column-name***,...)**

Specifies the partitioning key used when data in the table is partitioned. Each *column-name* must identify a column of the table and the same column must not be identified more than once. No column with data type that is a LONG VARCHAR, LONG VARGRAPHIC, BLOB, CLOB, DBCLOB, DATALINK, distinct type based on any of these types, or structured type may be used as part of a partitioning key (SQLSTATE 42962). A partitioning key cannot be specified for a table that is a subtable (SQLSTATE 42613), since the partitioning key is inherited from the root table in the table hierarchy.

If this clause is not specified, and this table resides in a multiple partition database partition group, then the partitioning key is defined as follows:

- if the table is a typed table, the object identifier column
- if a primary key is specified, the first column of the primary key is the partitioning key
- otherwise, the first column whose data type is not a LOB, LONG VARCHAR, LONG VARGRAPHIC, DATALINK column, distinct type based on one of these types, or structured type column is the partitioning key.

If none of the columns satisfy the requirement of the default partitioning key, the table is created without one. Such tables are allowed only in table spaces defined on single-partition database partition groups.

For tables in table spaces defined on single-partition database partition groups, any collection of non-long type columns can be used to define the partitioning key. If you do not specify this parameter, no partitioning key is created.

For restrictions related to the partitioning key, see 634.

USING HASHING

Specifies the use of the hashing function as the partitioning method for data distribution. This is the only partitioning method supported.

REPLICATED

Specifies that the data stored in the table is physically replicated on each database partition of the database partition group of the table space in which the table is defined. This means that a copy of all the

data in the table exists on each of these database partitions. This option can only be specified for a materialized query table (SQLSTATE 42997).

VALUE COMPRESSION
Specifies that NULL and 0-length data values are to be stored more efficiently for most data types. This also determines the row format that is to be used. If the table is a typed table, this option is only supported on the root table of the typed table hierarchy (SQLSTATE 428DR).

The 0-length data values for columns whose data type is BLOB, CLOB, DBCLOB, LONG VARCHAR, or LONG VARGRAPHIC are stored using minimal space. Each NULL value is stored without using an additional byte. The row format that is used to support this determines the byte counts for each data type, and tends to cause data fragmentation during updates. The new row format (specified for a column through the COMPRESS SYSTEM DEFAULT option) also allows system default values for the column to be stored more efficiently.

NOT LOGGED INITIALLY
Any changes made to the table by an Insert, Delete, Update, Create Index, Drop Index, or Alter Table operation in the same unit of work in which the table is created are not logged. For other considerations when using this option, see the "Notes" section of this statement.

All catalog changes and storage related information are logged, as are all operations that are done on the table in subsequent units of work.

Note: If non-logged activity occurs against a table that has the NOT LOGGED INITIALLY attribute activated, and if a statement fails (causing a rollback), or a ROLLBACK TO SAVEPOINT is executed, the entire unit of work is rolled back (SQL1476N). Furthermore, the table for which the NOT LOGGED INITIALLY attribute was activated is marked inaccessible after the rollback has occurred, and can only be dropped. Therefore, the opportunity for errors within the unit of work in which the NOT LOGGED INITIALLY attribute is activated should be minimized.

CCSID
Specifies the encoding scheme for string data stored in the table. If the CCSID clause is not specified, the default is CCSID UNICODE for Unicode databases, and CCSID ASCII for all other databases.

ASCII
Specifies that string data is encoded in the database code page. If the database is a Unicode database, CCSID ASCII cannot be specified (SQLSTATE 56031).

UNICODE
Specifies that string data is encoded in Unicode. If the database is a Unicode database, character data is in UTF-8, and graphic data is in UCS-2. If the database is not a Unicode database, character data is in UTF-8.

If the database is not a Unicode database, tables can be created with CCSID UNICODE, but the following rules apply:

- The alternate collating sequence must be specified in the database configuration before creating the table (SQLSTATE 56031). CCSID UNICODE tables collate with the alternate collating sequence specified in the database configuration.
- Tables or table functions created with CCSID ASCII, and tables or table functions created with CCSID UNICODE, cannot both be used in a single SQL statement (SQLSTATE 53090). This applies to tables and table functions referenced directly in the statement, as well as to tables and table functions referenced indirectly (such as, for example, through referential integrity constraints, triggers, materialized query tables, and tables in the body of views).
- Tables created with CCSID UNICODE cannot be referenced in SQL functions or SQL methods (SQLSTATE 560C0).
- An SQL statement that references a table created with CCSID UNICODE cannot invoke an SQL function or SQL method (SQLSTATE 53090).
- Graphic types and user-defined types cannot be used in CCSID UNICODE tables (SQLSTATE 560C1).
- Tables cannot have both the CCSID UNICODE clause and the DATA CAPTURE CHANGES clause specified (SQLSTATE 42613).
- The Explain tables cannot be created with CCSID UNICODE (SQLSTATE 55002).
- Declared global temporary tables cannot be created with CCSID UNICODE (SQLSTATE 56031).
- CCSID UNICODE tables cannot be created in a CREATE SCHEMA statement (SQLSTATE 53090).
- The exception table for a load operation must have the same CCSID as the target table for the operation (SQLSTATE 428A5).
- The exception table for a SET INTEGRITY statement must have the same CCSID as the target table for the statement (SQLSTATE 53090).
- The target table for event monitor data must not be declared as CCSID UNICODE (SQLSTATE 55049).
- Statements that reference a CCSID UNICODE table can only be invoked from a DB2 Version 8.1 or later client (SQLSTATE 42997).
- SQL statements are always interpreted in the database code page. In particular, this means that every character in literals, hex literals, and delimited identifiers must have a representation in the database code page; otherwise, the character will be replaced with the substitution character.

Host variables in the application are always in the application code page, regardless of the CCSID of any tables in the SQL statements that are invoked. DB2 will perform code page conversions as necessary to convert data between the application code page and the section code page. The registry variable DB2CODEPAGE can be set at the client to change the application code page.

OPTIONS (ADD *table-option-name string-constant*, **...)**

Table options are used to identify the remote base table. The *table-option-name* is the name of the option. The *string-constant* specifies the setting for the table option. The *string-constant* must be enclosed in single quotation marks.

The remote server (the server name that was specified in the CREATE SERVER statement) must be specified in the OPTIONS clause. The OPTIONS clause can also be used to override the schema or the unqualified name of the remote base table that is being created.

It is recommended that a schema name be specified. If a remote schema name is not specified, the qualifier for the table name is used. If the table name has no qualifier, the authorization ID of the statement is used.

If an unqualified name for the remote base table is not specified, *table-name* is used.

unique-constraint
Defines a unique or primary key constraint. If the table has a partitioning key, then any unique or primary key must be a superset of the partitioning key. A unique or primary key constraint cannot be specified for a table that is a subtable (SQLSTATE 429B3). Primary or unique keys cannot be subsets of dimensions (SQLSTATE 429BE). If the table is a root table, the constraint applies to the table and all its subtables.

CONSTRAINT *constraint-name*
Names the primary key or unique constraint.

UNIQUE (*column-name***,...)**
Defines a unique key composed of the identified columns. The identified columns must be defined as NOT NULL. Each *column-name* must identify a column of the table and the same column must not be identified more than once.

The number of identified columns must not exceed 16, and the sum of their stored lengths must not exceed 1024 (refer to "Byte Counts" on page 638 for the stored lengths). Unique keys with variable keyparts can have a size greater than 255 if the registry variable DB2_INDEX_2BYTEVARLEN is set to ON. No LOB, LONG VARCHAR, LONG VARGRAPHIC, DATALINK, distinct type based on one of these types, or structured type may be used as part of a unique key, even if the length attribute of the column is small enough to fit within the 1024-byte limit (SQLSTATE 54008).

The set of columns in the unique key cannot be the same as the set of columns in the primary key or another unique key (SQLSTATE 01543). (If LANGLEVEL is SQL92E or MIA, an error is returned, SQLSTATE 42891.)

A unique constraint cannot be specified if the table is a subtable (SQLSTATE 429B3), because unique constraints are inherited from the supertable.

The description of the table as recorded in the catalog includes the unique key and its unique index. A unique index will automatically be created for the columns in the sequence specified with ascending order for each column. The name of the index will be the same as the *constraint-name* if this does not conflict with an existing index in the schema where the table is created. If the index name conflicts, the name will be SQL, followed by a character timestamp (*yymmddhhmmssxxx*), with SYSIBM as the schema name.

PRIMARY KEY (*column-name***,...)**
Defines a primary key composed of the identified columns. The clause must not be specified more than once, and the identified columns must be defined as NOT NULL. Each *column-name* must identify a column of the table, and the same column must not be identified more than once.

The number of identified columns must not exceed 16, and the sum of their stored lengths must not exceed 1024 (refer to "Byte Counts" on page 638 for the stored lengths). Primary keys with variable keyparts can have a size greater than 255 if the registry variable DB2_INDEX_2BYTEVARLEN

is set to ON. No LOB, LONG VARCHAR, LONG VARGRAPHIC, DATALINK, distinct type based on one of these types, or structured type can be used as part of a primary key, even if the length attribute of the column is small enough to fit within the 1024-byte limit (SQLSTATE 54008).

The set of columns in the primary key cannot be the same as the set of columns in a unique key (SQLSTATE 01543). (If LANGLEVEL is SQL92E or MIA, an error is returned, SQLSTATE 42891.)

Only one primary key can be defined on a table.

A primary key cannot be specified if the table is a subtable (SQLSTATE 429B3) since the primary key is inherited from the supertable.

The description of the table as recorded in the catalog includes the primary key and its primary index. A unique index will automatically be created for the columns in the sequence specified with ascending order for each column. The name of the index will be the same as the *constraint-name* if this does not conflict with an existing index in the schema where the table is created. If the index name conflicts, the name will be SQL, followed by a character timestamp (*yymmddhhmmssxxx*), with SYSIBM as the schema name.

If the table has a partitioning key, the columns of a *unique-constraint* must be a superset of the partitioning key columns; column order is unimportant.

referential-constraint
Defines a referential constraint.

CONSTRAINT *constraint-name*
Names the referential constraint.

FOREIGN KEY (*column-name***,...)**
Defines a referential constraint with the specified *constraint-name*.

Let T1 denote the object table of the statement. The foreign key of the referential constraint is composed of the identified columns. Each name in the list of column names must identify a column of T1 and the same column must not be identified more than once. The number of identified columns must not exceed 16 and the sum of their stored lengths must not exceed 1024 (refer to "Byte Counts" on page 638 for the stored lengths). Foreign keys can be defined on variable length columns whose length is greater than 255 bytes. No LOB, LONG VARCHAR, LONG VARGRAPHIC, DATALINK, distinct type based on one of these types, or structured type column may be used as part of a foreign key (SQLSTATE 42962). There must be the same number of foreign key columns as there are in the parent key and the data types of the corresponding columns must be compatible (SQLSTATE 42830). Two column descriptions are compatible if they have compatible data types (both columns are numeric, character strings, graphic, date/time, or have the same distinct type).

references-clause
Specifies the parent table or the parent nickname, and the parent key for the referential constraint.

REFERENCES *table-name* or *nickname*
The table or nickname specified in a REFERENCES clause must identify a base table or a nickname that is described in the catalog, but must not identify a catalog table.

A referential constraint is a duplicate if its foreign key, parent key, and parent table or parent nickname are the same as the foreign key, parent key, and parent table or parent nickname of a previously specified referential constraint. Duplicate referential constraints are ignored, and a warning is returned (SQLSTATE 01543).

In the following discussion, let T2 denote the identified parent table, and let T1 denote the table being created (or altered). (T1 and T2 may be the same table).

The specified foreign key must have the same number of columns as the parent key of T2 and the description of the *n*th column of the foreign key must be comparable to the description of the *n*th column of that parent key. Datetime columns are not considered to be comparable to string columns for the purposes of this rule.

(*column-name***,...)**
The parent key of a referential constraint is composed of the identified columns. Each *column-name* must be an unqualified name that identifies a column of T2. The same column must not be identified more than once.

The list of column names must match the set of columns (in any order) of the primary key or a unique constraint that exists on T2 (SQLSTATE 42890). If a column name list is not specified, then T2 must have a primary key (SQLSTATE 42888). Omission of the column name list is an implicit specification of the columns of that primary key in the sequence originally specified.

The referential constraint specified by a FOREIGN KEY clause defines a relationship in which T2 is the parent and T1 is the dependent.

rule-clause
Specifies what action to take on dependent tables.

ON DELETE
Specifies what action is to take place on the dependent tables when a row of the parent table is deleted. There are four possible actions:

- NO ACTION (default)
- RESTRICT
- CASCADE
- SET NULL

The delete rule applies when a row of T2 is the object of a DELETE or propagated delete operation and that row has dependents in T1. Let *p* denote such a row of T2.

- If RESTRICT or NO ACTION is specified, an error occurs and no rows are deleted.
- If CASCADE is specified, the delete operation is propagated to the dependents of *p* in T1.
- If SET NULL is specified, each nullable column of the foreign key of each dependent of *p* in T1 is set to null.

SET NULL must not be specified unless some column of the foreign key allows null values. Omission of the clause is an implicit specification of ON DELETE NO ACTION.

If T1 is delete-connected to T2 through multiple paths, defining two SET NULL rules with overlapping foreign key definitions is not allowed. For example: T1 (i1, i2, i3). Rule1 with foreign key (i1, i2) and Rule2 with foreign key (i2, i3) is not allowed.

The firing order of the rules is:

1. RESTRICT
2. SET NULL OR CASCADE
3. NO ACTION

If any row in T1 is affected by two different rules, an error occurs and no rows are deleted.

A referential constraint cannot be defined if it would cause a table to be delete-connected to itself by a cycle involving two or more tables, and where one of the delete rules is RESTRICT or SET NULL (SQLSTATE 42915).

A referential constraint that would cause a table to be delete-connected to either itself or another table by multiple paths can be defined, except in the following cases (SQLSTATE 42915):

- A table must not be both a dependent table in a CASCADE relationship (self-referencing, or referencing another table), and have a self-referencing relationship in which the delete rule is RESTRICT or SET NULL.
- A key overlaps another key when at least one column in one key is the same as a column in the other key. When a table is delete-connected to another table through multiple relationships with overlapping foreign keys, those relationships must have the same delete rule, and none of the delete rules can be SET NULL.
- When a table is delete-connected to another table through multiple relationships, and at least one of those relationships is specified with a delete rule of SET NULL, the foreign key definitions of these relationships must not contain any partitioning key or MDC key column.
- When two tables are delete-connected to the same table through CASCADE relationships, the two tables must not be delete-connected to each other if the delete rule of the last relationship in each delete-connected path is RESTRICT or SET NULL.

If any row in T1 is affected by different delete rules, the result would be the effect of all the actions specified by these rules. AFTER triggers and CHECK constraints on T1 will also see the effect of all the actions. An example of this is a row that is targeted to be set null through one delete-connected path to an ancestor table, and targeted to be deleted by a second delete-connected path to the same ancestor table. The result would be the deletion of the row. AFTER DELETE triggers on this descendant table would be activated, but AFTER UPDATE triggers would not.

In applying the above rules to referential constraints, in which either the parent table or the dependent table is a member of a typed table hierarchy, all the referential constraints that apply to any table in the respective hierarchies are taken into consideration.

ON UPDATE
Specifies what action is to take place on the dependent tables when a row of the parent table is updated. The clause is optional. ON UPDATE NO ACTION is the default and ON UPDATE RESTRICT is the only alternative.

The difference between NO ACTION and RESTRICT is described in the "Notes" section of this statement.

constraint-attributes
Defines attributes associated with referential integrity or check constraints.

ENFORCED or **NOT ENFORCED**
Specifies whether the constraint is enforced by the database manager during normal operations such as insert, update, or delete. The default is ENFORCED.

ENFORCED
The constraint is enforced by the database manager. ENFORCED cannot be specified for a functional dependency (SQLSTATE 42621). ENFORCED cannot be specified when a referential constraint refers to a nickname (SQLSTATE 428G7).

NOT ENFORCED
The constraint is not enforced by the database manager. This should only be specified if the table data is independently known to conform to the constraint.

ENABLE QUERY OPTIMIZATION or **DISABLE QUERY OPTIMIZATION**
Specifies whether the constraint or functional dependency can be used for query optimization under appropriate circumstances. The default is ENABLE QUERY OPTIMIZATION.

ENABLE QUERY OPTIMIZATION
The constraint is assumed to be true and can be used for query optimization.

DISABLE QUERY OPTIMIZATION
The constraint cannot be used for query optimization.

check-constraint
Defines a check constraint. A *check-constraint* is a *search-condition* that must evaluate to not false or a functional dependency that is defined between columns.

CONSTRAINT *constraint-name*
Names the check constraint.

CHECK (*check-condition***)**
Defines a check constraint. The *search-condition* must be true or unknown for every row of the table.

search-condition
The *search-condition* has the following restrictions:

- A column reference must be to a column of the table being created.
- The *search-condition* cannot contain a TYPE predicate.
- The *search-condition* cannot contain any of the following (SQLSTATE 42621):
  - Subqueries

- Dereference operations or DEREF functions where the scoped reference argument is other than the object identifier (OID) column
- CAST specifications with a SCOPE clause
- Column functions
- Functions that are not deterministic
- Functions defined to have an external action
- User-defined functions defined with either CONTAINS SQL or READS SQL DATA
- Host variables
- Parameter markers
- Special registers
- References to generated columns other than the identity column

functional-dependency
Defines a functional dependency between columns.

column-name **DETERMINED BY** *column-name* or **(***column-name***,...) DETERMINED BY (***column-name***,...)**
The parent set of columns contains the identified columns that immediately precede the DETERMINED BY clause. The child set of columns contains the identified columns that immediately follow the DETERMINED BY clause. All of the restrictions on the *search-condition* apply to parent set and child set columns, and only simple column references are allowed in the set of columns (SQLSTATE 42621). The same column must not be identified more than once in the functional dependency (SQLSTATE 42709). The data type of the column must not be a LOB data type, a distinct type based on a LOB data type, or a structured type (SQLSTATE 42962). No column in the child set of columns can be a nullable column (SQLSTATE 42621).

If a check constraint is specified as part of a *column-definition*, a column reference can only be made to the same column. Check constraints specified as part of a table definition can have column references identifying columns previously defined in the CREATE TABLE statement. Check constraints are not checked for inconsistencies, duplicate conditions, or equivalent conditions. Therefore, contradictory or redundant check constraints can be defined, resulting in possible errors at execution time.

The *search-condition* "IS NOT NULL" can be specified; however, it is recommended that nullability be enforced directly, using the NOT NULL attribute of a column. For example, `CHECK (salary + bonus > 30000)` is accepted if salary is set to NULL, because CHECK constraints must be either satisfied or unknown, and in this case, salary is unknown. However, `CHECK (salary IS NOT NULL)` would be considered false and a violation of the constraint if salary is set to NULL.

Check constraints with *search-condition* are enforced when rows in the table are inserted or updated. A check constraint defined on a table automatically applies to all subtables of that table.

A functional dependency is not enforced by the database manager during normal operations such as insert, update, delete, or set integrity. The

functional dependency might be used during query rewrite to optimize queries. Incorrect results might be returned if the integrity of a functional dependency is not maintained.

Rules:

- The sum of the byte counts of the columns, including the inline lengths of all structured type columns, must not be greater than the row size limit that is based on the page size of the table space (SQLSTATE 54010). For more information, see "Byte Counts" on page 638. For typed tables, the byte count is applied to the columns of the root table of the table hierarchy, and every additional column introduced by every subtable in the table hierarchy (additional subtable columns must be considered nullable for byte count purposes, even if defined as not nullable). There is also an additional 4 bytes of overhead to identify the subtable to which each row belongs.
- The number of columns in a table cannot exceed 1 012 (SQLSTATE 54011). For typed tables, the total number of attributes of the types of all of the subtables in the table hierarchy cannot exceed 1010.
- An object identifier column of a typed table cannot be updated (SQLSTATE 42808).
- Any unique or primary key constraint defined on the table must be a superset of the partitioning key (SQLSTATE 42997).
- The following table provides the supported combinations of DATALINK options in the *file-link-options* (SQLSTATE 42613). WRITE PERMISSION ADMIN can only combine with READ PERMISSION DB. (Other combinations in the RECOVERY and the ON UNLINK clause are supported.)

Table 59. Valid DATALINK File Control Option Combinations. Any combination that cannot be found in this table is not supported, and results in SQLSTATE 42613.

INTEGRITY	READ PERMISSION	WRITE PERMISSION	RECOVERY	ON UNLINK
ALL	FS	FS	NO	Not applicable
ALL	FS	BLOCKED	NO	RESTORE
ALL	FS	BLOCKED	YES	RESTORE
ALL	DB	BLOCKED	NO	RESTORE
ALL	DB	BLOCKED	NO	DELETE
ALL	DB	BLOCKED	YES	RESTORE
ALL	DB	BLOCKED	YES	DELETE
ALL	DB	ADMIN	NO	RESTORE
ALL	DB	ADMIN	NO	DELETE
ALL	DB	ADMIN	YES	RESTORE
ALL	DB	ADMIN	YES	DELETE

- The following rules only apply to partitioned databases.
  - Tables composed only of columns with types LOB, LONG VARCHAR, LONG VARGRAPHIC, DATALINK, distinct type based on one of these types, or structured type can only be created in table spaces defined on single-partition database partition groups.
  - The partitioning key definition of a table in a table space defined on a multiple partition database partition group cannot be altered.
  - The partitioning key column of a typed table must be the OID column.

- A range-clustered table cannot be specified in a database with multiple database partitions (SQLSTATE 42997).
- The following restrictions apply to range-clustered tables:
  - VALUE COMPRESSION cannot be activated.
  - A clustering index cannot be created.
  - Altering the table to add a column is not supported.
  - Altering the table to change the data type of a column is not supported.
  - Altering the table to change PCTFREE is not supported.
  - Altering the table to set APPEND ON is not suported.
  - DETAILED statistics are not available.
  - The load utility cannot be used to populate the table.

Notes:

- ***Compatibilities***
  - For compatibility with previous versions of DB2:
    - The CONSTRAINT keyword can be omitted from a *column-definition* defining a references-clause.
    - *constraint-name* can be specified following FOREIGN KEY (without the CONSTRAINT keyword).
    - SUMMARY can optionally be specified after CREATE.
    - DEFINITION ONLY can be specified in place of WITH NO DATA.
  - For compatibility with previous versions of DB2, and for consistency:
    - A comma can be used to separate multiple options in the *identity-options* clause.
  - For compatibility with DB2 UDB for OS/390 and z/OS:
    - The following syntax is accepted as the default behavior:
      - IN database-name.tablespace-name
      - IN DATABASE database-name
      - FOR MIXED DATA
      - FOR SBCS DATA
    - The following syntax is also supported:
      - NOMINVALUE, NOMAXVALUE, NOCYCLE, NOCACHE, and NOORDER
- Creating a table with a schema name that does not already exist will result in the implicit creation of that schema provided the authorization ID of the statement has IMPLICIT_SCHEMA authority. The schema owner is SYSIBM. The CREATEIN privilege on the schema is granted to PUBLIC.
- If a foreign key is specified:
  - All packages with a delete usage on the parent table are invalidated.
  - All packages with an update usage on at least one column in the parent key are invalidated.
- Creating a subtable causes invalidation of all packages that depend on any table in table hierarchy.
- VARCHAR and VARGRAPHIC columns that are greater than 4 000 and 2 000 respectively should not be used as input parameters in functions in SYSFUN schema. Errors will occur when the function is invoked with an argument value that exceeds these lengths (SQLSTATE 22001).

- The use of NO ACTION or RESTRICT as delete or update rules for referential constraints determines when the constraint is enforced. A delete or update rule of RESTRICT is enforced *before* all other constraints, including those referential constraints with modifying rules such as CASCADE or SET NULL. A delete or update rule of NO ACTION is enforced *after* other referential constraints. One example where different behavior is evident involves the deletion of rows from a view that is defined as a UNION ALL of related tables.

  ```
  Table T1 is a parent of table T3; delete rule as noted below.
  Table T2 is a parent of table T3; delete rule CASCADE.

  CREATE VIEW V1 AS SELECT * FROM T1 UNION ALL SELECT * FROM T2

  DELETE FROM V1
  ```

  If table T1 is a parent of table T3 with a delete rule of RESTRICT, a restrict violation will be raised (SQLSTATE 23001) if there are any child rows for parent keys of T1 in T3.

  If table T1 is a parent of table T3 with a delete rule of NO ACTION, the child rows may be deleted by the delete rule of CASCADE when deleting rows from T2 before the NO ACTION delete rule is enforced for the deletes from T1. If deletes from T2 did not result in deleting all child rows for parent keys of T1 in T3, then a constraint violation will be raised (SQLSTATE 23504).

  Note that the SQLSTATE returned is different depending on whether the delete or update rule is RESTRICT or NO ACTION.
- For tables in table spaces defined on multiple partition database partition groups, table collocation should be considered in choosing the partitioning keys. Following is a list of items to consider:
  - The tables must be in the same database partition group for collocation. The table spaces may be different, but must be defined in the same database partition group.
  - The partitioning keys of the tables must have the same number of columns, and the corresponding key columns must be partition compatible for collocation.
  - The choice of partitioning key also has an impact on performance of joins. If a table is frequently joined with another table, you should consider the joining column(s) as a partitioning key for both tables.
- The NOT LOGGED INITIALLY clause cannot be used when DATALINK columns with the FILE LINK CONTROL attribute are present in the table (SQLSTATE 42613).
- The NOT LOGGED INITIALLY option is useful for situations where a large result set needs to be created with data from an alternate source (another table or a file) and recovery of the table is not necessary. Using this option will save the overhead of logging the data. The following considerations apply when this option is specified:
  - When the unit of work is committed, all changes that were made to the table during the unit of work are flushed to disk.
  - When you run the rollforward utility and it encounters a log record that indicates that a table in the database was either populated by the Load utility or created with the NOT LOGGED INITIALLY option, the table will be marked as unavailable. The table will be dropped by the rollforward utility if it later encounters a DROP TABLE log. Otherwise, after the database is recovered, an error will be issued if any attempt is made to access the table (SQLSTATE 55019). The only operation permitted is to drop the table.
  - Once such a table is backed up as part of a database or table space back up, recovery of the table becomes possible.

- A REFRESH DEFERRED system-maintained materialized query table defined with ENABLE QUERY OPTIMIZATION can be used to optimize the processing of queries if CURRENT REFRESH AGE is set to ANY and CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION is set such that it includes system-maintained materialized query tables. A REFRESH DEFERRED user-maintained materialized query table defined with ENABLE QUERY OPTIMIZATION can be used to optimize the processing of queries if CURRENT REFRESH AGE is set to ANY and CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION is set such that it includes user-maintained materialized query tables. A REFRESH IMMEDIATE materialized query table defined with ENABLE QUERY OPTIMIZATION is always considered for optimization. For this optimization to be able to use a REFRESH DEFERRED or a REFRESH IMMEDIATE materialized query table, the fullselect must conform to certain rules in addition to those already described. The fullselect must:
  - be a subselect with a GROUP BY clause or a subselect with a single table reference
  - not include DISTINCT anywhere in the select list
  - not include any special registers
  - not include functions that are not deterministic.

  If the query specified when creating a materialized query table does not conform to these rules, a warning is returned (SQLSTATE 01633).
- If a materialized query table is defined with REFRESH IMMEDIATE, or a staging table is defined with PROPAGATE IMMEDIATE, it is possible for an error to occur when attempting to apply the change resulting from an insert, update, or delete operation on an underlying table. The error will cause the failure of the insert, update, or delete operation on the underlying table.
- Materialized query tables or staging tables cannot be used as exception tables when constraints are checked in bulk, such as during load operations or during execution of the SET INTEGRITY statement.
- Certain operations cannot be performed on a table that is referenced by a materialized query table defined with REFRESH IMMEDIATE, or defined with REFRESH DEFERRED with an associated staging table:
  - IMPORT REPLACE cannot be used.
  - ALTER TABLE NOT LOGGED INITIALLY WITH EMPTY TABLE cannot be done.
- In a federated system, nicknames for relational data sources or local tables can be used as the underlying tables to create a materialized query table. Nicknames for non-relational data sources are not supported. When a nickname is one of the underlying tables, the REFRESH DEFERRED option must be used. System-maintained materialized query tables that reference nicknames are not supported in a partitioned database environment.
- ***Transparent DDL:*** In a federated system, a remote base table can be created, altered, or dropped using DB2 UDB SQL. This capability is known as *transparent DDL*. Before a remote base table can be created on a data source, the federated server must be configured to access that data source. This configuration includes creating the wrapper for the data source, supplying the server definition for the server where the remote base table will be located, and creating the user mappings between the federated server and the data source.

  Transparent DDL does impose some limitations on what can be included in the CREATE TABLE statement:
  - Only columns and a primary key can be created on the remote base table.
  - The remote data source must support:

- The remote column data types to which the DB2 column data types are mapped
- The primary key option in the CREATE TABLE statement

Depending on how the data source responds to requests it does not support, an error might be returned or the request might be ignored.

When a remote base table is created using transparent DDL, a nickname is automatically created for that remote base table.

- A referential constraint may be defined in such a way that either the parent table or the dependent table is a part of a table hierarchy. In such a case, the effect of the referential constraint is as follows:
  1. Effects of INSERT, UPDATE, and DELETE statements:
     - If a referential constraint exists, in which PT is a parent table and DT is a dependent table, the constraint ensures that for each row of DT (or any of its subtables) that has a non-null foreign key, a row exists in PT (or one of its subtables) with a matching parent key. This rule is enforced against any action that affects a row of PT or DT, regardless of how that action is initiated.
  2. Effects of DROP TABLE statements:
     - for referential constraints in which the dropped table is the parent table or dependent table, the constraint is dropped
     - for referential constraints in which a supertable of the dropped table is the parent table the rows of the dropped table are considered to be deleted from the supertable. The referential constraint is checked and its delete rule is invoked for each of the deleted rows.
     - for referential constraints in which a supertable of the dropped table is the dependent table, the constraint is not checked. Deletion of a row from a dependent table cannot result in violation of a referential constraint.
- ***Privileges:*** When any table is created, the definer of the table is granted CONTROL privilege. When a subtable is created, the SELECT privilege that each user or group has on the immediate supertable is automatically granted on the subtable with the table definer as the grantor.
- ***Row Size:*** The maximum number of bytes allowed in the row of a table is dependent on the page size of the table space in which the table is created (*tablspace-name1*). The following list shows the row size limit and number of columns limit associated with each table space page size.

Table 60. Limits for Number of Columns and Row Size in Each Table Space Page Size

Page Size	Row Size Limit	Column Count Limit
4K	4 005	500
8K	8 101	1 012
16K	16 293	1 012
32K	32 677	1 012

The actual number of columns for a table may be further limited by the following formula:

```
Total Columns * 8 + Number of LOB Columns * 12 +
  Number of Datalink Columns * 28 <= row size limit for page size.
```

- ***Byte Counts:*** The following table contains the byte counts of columns by data type for columns that do not allow null values. In tables without value compression, each column that allows nulls requires an additional byte.

If a table is based on a structured type, an additional 4 bytes of overhead is reserved to identify rows of subtables, regardless of whether or not subtables are defined. Additional subtable columns must be considered nullable for byte count purposes, even if defined as not nullable.

When calculating stored lengths, to ensure that the 1024-byte limit is not exceeded in an index or in constraints (note that constraints are enforced through indexes), the overhead is 2 bytes instead of 4 bytes.

Table 61. Byte Counts of Columns by Data Type

Data type	Byte count when VALUE COMPRESSION is activated for the table	Byte count when VALUE COMPRESSION is not activated, either implicitly or explicitly, for the table; if the column is nullable, add 1 to the indicated byte count
ROW OVERHEAD	2	0
INTEGER	6	4
SMALLINT	4	2
BIGINT	10	8
REAL	6	4
DOUBLE	10	8
DECIMAL	The integral part of (p/2)+3, where p is the precision	The integral part of (p/2)+1, where p is the precision
CHAR(n)	n+2	n
VARCHAR(n)	n+2	n+4 (within a table); n+2 (within an index)
LONG VARCHAR	22	24
GRAPHIC(n)	n*2+2	n*2
VARGRAPHIC(n)	(n*2)+2	(n*2)+4 (within a table); (n*2)+2 (within an index)
LONG VARGRAPHIC	22	24
DATE	6	4
TIME	5	3
TIMESTAMP	12	10
DATALINK(n)	n+52	n+54
Maximum LOB length 1024	70[1]	72
Maximum LOB length 8192	94	96
Maximum LOB length 65 536	118	120
Maximum LOB length 524 000	142	144
Maximum LOB length 4 190 000	166	168
Maximum LOB length 134 000 000	198	200
Maximum LOB length 536 000 000	222	224
Maximum LOB length 1 070 000 000	254	256

Table 61. Byte Counts of Columns by Data Type (continued)

Data type	Byte count when VALUE COMPRESSION is activated for the table	Byte count when VALUE COMPRESSION is not activated, either implicitly or explicitly, for the table; if the column is nullable, add 1 to the indicated byte count
Maximum LOB length 1 470 000 000	278	280
Maximum LOB length 2 147 483 647	314	316

[1] Each LOB value has a *LOB descriptor* in the base record that points to the location of the actual value. The size of the descriptor varies according to the maximum length defined for the column.

For a *distinct type*, the byte count is equivalent to the length of the source type of the distinct type. For a *reference type*, the byte count is equivalent to the length of the built-in data type on which the reference type is based. For a *structured type*, the byte count is equivalent to the INLINE LENGTH + 4. The INLINE LENGTH is the value specified (or implicitly calculated) for the column in the *column-options* clause.

- ***Dimension Columns:*** Because each distinct value of a dimension column is assigned to a different block of the table, clustering on an expression may be desirable, such as "INTEGER(ORDER_DATE)/100". In this case, a generated column can be defined for the table, and this generated column may then be used in the ORGANIZE BY DIMENSIONS clause. If the expression is monotonic with respect to a column of the table, DB2 may use the dimension index to satisfy range predicates on that column. For example, if the expression is simply *column-name + some-positive-constant*, it is monotonic increasing. User-defined functions, certain built-in functions, and using more than one column in an expression, prevent monotonicity or its detection.

  Dimensions involving generated columns whose expressions are non-monotonic, or whose monotonicity cannot be determined, can still be created, but range queries along slice or cell boundaries of these dimensions are not supported. Equality and IN predicates *can* be processed by slices or cells.

  A generated column is monotonic if the following is true with respect to the generating function, fn:

  – Monotonic increasing.

    For every possible pair of values x1 and x2, if x2>x1, then fn(x2)>fn(x1). For example:

    ```
    SALARY - 10000
    ```

  – Monotonic decreasing.

    For every possible pair of values x1 and x2, if x2>x1, then fn(x2)<fn(x1). For example:

    ```
    -SALARY
    ```

  – Monotonic non-decreasing.

    For every possible pair of values x1 and x2, if x2>x1, then fn(x2)>=fn(x1). For example:

    ```
    SALARY/1000
    ```

  – Monotonic non-increasing.

    For every possible pair of values x1 and x2, if x2>x1, then fn(x2)<=fn(x1). For example:

    ```
    -SALARY/1000
    ```

The expression ″PRICE*DISCOUNT″ is not monotonic, because it involves more than one column of the table.

- ***Range-clustered tables:*** Organizing a table by key sequence is effective for certain types of tables. The table should have an integer key that is tightly clustered (dense) over the range of possible values. The columns of this integer key must not be nullable, and the key should logically be the primary key of the table. The organization of a range-clustered table precludes the need for a separate unique index object, providing direct access to the row for a specified key value, or a range of rows for a specified range of key values. The allocation of all the space for the complete set of rows in the defined key sequence range is done during table creation, and must be considered when defining a range-clustered table. The storage space is not available for any other use, even though the rows are initially marked deleted. If the full key sequence range will be populated with data only over a long period of time, this table organization may not be an appropriate choice.

Examples:

Example 1: Create table TDEPT in the DEPARTX table space. DEPTNO, DEPTNAME, MGRNO, and ADMRDEPT are column names. CHAR means the column will contain character data. NOT NULL means that the column cannot contain a null value. VARCHAR means the column will contain varying-length character data. The primary key consists of the column DEPTNO.

```
CREATE TABLE TDEPT
  (DEPTNO   CHAR(3)     NOT NULL,
   DEPTNAME VARCHAR(36) NOT NULL,
   MGRNO    CHAR(6),
   ADMRDEPT CHAR(3)     NOT NULL,
   PRIMARY KEY(DEPTNO))
IN DEPARTX
```

Example 2: Create table PROJ in the SCHED table space. PROJNO, PROJNAME, DEPTNO, RESPEMP, PRSTAFF, PRSTDATE, PRENDATE, and MAJPROJ are column names. CHAR means the column will contain character data. DECIMAL means the column will contain packed decimal data. 5,2 means the following: 5 indicates the number of decimal digits, and 2 indicates the number of digits to the right of the decimal point. NOT NULL means that the column cannot contain a null value. VARCHAR means the column will contain varying-length character data. DATE means the column will contain date information in a three-part format (year, month, and day).

```
CREATE TABLE PROJ
  (PROJNO   CHAR(6)      NOT NULL,
   PROJNAME VARCHAR(24)  NOT NULL,
   DEPTNO   CHAR(3)      NOT NULL,
   RESPEMP  CHAR(6)      NOT NULL,
   PRSTAFF  DECIMAL(5,2)         ,
   PRSTDATE DATE                 ,
   PRENDATE DATE                 ,
   MAJPROJ  CHAR(6)      NOT NULL)
IN SCHED
```

Example 3: Create a table called EMPLOYEE_SALARY where any unknown salary is considered 0. No table space is specified, so that the table will be created in a table space selected by the system based on the rules described for the *IN tablespace-name1* clause.

```
CREATE TABLE EMPLOYEE_SALARY
  (DEPTNO   CHAR(3)      NOT NULL,
   DEPTNAME VARCHAR(36)  NOT NULL,
   EMPNO    CHAR(6)      NOT NULL,
   SALARY   DECIMAL(9,2) NOT NULL WITH DEFAULT)
```

Example 4: Create distinct types for total salary and miles and use them for columns of a table created in the default table space. In a dynamic SQL statement assume the CURRENT SCHEMA special register is JOHNDOE and the CURRENT PATH is the default ("SYSIBM","SYSFUN","JOHNDOE").

If a value for SALARY is not specified it must be set to 0 and if a value for LIVING_DIST is not specified it must to set to 1 mile.

```
CREATE DISTINCT TYPE JOHNDOE.T_SALARY AS INTEGER WITH COMPARISONS

CREATE DISTINCT TYPE JOHNDOE.MILES  AS FLOAT WITH COMPARISONS

CREATE TABLE EMPLOYEE
  (ID          INTEGER NOT NULL,
   NAME        CHAR (30),
   SALARY      T_SALARY NOT NULL WITH DEFAULT,
   LIVING_DIST MILES    DEFAULT MILES(1) )
```

Example 5: Create distinct types for image and audio and use them for columns of a table. No table space is specified, so that the table will be created in a table space selected by the system based on the rules described for the IN *tablespace-name1* clause. Assume the CURRENT PATH is the default.

```
CREATE DISTINCT TYPE IMAGE AS BLOB (10M)

CREATE DISTINCT TYPE AUDIO AS BLOB (1G)

CREATE TABLE PERSON
  (SSN    INTEGER NOT NULL,
   NAME   CHAR (30),
   VOICE  AUDIO,
   PHOTO  IMAGE)
```

Example 6: Create table EMPLOYEE in the HUMRES table space. The constraints defined on the table are the following:

- The values of department number must lie in the range 10 to 100.
- The job of an employee can only be either 'Sales', 'Mgr' or 'Clerk'.
- Every employee that has been with the company since 1986 must make more than $40,500.

Note: If the columns included in the check constraints are nullable they could also be NULL.

```
CREATE TABLE EMPLOYEE
  (ID          SMALLINT NOT NULL,
   NAME        VARCHAR(9),
   DEPT        SMALLINT CHECK (DEPT BETWEEN 10 AND 100),
   JOB         CHAR(5) CHECK (JOB IN ('Sales','Mgr','Clerk')),
   HIREDATE    DATE,
   SALARY      DECIMAL(7,2),
   COMM        DECIMAL(7,2),
   PRIMARY KEY (ID),
   CONSTRAINT  YEARSAL CHECK (YEAR(HIREDATE) > 1986
     OR SALARY > 40500)
  )
  IN HUMRES
```

Example 7: Create a table that is wholly contained in the PAYROLL table space.

```
CREATE TABLE EMPLOYEE .....
  IN PAYROLL
```

Example 8: Create a table with its data part in ACCOUNTING and its index part in ACCOUNT_IDX.

```
CREATE TABLE SALARY.....
  IN ACCOUNTING INDEX IN ACCOUNT_IDX
```

Example 9: Create a table and log SQL changes in the default format.

```
CREATE TABLE SALARY1 .....
```

or

```
CREATE TABLE SALARY1 .....
  DATA CAPTURE NONE
```

Example 10: Create a table and log SQL changes in an expanded format.

```
CREATE TABLE SALARY2 .....
  DATA CAPTURE CHANGES
```

Example 11: Create a table EMP_ACT in the SCHED table space. EMPNO, PROJNO, ACTNO, EMPTIME, EMSTDATE, and EMENDATE are column names. Constraints defined on the table are:

- The value for the set of columns, EMPNO, PROJNO, and ACTNO, in any row must be unique.
- The value of PROJNO must match an existing value for the PROJNO column in the PROJECT table and if the project is deleted all rows referring to the project in EMP_ACT should also be deleted.

```
CREATE TABLE EMP_ACT
  (EMPNO       CHAR(6) NOT NULL,
   PROJNO      CHAR(6) NOT NULL,
   ACTNO       SMALLINT NOT NULL,
   EMPTIME     DECIMAL(5,2),
   EMSTDATE    DATE,
   EMENDATE    DATE,
   CONSTRAINT EMP_ACT_UNIQ UNIQUE (EMPNO,PROJNO,ACTNO),
   CONSTRAINT FK_ACT_PROJ FOREIGN KEY (PROJNO)
                          REFERENCES PROJECT (PROJNO) ON DELETE CASCADE
  )
  IN SCHED
```

A unique index called EMP_ACT_UNIQ is automatically created in the same schema to enforce the unique constraint.

Example 12: Create a table that is to hold information about famous goals for the ice hockey hall of fame. The table will list information about the player who scored the goal, the goaltender against who it was scored, the date and place, and a description. When available, it will also point to places where newspaper articles about the game are stored and where still and moving pictures of the goal are stored. The newspaper articles are to be linked so they cannot be deleted or renamed but all existing display and update applications must continue to operate. The still pictures and movies are to be linked with access under complete control of DB2. The still pictures are to have recovery and are to be returned to their original owner if unlinked. The movie pictures are not to have recovery and are to be deleted if unlinked. The description column and the three DATALINK columns are nullable.

```
CREATE TABLE HOCKEY_GOALS
  ( BY_PLAYER      VARCHAR(30)   NOT NULL,
    BY_TEAM        VARCHAR(30)   NOT NULL,
    AGAINST_PLAYER VARCHAR(30)   NOT NULL,
    AGAINST_TEAM   VARCHAR(30)   NOT NULL,
    DATE_OF_GOAL   DATE          NOT NULL,
    DESCRIPTION    CLOB(5000),
    ARTICLES       DATALINK  LINKTYPE URL FILE LINK CONTROL MODE DB2OPTIONS,
    SNAPSHOT       DATALINK  LINKTYPE URL FILE LINK CONTROL
                             INTEGRITY ALL
                             READ PERMISSION DB WRITE PERMISSION BLOCKED
                             RECOVERY YES ON UNLINK RESTORE,
    MOVIE          DATALINK  LINKTYPE URL FILE LINK CONTROL
                             INTEGRITY ALL
                             READ PERMISSION DB WRITE PERMISSION BLOCKED
                             RECOVERY NO ON UNLINK DELETE )
```

Example 13: Suppose an exception table is needed for the EMPLOYEE table. One can be created using the following statement.

```
CREATE TABLE EXCEPTION_EMPLOYEE AS
  (SELECT EMPLOYEE.*,
    CURRENT TIMESTAMP AS TIMESTAMP,
    CAST ('' AS CLOB(32K)) AS MSG
  FROM EMPLOYEE
  ) WITH NO DATA
```

Example 14: Given the following table spaces with the indicated attributes:

```
TBSPACE            PAGESIZE    USER   USERAUTH
------------------ ----------- ------ --------
DEPT4K                    4096 BOBBY  Y
PUBLIC4K                  4096 PUBLIC Y
DEPT8K                    8192 BOBBY  Y
DEPT8K                    8192 RICK   Y
PUBLIC8K                  8192 PUBLIC Y
```

- If RICK creates the following table, it is placed in table space PUBLIC4K since the byte count is less than 4005; but if BOBBY creates the same table, it is placed in table space DEPT4K, since BOBBY has USE privilege because of an explicit grant:

  ```
  CREATE TABLE DOCUMENTS
    (SUMMARY    VARCHAR(1000),
     REPORT     VARCHAR(2000))
  ```

- If BOBBY creates the following table, it is placed in table space DEPT8K since the byte count is greater than 4005, and BOBBY has USE privilege because of an explicit grant. However, if DUNCAN creates the same table, it is placed in table space PUBLIC8K, since DUNCAN has no specific privileges:

  ```
  CREATE TABLE CURRICULUM
    (SUMMARY    VARCHAR(1000),
     REPORT     VARCHAR(2000),
     EXERCISES  VARCHAR(1500))
  ```

Example 15: Create a table with a LEAD column defined with the structured type EMP. Specify an INLINE LENGTH of 300 bytes for the LEAD column, indicating that any instances of LEAD that cannot fit within the 300 bytes are stored outside the table (separately from the base table row, similar to the way LOB values are handled).

```
CREATE TABLE PROJECTS (PID INTEGER,
  LEAD EMP INLINE LENGTH 300,
  STARTDATE DATE,
            ...)
```

Example 16: Create a table DEPT with five columns named DEPTNO, DEPTNAME, MGRNO, ADMRDEPT, and LOCATION. Column DEPT is to be defined as an IDENTITY column such that DB2 will always generate a value for it. The values for the DEPT column should begin with 500 and increment by 1.

```
CREATE TABLE DEPT
  (DEPTNO     SMALLINT      NOT NULL
                GENERATED ALWAYS AS IDENTITY
                  (START WITH 500, INCREMENT BY 1),
   DEPTNAME   VARCHAR(36)   NOT NULL,
   MGRNO      CHAR(6),
   ADMRDEPT   SMALLINT      NOT NULL,
   LOCATION   CHAR(30))
```

Example 17: Create a SALES table that is partitioned on the YEAR column, and that has dimensions on the REGION and YEAR columns. Data will be distributed across partitions according to hashed values of the YEAR column. On each partition, data will be organized into extents based on unique combinations of values of the REGION and YEAR columns on those partitions.

```
CREATE TABLE SALES
  (CUSTOMER   VARCHAR(80),
   REGION     CHAR(5),
   YEAR       INTEGER)
PARTITIONING KEY (YEAR)
ORGANIZE BY DIMENSIONS (REGION, YEAR)
```

Example 18: Create a SALES table with a PURCHASEYEARMONTH column that is generated from the PURCHASEDATE column. Use an expression to create a column that is monotonic with respect to the original PURCHASEDATE column, and is therefore suitable for use as a dimension. The table is partitioned on the REGION column, and organized within each partition into extents according to the PURCHASEYEARMONTH column; that is, different regions will be on different partitions, and different purchase months will belong to different cells (or sets of extents) within those partitions.

```
CREATE TABLE SALES
  (CUSTOMER            VARCHAR(80),
   REGION              CHAR(5),
   PURCHASEDATE        DATE,
   PURCHASEYEARMONTH   INTEGER
                         GENERATED ALWAYS AS (INTEGER(PURCHASEDATE)/100))
PARTITIONING KEY (REGION)
ORGANIZE BY DIMENSIONS (PURCHASEYEARMONTH)
```

Example 19: Create a CUSTOMER table with a CUSTOMERNUMDIM column that is generated from the CUSTOMERNUM column. Use an expression to create a column that is monotonic with respect to the original CUSTOMERNUM column, and is therefore suitable for use as a dimension. The table is organized into cells according to the CUSTOMERNUMDIM column, so that there is a different cell in the table for every 50 customers. If a unique index were created on CUSTOMERNUM, customer numbers would be clustered in such a way that each set of 50 values would be found in a particular set of extents in the table.

```
CREATE TABLE CUSTOMER
  (CUSTOMERNUM      INTEGER,
   CUSTOMERNAME     VARCHAR(80),
   ADDRESS          VARCHAR(200),
   CITY             VARCHAR(50),
   COUNTRY          VARCHAR(50),
   CODE             VARCHAR(15),
   CUSTOMERNUMDIM   INTEGER
                      GENERATED ALWAYS AS (CUSTOMERNUM/50))
ORGANIZE BY DIMENSIONS (CUSTOMERNUMDIM)
```

Example 20: Create a remote base table called EMPLOYEE on the Oracle server, ORASERVER. A nickname, named EMPLOYEE, which refers to this newly created remote base table, will also automatically be created.

```
CREATE TABLE EMPLOYEE
  (EMP_NO       CHAR(6)       NOT NULL,
   FIRST_NAME   VARCHAR(12)   NOT NULL,
   MID_INT      CHAR(1)       NOT NULL,
   LAST_NAME    VARCHAR(15)   NOT NULL,
   HIRE_DATE    DATE,
   JOB          CHAR(8),
   SALARY       DECIMAL(9,2),
   PRIMARY KEY (EMP_NO))
OPTIONS
  (REMOTE_SERVER 'ORASERVER',
   REMOTE_SCHEMA 'J15USER1',
   REMOTE_TABNAME 'EMPLOYEE')
```

The following CREATE TABLE statements show how to specify the table name, or the table name and the explicit remote base table name, to get the desired case. The lowercase identifier, employee, is used to illustrate the implicit folding of identifiers.

Create a remote base table called EMPLOYEE (uppercase characters) on an Informix server, and create a nickname named EMPLOYEE (uppercase characters) on that table:

```
CREATE TABLE employee
  (EMP_NO CHAR(6) NOT NULL,
   ...)
OPTIONS
  (REMOTE_SERVER 'INFX_SERVER')
```

If the REMOTE_TABNAME option is not specified, and *table-name* is not delimited, the remote base table name will be in uppercase characters, even if the remote data source normally stores names in lowercase characters.

Create a remote base table called employee (lowercase characters) on an Informix server, and create a nickname named EMPLOYEE (uppercase characters) on that table:

```
CREATE TABLE employee
  (EMP_NO CHAR(6) NOT NULL,
   ...)
OPTIONS
  (REMOTE_SERVER 'INFX_SERVER',
   REMOTE_TABNAME 'employee')
```

When creating a table at a remote data source that supports delimited identifiers, use the REMOTE_TABNAME option and a character string constant that specifies the table name in the desired case.

Create a remote base table called employee (lowercase characters) on an Informix server, and create a nickname named employee (lowercase characters) on that table:

```
CREATE TABLE "employee"
  (EMP_NO CHAR(6) NOT NULL,
   ...)
OPTIONS
  (REMOTE_SERVER 'INFX_SERVER')
```

If the REMOTE_TABNAME option is not specified, and *table-name* is delimited, the remote base table name will be identical to *table-name*.

Example 21: Create a range-clustered table that can be used to locate a student using a student ID. For each student record, include the school ID, program ID, student number, student ID, student first name, student last name, and student grade point average (GPA).

```
CREATE TABLE STUDENTS
  (SCHOOL_ID     INTEGER   NOT NULL,
   PROGRAM_ID    INTEGER   NOT NULL,
   STUDENT_NUM   INTEGER   NOT NULL,
   STUDENT_ID    INTEGER   NOT NULL,
   FIRST_NAME    CHAR(30),
   LAST_NAME     CHAR(30),
   GPA           DOUBLE)
ORGANIZE BY KEY SEQUENCE
  (STUDENT_ID
    STARTING FROM 1
    ENDING AT 1000000)
  DISALLOW OVERFLOW
```

The size of each record is the sum of the columns, plus alignment, plus the range-clustered table row header. In this case, the row size is 98 bytes: 4 + 4 + 4 + 4 + 30 + 30 + 8 + 3 (for nullable columns) + 1 (for alignment) + 10 (for the header). With a 4-KB page size (or 4096 bytes), after accounting for page overhead, there are 4038 bytes available, enough room for 41 records per page. Allowing for 1 million student records, there is a need for (1 million divided by 41 records per page) 24 391 pages. With two additional pages for table overhead, the final number of 4-KB pages that are allocated when the table is created is 24 393.

Example 22: Create a table named DEPARTMENT with a functional dependency that has no specified constraint name.

```
CREATE TABLE DEPARTMENT
  (DEPTNO     SMALLINT      NOT NULL,
   DEPTNAME   VARCHAR(36)   NOT NULL,
   MGRNO      CHAR(6),
   ADMRDEPT   SMALLINT      NOT NULL,
   LOCATION   CHAR(30),
  CHECK (DEPTNAME DETERMINED BY DEPTNO) NOT ENFORCED)
```

Related concepts:

- "Multidimensional clustering tables" in the *Administration Guide: Planning*
- "What is transparent DDL?" in the *Federated Systems Guide*

Related tasks:

- "Creating new remote tables using transparent DDL" in the *Federated Systems Guide*

Related reference:

- "Subselect" on page 904
- "ALTER TABLE" on page 525
- "CREATE TABLESPACE" on page 648
- "DECLARE GLOBAL TEMPORARY TABLE statement" in the *SQL Reference, Volume 2*
- "Assignments and comparisons" in the *SQL Reference, Volume 1*
- "Partition-compatible data types" in the *SQL Reference, Volume 1*

Related samples:

- "dtudt.c -- How to create, use, and drop user-defined distinct types."
- "tbconstr.c -- How to work with constraints associated with tables"
- "tbcreate.c -- How to create, alter and drop tables"
- "dtudt.sqc -- How to create, use, and drop user-defined distinct types (C)"
- "tbconstr.sqc -- How to create, use, and drop constraints (C)"
- "tbcreate.sqc -- How to create and drop tables (C)"
- "tbident.sqc -- How to use identity columns (C)"
- "tbtrig.sqc -- How to use a trigger on a table (C)"
- "dtudt.sqC -- How to create, use, and drop user-defined distinct types (C++)"
- "tbconstr.sqC -- How to create, use, and drop constraints (C++)"
- "tbcreate.sqC -- How to create and drop tables (C++)"
- "tbtrig.sqC -- How to use a trigger on a table (C++)"
- "DtUdt.java -- How to create, use and drop user defined distinct types (JDBC)"
- "TbConstr.java -- How to create, use and drop constraints (JDBC)"
- "TbCreate.java -- How to create and drop tables (JDBC)"
- "TbGenCol.java -- How to use generated columns (JDBC)"
- "TbIdent.java -- How to use Identity Columns (JDBC)"
- "TbTrig.java -- How to use triggers (JDBC)"
- "DtUdt.sqlj -- How to create, use and drop user defined distinct types (SQLj)"
- "TbConstr.sqlj -- How to create, use and drop constraints (SQLj)"
- "TbCreate.sqlj -- How to create and drop tables (SQLj)"
- "TbIdent.sqlj -- How to use Identity Columns (SQLj)"
- "TbTrig.sqlj -- How to use triggers (SQLj)"
- "impexp.sqb -- Export and import tables with table data (MF COBOL)"

CREATE TABLESPACE

The CREATE TABLESPACE statement creates a new table space within the database, assigns containers to the table space, and records the table space definition and attributes in the catalog.

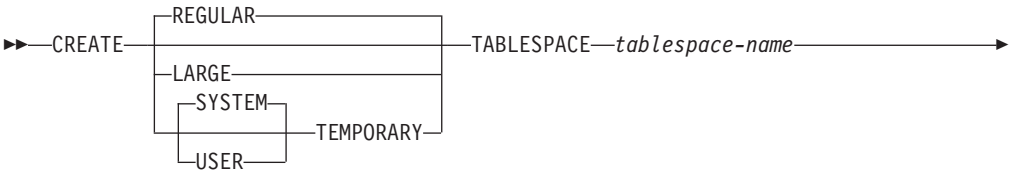
Invocation:

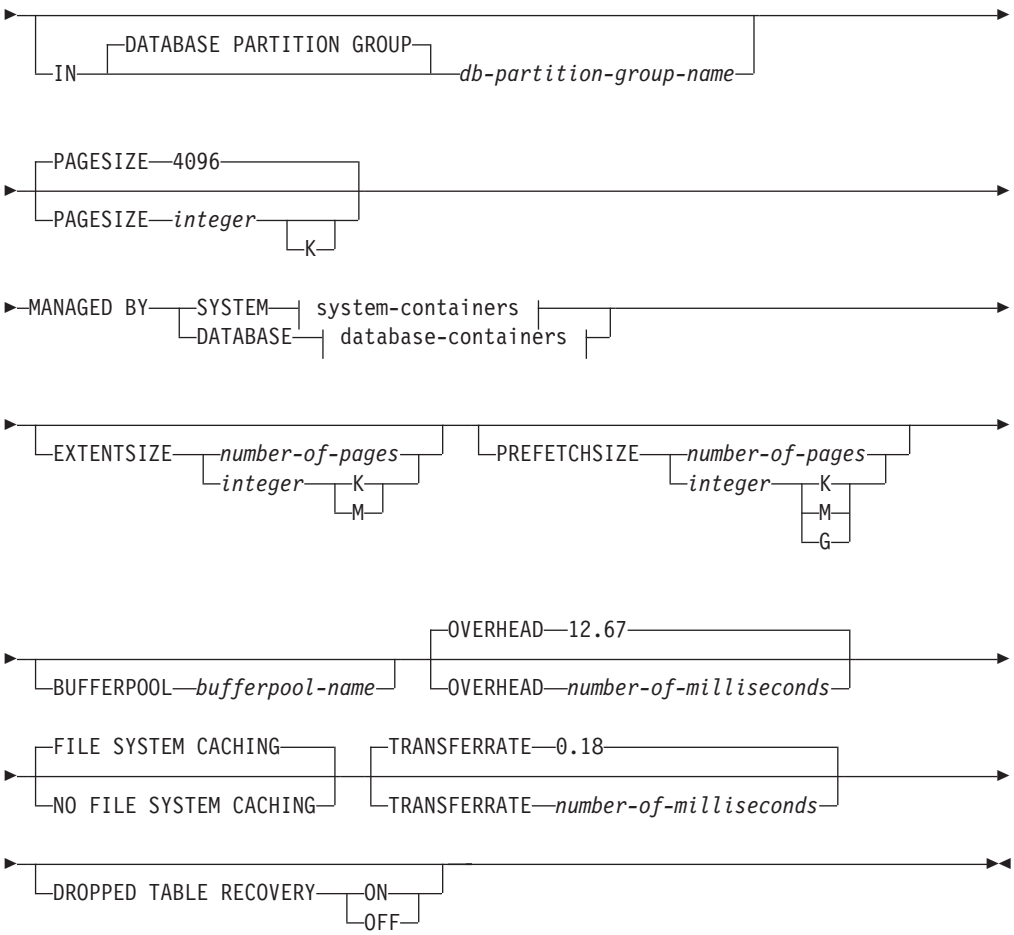
This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization:

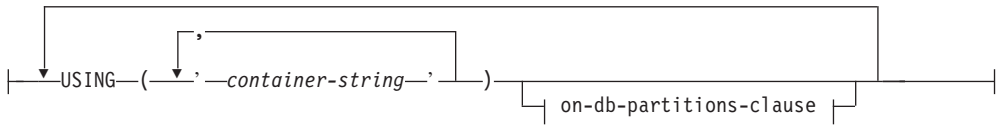
The authorization ID of the statement must have SYSCTRL or SYSADM authority.

Syntax:

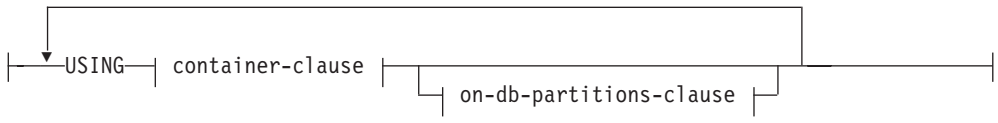




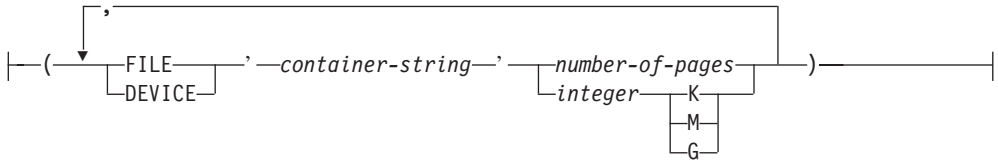
system-containers:



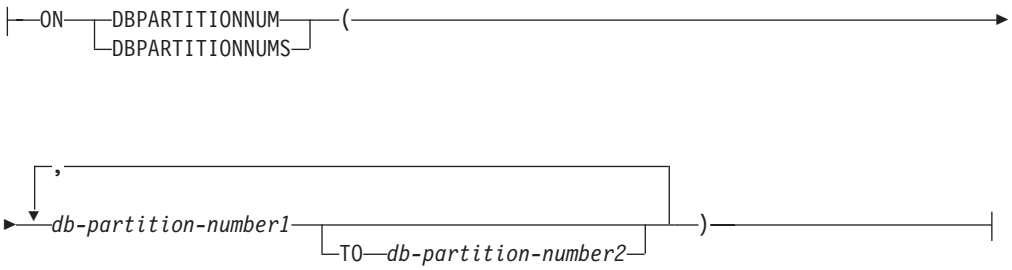
database-containers:



container-clause:



on-db-partitions-clause:



Description:

REGULAR
Stores all data except for temporary tables.

LARGE
Stores long or LOB table columns. It can also store structured type columns or index data. The table space must be a DMS table space.

SYSTEM TEMPORARY
Stores temporary tables (work areas used by the database manager to perform operations such as sorts or joins). The keyword SYSTEM is optional. Note that a database must always have at least one SYSTEM TEMPORARY table space, as temporary tables can only be stored in such a table space. A temporary table space is created automatically when a database is created.

USER TEMPORARY
Stores declared global temporary tables. Note that no user temporary table spaces exist when a database is created. At least one user temporary table space should be created with appropriate USE privileges, to allow definition of declared temporary tables.

tablespace-name
Names the table space. This is a one-part name. It is an SQL identifier (either ordinary or delimited). The *tablespace-name* must not identify a table space that already exists in the catalog (SQLSTATE 42710). The *tablespace-name* must not begin with the characters 'SYS' (SQLSTATE 42939).

IN DATABASE PARTITION GROUP *db-partition-group-name*
Specifies the database partition group for the table space. The database partition group must exist. The only database partition group that can be specified when creating a SYSTEM TEMPORARY table space is IBMTEMPGROUP. The DATABASE PARTITION GROUP keywords are optional.

If the database partition group is not specified, the default database partition group (IBMDEFAULTGROUP) is used for REGULAR, LARGE, and USER TEMPORARY table spaces. For SYSTEM TEMPORARY table spaces, the default database partition group IBMTEMPGROUP is used.

PAGESIZE *integer* **[K]**
Defines the size of pages used for the table space. The valid values for *integer* without the suffix K are 4 096 or 8 192, 16 384, or 32 768. The valid values for *integer* with the suffix K are 4 or 8, 16, or 32. An error occurs if the page size is not one of these values (SQLSTATE 428DE) or the page size is not the same as the page size of the bufferpool associated with the table space (SQLSTATE 428CB). The default is 4 096 byte (4K) pages. Any number of spaces is allowed between *integer* and K, including no space.

MANAGED BY SYSTEM
Specifies that the table space is to be a system managed space (SMS) table space.

system-containers
Specify the containers for an SMS table space.

USING *('container-string',...)*
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. 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 exist, it must not contain any files or subdirectories (SQLSTATE 428B2).

The format of *container-string* is dependent on the operating system. The containers are specified in the normal manner for the operating system. For example, a Windows directory path begins with a drive letter and a ":", while on UNIX-based systems, a path begins with a "/".

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..

on-db-partitions-clause
Specifies the partition or partitions on which the containers are created in a partitioned database. If this clause is not specified, then the containers are created on the partitions in the database partition group that are not explicitly specified in any other *on-db-partitions-clause*s. For a SYSTEM TEMPORARY table space defined on database partition group IBMTEMPGROUP, when the *on-db-partitions-clause* is not specified, the containers will also be created on all new partitions added to the database.

MANAGED BY DATABASE
Specifies that the table space is to be a database managed space (DMS) table space.

database-containers
Specify the containers for a DMS table space.

USING
Introduces a container-clause.

container-clause
Specifies the containers for a DMS table space.

(FILE | DEVICE *'container-string' number-of-pages,...***)**
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. The size 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 pagesize is used to determine the number of pages for the container. A mixture of FILE and DEVICE containers can be specified. The *container-string* cannot exceed 254 bytes in length.

For a FILE container, the *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.

For a DEVICE container, the *container-string* must be a device name. The device must already exist.

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. The containers will be specified in the normal manner for the operating system.

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..

on-db-partitions-clause
Specifies the partition or partitions on which the containers are created in a partitioned database. If this clause is not specified, then the containers are created on the partitions in the database partition group that are not explicitly specified in any other *on-db-partitions-clause*. For a SYSTEM TEMPORARY table space defined on database partition group IBMTEMPGROUP, when the *on-db-partitions-clause* is not specified, the containers will also be created on all new partitions added to the database.

on-db-partitions-clause
Specifies the partitions on which containers are created in a partitioned database.

ON DBPARTITIONNUMS
Keywords that indicate that specific partitions are specified. DBPARTITIONNUM is a synonym for DBPARTITIONNUMS.

db-partition-number1
Specify a database partition number.

TO *db-partition-number2*
Specify a range of partition numbers. The value of *db-partition-number2* must be greater than or equal to the value of *db-partition-number1* (SQLSTATE 428A9). All partitions between and including the specified

partition numbers are included in the partitions for which the containers are created if the partition is included in the database partition group of the table space.

The partition specified by number and every partition in the range of partitions must exist in the database partition group on which the table space is defined (SQLSTATE 42729). A partition-number may only appear explicitly or within a range in exactly one *on-db-partitions-clause* for the statement (SQLSTATE 42613).

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 pagesize is used to determine the number of pages value for prefetch size. Prefetching reads in data needed by a query prior to it being referenced by the query, so that the query need not wait for I/O to be performed.

The default value is provided by the DFT_PREFETCH_SZ configuration parameter.

BUFFERPOOL *bufferpool-name*
The name of the buffer pool used for tables in this table space. The buffer pool must exist (SQLSTATE 42704). If not specified, the default buffer pool (IBMDEFAULTBP) is used. The page size of the bufferpool must match the page size specified (or defaulted) for the table space (SQLSTATE 428CB). The database partition group of the table space must be defined for the bufferpool (SQLSTATE 42735).

OVERHEAD *number-of-milliseconds*
Any numeric literal (integer, decimal, or floating point) that specifies the I/O controller overhead and disk seek and latency time, in milliseconds. The number should be an average for all containers that belong to the table space, if not the same for all containers. This value is used to determine the cost of I/O during query optimization.

FILE SYSTEM CACHING or **NO FILE SYSTEM CACHING**
Specifies whether or not I/O operations are to be cached at the file system level. The default is FILE SYSTEM CACHING.

FILE SYSTEM CACHING
Specifies that all I/O operations in the target table space are to be cached at the file system level.

NO FILE SYSTEM CACHING
Specifies that all I/O operations are to bypass the file system-level cache.

TRANSFERRATE *number-of-milliseconds*
Any numeric literal (integer, decimal, or floating point) that specifies the time to read one page into memory, in milliseconds. The number should be an average for all containers that belong to the table space, if not the same for all containers. This value is used to determine the cost of I/O during query optimization.

DROPPED TABLE RECOVERY
Dropped tables in the specified table space may be recovered using the RECOVER TABLE ON option of the ROLLFORWARD command. This clause can only be specified for a REGULAR table space (SQLSTATE 42613).

Notes:

- ***Compatibilities***
  - For compatibility with previous versions of DB2:
    - NODE can be specified in place of DBPARTITIONNUM
    - NODES can be specified in place of DBPARTITIONNUMS
    - NODEGROUP can be specified in place of DATABASE PARTITION GROUP
    - LONG can be specified in place of LARGE
- Choosing between a database-managed space or a system-managed space for a table space is a fundamental choice involving trade-offs.
- Each container definition requires 53 bytes plus the number of bytes necessary to store the container name. The combined length of all container names for the table space cannot exceed 20 480 bytes (SQLSTATE 54034).
- When more than one TEMPORARY table space exists in the database, they will be used in round-robin fashion in order to balance their usage.
- In a partitioned database, if more than one database partition resides on the same physical node, the same device or specific path cannot be specified for such database partitions (SQLSTATE 42730). For this environment, either specify a unique *container-string* for each database partition or use a relative path name.
- You can specify a database partition expression for container string syntax when creating either SMS or DMS containers. You would typically specify the database partition expression if you were using multiple logical database partitions in the partitioned database system. This ensures that container names are unique across nodes (database partition servers). When you specify the expression, the database partition number is part of the container name or, if you specify additional arguments, the result of the argument is part of the container name.

  You use the argument " $N" ([blank]$N) to indicate a database partition expression. A database partition expression can be used anywhere in the container name, 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 container name after the database partition expression is evaluated. If there is no space character in the container name 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 62. Arguments for Creating Containers. Operators are evaluated from left to right. The database partition number in the examples is assumed to be 5.

Syntax	Example	Value
[blank]$N	" $N"	5

Table 62. Arguments for Creating Containers (continued). Operators are evaluated from left to right. The database partition number in the examples is assumed to be 5.

Syntax	Example	Value
[blank]$N+[number]	" $N+1011"	1016
[blank]$N%[number]	" $N%3" [a]	2
[blank]$N+[number]%[number]	" $N+12%13"	4
[blank]$N%[number]+[number]	" $N%3+20"	22
[a] % is modulus.		

For example:

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

On a two database partition system, the following containers
 would be created:
  /dev/rcont0  - on DATABASE PARTITION 0
  /dev/rcont1  - on DATABASE PARTITION 1


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

On a four database partition system, the following containers
 would be created:
  /DB2/containers/TS2/container100  - on DATABASE PARTITION 0
  /DB2/containers/TS2/container101  - on DATABASE PARTITION 1
  /DB2/containers/TS2/container102  - on DATABASE PARTITION 2
  /DB2/containers/TS2/container103  - on DATABASE PARTITION 3


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

On a two database partition system, the following containers
 would be created:
  /TS3/cont0  - On DATABASE PARTITION 0
  /TS3/cont2  - On DATABASE PARTITION 0
  /TS3/cont1  - On DATABASE PARTITION 1
  /TS3/cont3  - On DATABASE PARTITION 1


If database partition = 5, the containers:
  '/dbdir/node $N /cont1'
  '/ $N+1000 /file1'
  ' $N%10 /container'
  '/dir/ $N%5+2000 /dmscont'

are created as:
  '/dbdir/node5/cont1'
  '/1005/file1'
  '5/container'
  '/dir/2000/dmscont'
```

Examples:

Example 1: Create a regular DMS table space on a UNIX-based system using 3 devices of 10 000 4K pages each. Specify their I/O characteristics.

```
CREATE TABLESPACE PAYROLL
  MANAGED BY DATABASE
  USING (DEVICE'/dev/rhdisk6' 10000,
```

```
    DEVICE '/dev/rhdisk7' 10000,
    DEVICE '/dev/rhdisk8' 10000)
  OVERHEAD 12.67
  TRANSFERRATE 0.18
```

Example 2: Create a regular SMS table space on Windows NT or Windows 2000 using 3 directories on three separate drives, with a 64-page extent size, and a 32-page prefetch size.

```
CREATE TABLESPACE ACCOUNTING
  MANAGED BY SYSTEM
  USING ('d:\acc_tbsp', 'e:\acc_tbsp', 'f:\acc_tbsp')
  EXTENTSIZE 64
  PREFETCHSIZE 32
```

Example 3: Create a temporary DMS table space on Unix using 2 files of 50,000 pages each, and a 256-page extent size.

```
CREATE TEMPORARY TABLESPACE TEMPSPACE2
  MANAGED BY DATABASE
  USING (FILE '/tmp/tempspace2.f1' 50000,
    FILE '/tmp/tempspace2.f2' 50000)
  EXTENTSIZE 256
```

Example 4: Create a DMS table space on database partition group ODDNODEGROUP (partitions 1,3,5) on a Unix partitioned database. On all partitions, use the device /dev/rhdisk0 for 10 000 4K pages. Also specify a partition-specific device for each partition with 40 000 4K pages.

```
CREATE TABLESPACE PLANS
  MANAGED BY DATABASE
  USING (DEVICE '/dev/rhdisk0' 10000, DEVICE '/dev/rn1hd01' 40000)
  ON DBPARTITIONNUM (1)
  USING (DEVICE '/dev/rhdisk0' 10000, DEVICE '/dev/rn3hd03' 40000)
  ON DBPARTITIONNUM (3)
  USING (DEVICE '/dev/rhdisk0' 10000, DEVICE '/dev/rn5hd05' 40000)
  ON DBPARTITIONNUM (5)
```

Related samples:

- "tbtemp.sqc -- How to use a declared temporary table (C)"
- "TbTemp.java -- How to use Declared Temporary Table (JDBC)"

CREATE VIEW

The CREATE VIEW statement creates a view on one or more tables, views or nicknames.

Invocation:

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization:

The privileges held by the authorization ID of the statement must include at least one of the following:

- SYSADM or DBADM authority or
- For each table, view or nickname identified in any fullselect:

- CONTROL privilege on that table or view, or
- SELECT privilege on that table or view

and at least one of the following:

- IMPLICIT_SCHEMA authority on the database, if the implicit or explicit schema name of the view does not exist
- CREATEIN privilege on the schema, if the schema name of the view refers to an existing schema.

If creating a subview, the authorization ID of the statement must:

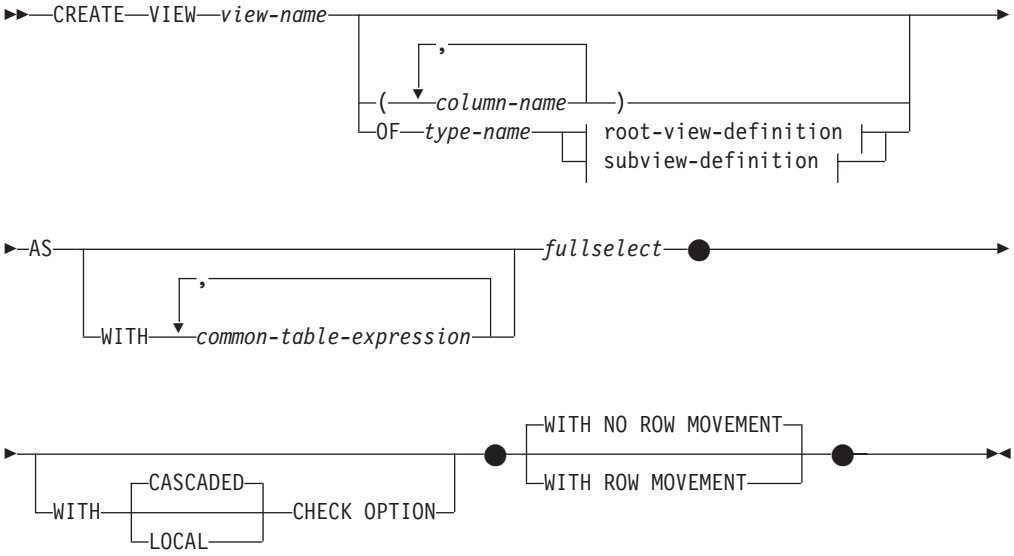
- be the same as the definer of the root table of the table hierarchy.
- have SELECT WITH GRANT on the underlying table of the subview or the superview must not have SELECT privilege granted to any user other than the view definer.

Group privileges are not considered for any table or view specified in the CREATE VIEW statement.

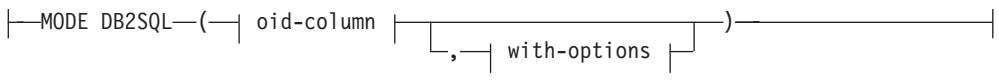
Privileges are not considered when defining a view on federated database nickname. Authorization requirements of the data source for the table or view referenced by the nickname are applied when the query is processed. The authorization ID of the statement may be mapped to a different remote authorization ID.

If a view definer can only create the view because the definer has SYSADM authority, then the definer is granted explicit DBADM authority for the purpose of creating the view.

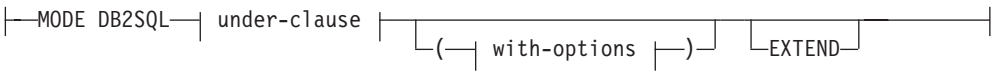
Syntax:



root-view-definition:

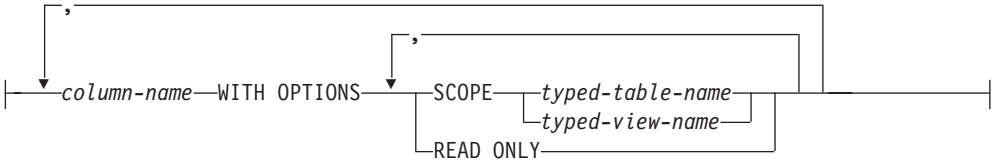


subview-definition:



oid-column:

with-options:



under-clause:

Description:

view-name
Names the view. The name, including the implicit or explicit qualifier, must not identify a table, view, nickname or alias described in the catalog. The qualifier must not be SYSIBM, SYSCAT, SYSFUN, or SYSSTAT (SQLSTATE 42939).

The name can be the same as the name of an inoperative view (see "Inoperative views" on page 665). In this case the new view specified in the CREATE VIEW statement will replace the inoperative view. The user will get a warning (SQLSTATE 01595) when an inoperative view is replaced. No warning is returned if the application was bound with the bind option SQLWARN set to NO.

column-name
Names the columns in the view. If a list of column names is specified, it must consist of as many names as there are columns in the result table of the fullselect. Each *column-name* must be unique and unqualified. If a list of column names is not specified, the columns of the view inherit the names of the columns of the result table of the fullselect.

A list of column names must be specified if the result table of the fullselect has duplicate column names or an unnamed column (SQLSTATE 42908). An unnamed column is a column derived from a constant, function, expression, or set operation that is not named using the AS clause of the select list.

OF *type-name*
Specifies that the columns of the view are based on the attributes of the structured type identified by *type-name*. If *type-name* is specified without a schema name, the type name is resolved by searching the schemas on the SQL path (defined by the FUNCPATH preprocessing option for static SQL and by the CURRENT PATH register for dynamic SQL). The type name must be the

name of an existing user-defined type (SQLSTATE 42704) and it must be a structured type that is instantiable (SQLSTATE 428DP).

MODE DB2SQL
This clause is used to specify the mode of the typed view. This is the only valid mode currently supported.

UNDER *superview-name*
Indicates that the view is a subview of *superview-name*. The superview must be an existing view (SQLSTATE 42704) and the view must be defined using a structured type that is the immediate supertype of *type-name* (SQLSTATE 428DB). The schema name of *view-name* and *superview-name* must be the same (SQLSTATE 428DQ). The view identified by *superview-name* must not have any existing subview already defined using *type-name* (SQLSTATE 42742).

The columns of the view include the object identifier column of the superview with its type modified to be REF(*type-name*), followed by columns based on the attributes of *type-name* (remember that the type includes the attributes of its supertype).

INHERIT SELECT PRIVILEGES
Any user or group holding a SELECT privilege on the superview will be granted an equivalent privilege on the newly created subview. The subview definer is considered to be the grantor of this privilege.

OID-column
Defines the object identifier column for the typed view.

REF IS *OID-column-name* **USER GENERATED**
Specifies that an object identifier (OID) column is defined in the view as the first column. An OID is required for the root view of a view hierarchy (SQLSTATE 428DX). The view must be a typed view (the OF clause must be present) that is not a subview (SQLSTATE 42613). The name for the column is defined as *OID-column-name* and cannot be the same as the name of any attribute of the structured type *type-name* (SQLSTATE 42711). The first column specified in *fullselect* must be of type REF(*type-name*) (you may need to cast it so that it has the appropriate type). If UNCHECKED is not specified, it must be based on a not nullable column on which uniqueness is enforced through an index (primary key, unique constraint, unique index, or OID-column). This column will be referred to as the *object identifier column* or *OID column*. The keywords USER GENERATED indicate that the initial value for the OID column must be provided by the user when inserting a row. Once a row is inserted, the OID column cannot be updated (SQLSTATE 42808).

UNCHECKED
Defines the object identifier column of the typed view definition to assume uniqueness even though the system can not prove this uniqueness. This is intended for use with tables or views that are being defined into a typed view hierarchy where the user knows that the data conforms to this uniqueness rule but it does not comply with the rules that allow the system to prove uniqueness. UNCHECKED option is mandatory for view hierarchies that range over multiple hierarchies or legacy tables or views By specifying UNCHECKED, the user takes responsibility for ensuring that each row of the view has a unique OID. If the user fails to ensure this property, and a view contains duplicate OID values, then a path-expression or DEREF operator involving one of the non-unique OID values may result in an error (SQLSTATE 21000).

with-options
Defines additional options that apply to columns of a typed view.

column-name **WITH OPTIONS**
Specifies the name of the column for which additional options are specified. The *column-name* must correspond to the name of an attribute defined in (not inherited by) the *type-name* of the view. The column must be a reference type (SQLSTATE 42842). It cannot correspond to a column that also exists in the superview (SQLSTATE 428DJ). A column name can only appear in one WITH OPTIONS SCOPE clause in the statement (SQLSTATE 42613).

SCOPE
Identifies the scope of the reference type column. A scope must be specified for any column that is intended to be used as the left operand of a dereference operator or as the argument of the DEREF function.

Specifying the scope for a reference type column may be deferred to a subsequent ALTER VIEW statement (if the scope is not inherited) to allow the target table or view to be defined, usually in the case of mutually referencing views and tables. If no scope is specified for a reference type column of the view and the underlying table or view column was scoped, then the underlying column's scope is inherited by the reference type column. The column remains unscoped if the underlying table or view column did not have a scope. See 663 for more information about scope and reference type columns.

typed-table-name
The name of a typed table. The table must already exist or be the same as the name of the table being created (SQLSTATE 42704). The data type of *column-name* must be REF(*S*), where *S* is the type of *typed-table-name* (SQLSTATE 428DM). No checking is done of any existing values in *column-name* to ensure that the values actually reference existing rows in *typed-table-name*.

typed-view-name
The name of a typed view. The view must already exist or be the same as the name of the view being created (SQLSTATE 42704). The data type of *column-name* must be REF(*S*), where *S* is the type of *typed-view-name* (SQLSTATE 428DM). No checking is done of any existing values in *column-name* to ensure that the values actually reference existing rows in *typed-view-name*.

READ ONLY
Identifies the column as a read-only column. This option is used to force a column to be read-only so that subview definitions can specify an expression for the same column that is implicitly read-only.

AS
Identifies the view definition.

WITH *common-table-expression*
Defines a common table expression for use with the fullselect that follows. A common table expression cannot be specified when defining a typed view.

fullselect
Defines the view. At any time, the view consists of the rows that would result if the SELECT statement were executed. The fullselect must not reference host variables, parameter markers, or declared temporary tables. However, a parameterized view can be created as an SQL table function.

The fullselect cannot include an SQL data change statement in the FROM clause (SQLSTATE 428FL).

For Typed Views and Subviews: The *fullselect* must conform to the following rules otherwise an error is returned (SQLSTATE 428EA unless otherwise specified).

- The fullselect must not include references to the DBPARTITIONNUM or HASHEDVALUE functions, non-deterministic functions, or functions defined to have external action.
- The body of the view must consist of a single subselect, or a UNION ALL of two or more subselects. Let each of the subselects participating directly in the view body be called a *branch* of the view. A view may have one or more branches.
- The FROM-clause of each branch must consist of a single table or view (not necessarily typed), called the *underlying* table or view of that branch.
- The underlying table or view of each branch must be in a separate hierarchy (that is, a view cannot have multiple branches with their underlying tables or views in the same hierarchy).
- None of the branches of a typed view definition may specify GROUP BY or HAVING.
- If the view body contains UNION ALL, then the root view in the hierarchy must specify the UNCHECKED option for its OID column.

For a hierarchy of views and subviews: Let BR1 and BR2 be any branches that appear in the definitions of views in the hierarchy. Let T1 be the underlying table or view of BR1, and let T2 be the underlying table or view of BR2. Then:

- If T1 and T2 are not in the same hierarchy, then the root view in the view hierarchy must specify the UNCHECKED option for its OID column.
- If T1 and T2 are in the same hierarchy, then BR1 and BR2 must contain predicates or ONLY-clauses that are sufficient to guarantee that their row-sets are disjoint.

For typed subviews defined using EXTEND AS: For every branch in the body of the subview:

- The underlying table of each branch must be a (not necessarily proper) subtable of some underlying table of the immediate superview.
- The expressions in the SELECT list must be assignable to the non-inherited columns of the subview (SQLSTATE 42854).

For typed subviews defined using AS without EXTEND:

- For every branch in the body of the subview, the expressions in the SELECT-list must be assignable to the declared types of the inherited and non-inherited columns of the subview (SQLSTATE 42854).
- The OID-expression of each branch over a given hierarchy in the subview must be equivalent (except for casting) to the OID-expression in the branch over the same hierarchy in the root view.
- The expression for a column not defined (implicitly or explicitly) as READ ONLY in a superview must be equivalent in all branches over the same underlying hierarchy in its subviews.

WITH CHECK OPTION

Specifies the constraint that every row that is inserted or updated through the

view must conform to the definition of the view. A row that does not conform to the definition of the view is a row that does not satisfy the search conditions of the view.

WITH CHECK OPTION must not be specified if any of the following conditions is true:

- The view is read-only (SQLSTATE 42813). If WITH CHECK OPTION is specified for an updatable view that does not allow inserts, the constraint applies to updates only.
- The view references the NODENUMBER or PARTITION function, a non-deterministic function, or a function with external action (SQLSTATE 42997).
- A nickname is the update target of the view.
- A view that has an INSTEAD OF trigger defined on it is the update target of the view (SQLSTATE 428FQ).

If WITH CHECK OPTION is omitted, the definition of the view is not used in the checking of any insert or update operations that use the view. Some checking might still occur during insert or update operations if the view is directly or indirectly dependent on another view that includes WITH CHECK OPTION. Because the definition of the view is not used, rows might be inserted or updated through the view that do not conform to the definition of the view.

CASCADED
The WITH CASCADED CHECK OPTION constraint on a view *V* means that *V* inherits the search conditions as constraints from any updatable view on which *V* is dependent. Furthermore, every updatable view that is dependent on *V* is also subject to these constraints. Thus, the search conditions of *V* and each view on which *V* is dependent are ANDed together to form a constraint that is applied for an insert or update of *V* or of any view dependent on *V*.

LOCAL
The WITH LOCAL CHECK OPTION constraint on a view *V* means the search condition of *V* is applied as a constraint for an insert or update of *V* or of any view that is dependent on *V*.

The difference between CASCADED and LOCAL is shown in the following example. Consider the following updatable views (substituting for Y from column headings of the table that follows):

```
V1 defined on table T
V2 defined on V1 WITH Y CHECK OPTION
V3 defined on V2
V4 defined on V3 WITH Y CHECK OPTION
V5 defined on V4
```

The following table shows the search conditions against which inserted or updated rows are checked:

	Y is LOCAL	**Y is CASCADED**
V1 checked against:	no view	no view
V2 checked against:	V2	V2, V1
V3 checked against:	V2	V2, V1
V4 checked against:	V2, V4	V4, V3, V2, V1
V5 checked against:	V2, V4	V4, V3, V2, V1

Consider the following updatable view which shows the impact of the WITH CHECK OPTION using the default CASCADED option:

```
CREATE VIEW V1 AS SELECT COL1 FROM T1 WHERE COL1 > 10

CREATE VIEW V2 AS SELECT COL1 FROM V1 WITH CHECK OPTION

CREATE VIEW V3 AS SELECT COL1 FROM V2 WHERE COL1 < 100
```

The following INSERT statement using *V1* will succeed because *V1* does not have a WITH CHECK OPTION and *V1* is not dependent on any other view that has a WITH CHECK OPTION.

```
INSERT INTO V1 VALUES(5)
```

The following INSERT statement using *V2* will result in an error because *V2* has a WITH CHECK OPTION and the insert would produce a row that did not conform to the definition of *V2*.

```
INSERT INTO V2 VALUES(5)
```

The following INSERT statement using *V3* will result in an error even though it does not have WITH CHECK OPTION because *V3* is dependent on *V2* which does have a WITH CHECK OPTION (SQLSTATE 44000).

```
INSERT INTO V3 VALUES(5)
```

The following INSERT statement using *V3* will succeed even though it does not conform to the definition of *V3* (*V3* does not have a WITH CHECK OPTION); it does conform to the definition of *V2* which does have a WITH CHECK OPTION.

```
INSERT INTO V3 VALUES(200)
```

WITH NO ROW MOVEMENT or **WITH ROW MOVEMENT**

Specifies the action to take for an updatable UNION ALL view when a row is updated in a way that violates a check constraint on the underlyig table. The default is WITH NO ROW MOVEMENT.

WITH NO ROW MOVEMENT

Specifies that an error (SQLSTATE 23513) is to be returned if a row is updated in a way that violates a check constraint on the underlying table.

WITH ROW MOVEMENT

Specifies that an updated row is to be moved to the appropriate underlying table, even if it violates a check constraint on that table.

Row movement involves deletion of the rows that violate the check constraint, and insertion of those rows back into the view. The WITH ROW MOVEMENT clause can only be specified for UNION ALL views whose columns are all updatable (SQLSTATE 429BJ). If a row is inserted (perhaps after trigger activation) into the same underlying table from which it was deleted, an error is returned (SQLSTATE 23524). A view defined using the WITH ROW MOVEMENT clause must not contain nested UNION ALL operations, except in the outermost fullselect (SQLSTATE 429BJ).

Notes:

- *Compatibilities:*
  - For compatibility with previous versions of DB2:

- The FEDERATED keyword can be specified between the keywords CREATE and VIEW. The FEDERATED keyword is ignored, however, because a warning is no longer returned if federated objects are used in the view definition.

- Creating a view with a schema name that does not already exist will result in the implicit creation of that schema provided the authorization ID of the statement has IMPLICIT_SCHEMA authority. The schema owner is SYSIBM. The CREATEIN privilege on the schema is granted to PUBLIC.
- View columns inherit the NOT NULL WITH DEFAULT attribute from the base table or view except when columns are derived from an expression. When a row is inserted or updated into an updatable view, it is checked against the constraints (primary key, referential integrity, and check) if any are defined on the base table.
- A new view cannot be created if it uses an inoperative view in its definition. (SQLSTATE 51024).
- This statement does not support declared temporary tables (SQLSTATE 42995).
- ***Deletable views:*** A view is *deletable* if an INSTEAD OF trigger for the delete operation has been defined for the view, or if all of the following are true:
  - each FROM clause of the outer fullselect identifies only one base table (with no OUTER clause), deletable view (with no OUTER clause), deletable nested table expression, or deletable common table expression (cannot identify a nickname)
  - the outer fullselect does not include a VALUES clause
  - the outer fullselect does not include a GROUP BY clause or HAVING clause
  - the outer fullselect does not include column functions in the select list
  - the outer fullselect does not include SET operations (UNION, EXCEPT or INTERSECT) with the exception of UNION ALL
  - the base tables in the operands of a UNION ALL must not be the same table and each operand must be deletable
  - the select list of the outer fullselect does not include DISTINCT
  - the FROM clause of the outer fullselect does not include a *data-change-table-reference*
- ***Updatable views:*** A column of a view is *updatable* if an INSTEAD OF trigger for the update operation has been defined for the view, or if all of the following are true:
  - the view is deletable (independent of an INSTEAD OF trigger for delete), the column resolves to a column of a base table (not using a dereference operation), and the READ ONLY option is not specified
  - all the corresponding columns of the operands of a UNION ALL have exactly matching data types (including length or precision and scale) and matching default values if the fullselect of the view includes a UNION ALL

  A view is updatable if *any* column of the view is updatable.
- ***Insertable views:***
  - A view is insertable if an INSTEAD OF trigger for the insert operation has been defined for the view, or at least one column of the view is updatable (independent of an INSTEAD OF trigger for update), and the fullselect of the view does not include UNION ALL.
  - A given row can be inserted into a view (including a UNION ALL) if, and only if, it fulfills the check constraints of exactly one of the underlying base tables.

  - To insert into a view that includes non-updatable columns, those columns must be omitted from the column list.
- ***Read-only views:*** A view is *read-only* if it is *not* deletable, updatable, or insertable.

  The READONLY column in the SYSCAT.VIEWS catalog view indicates if a view is read-only without considering INSTEAD OF triggers.
- Common table expressions and nested table expressions follow the same set of rules for determining whether they are deletable, updatable, insertable, or read-only.
- ***Inoperative views:*** An *inoperative view* is a view that is no longer available for SQL statements. A view becomes inoperative if:
  - A privilege, upon which the view definition is dependent, is revoked.
  - An object such as a table, nickname, alias or function, upon which the view definition is dependent, is dropped.
  - A view, upon which the view definition is dependent, becomes inoperative.
  - A view that is the superview of the view definition (the subview) becomes inoperative.

  In practical terms, an inoperative view is one in which the view definition has been unintentionally dropped. For example, when an alias is dropped, any view defined using that alias is made inoperative. All dependent views also become inoperative and packages dependent on the view are no longer valid.

  Until the inoperative view is explicitly recreated or dropped, a statement using that inoperative view cannot be compiled (SQLSTATE 51024) with the exception of the CREATE ALIAS, CREATE VIEW, DROP VIEW, and COMMENT ON TABLE statements. Until the inoperative view has been explicitly dropped, its qualified name cannot be used to create another table or alias (SQLSTATE 42710).

  An inoperative view may be recreated by issuing a CREATE VIEW statement using the definition text of the inoperative view. This view definition text is stored in the TEXT column of the SYSCAT.VIEWS catalog. When recreating an inoperative view, it is necessary to explicitly grant any privileges required on that view by others, due to the fact that all authorization records on a view are deleted if the view is marked inoperative. Note that there is no need to explicitly drop the inoperative view in order to recreate it. Issuing a CREATE VIEW statement with the same *view-name* as an inoperative view will cause that inoperative view to be replaced, and the CREATE VIEW statement will return a warning (SQLSTATE 01595).

  Inoperative views are indicated by an X in the VALID column of the SYSCAT.VIEWS catalog view and an X in the STATUS column of the SYSCAT.TABLES catalog view.
- ***Privileges:***

  The definer of a view always receives the SELECT privilege on the view as well as the right to drop the view. The definer of a view will get CONTROL privilege on the view only if the definer has CONTROL privilege on every base table, view, or nickname identified in the fullselect, or if the definer has SYSADM or DBADM authority.

  The definer of the view is granted INSERT, UPDATE, column level UPDATE or DELETE privileges on the view if the view is not read-only and the definer has the corresponding privileges on the underlying objects.

For a view defined WITH ROW MOVEMENT, the definer acquires the UPDATE privilege on the view only if the definer has the UPDATE privilege on all columns of the view, as well as INSERT and DELETE privileges on all underlying tables or views.

The definer of a view only acquires privileges if the privileges from which they are derived exist at the time the view is created. The definer must have these privileges either directly or because PUBLIC has these privilege. Privileges are not considered when defining a view on a federated server nickname. However, when using a view on a nickname, the user's authorization ID must have valid select privileges on the table or view that the nickname references at the data source. Otherwise, an error is returned. Privileges held by groups of which the view definer is a member, are not considered.

When a subview is created, the SELECT privileges held on the immediate superview are automatically granted on the subview.

- ***Scope and REF columns:***

  When selecting a reference type column in the fullselect of a view definition, consider the target type and scope that is required.

  – If the required target type and scope is the same as the underlying table or view, the column can simply be selected.

  – If the scope needs to be changed, use the WITH OPTIONS SCOPE clause to define the required scope table or view.

  – If the target type of the reference needs to be changed, the column must be cast first to the representation type of the reference and then to the new reference type. The scope in this case can be specified in the cast to the reference type or using the WITH OPTIONS SCOPE clause. For example, assume you select column Y defined as REF(TYP1) SCOPE TAB1. You want this to be defined as REF(VTYP1) SCOPE VIEW1. The select list item would be as follows:

    ```
    CAST(CAST(Y AS VARCHAR(16) FOR BIT DATA) AS REF(VTYP1) SCOPE VIEW1)
    ```

- ***Identity columns:*** A column of a view is considered an identity column, if the element of the corresponding column in the fullselect of the view definition is the name of an identity column of a table, or the name of a column of a view which directly or indirectly maps to the name of an identity column of a base table.

  In all other cases, the columns of a view will not get the identity property. For example:

  – the select-list of the view definition includes multiple instances of the name of an identity column (that is, selecting the same column more than once)

  – the view definition involves a join

  – a column in the view definition includes an expression that refers to an identity column

  – the view definition includes a UNION

  When inserting into a view for which the select list of the view definition directly or indirectly includes the name of an identity column of a base table, the same rules apply as if the INSERT statement directly referenced the identity column of the base table.

- ***Federated views:*** A federated view is a view that includes a reference to a nickname somewhere in the fullselect. The presence of such a nickname changes the authorization model used for the view when the view is subsequently referenced in a query.

  When the view is created, no privilege checking is done to determine whether the view definer has access to the underlying data source table or view of a

nickname. Privilege checking of references to tables or views at the federated database are handled as usual, requiring the view definer to have at least SELECT privilege on such objects.

When a federated view is subsequently referenced in a query, the nicknames result in queries against the data source, and the authorization ID that issued the query (or the remote authorization ID to which it maps) must have the necessary privileges to access the data source table or view. The authorization ID that issues the query referencing the federated view is not required to have any additional privileges on tables or views (non-federated) that exist at the federated server.

- ***ROW MOVEMENT, triggers and constraints:*** When a view that is defined using the WITH ROW MOVEMENT clause is updated, the sequence of trigger and constraints operations is as follows:
  1. BEFORE UPDATE triggers are activated for all rows being updated, including rows that will eventually be moved.
  2. The update operation is processed.
  3. Constraints are processed for all updated rows.
  4. AFTER UPDATE triggers (both row-level and statement-level) are activated in creation order, for all rows that satisfy the constraints after the update operation. Because this is an UPDATE statement, all UPDATE statement-level triggers are activated for all underlying tables.
  5. BEFORE DELETE triggers are activated for all rows that did not satisfy the constraints after the update operation (these are the rows that are to be moved).
  6. The delete operation is processed.
  7. Constraints are processed for all deleted rows.
  8. AFTER DELETE triggers (both row-level and statement-level) are activated in creation order, for all deleted rows. Statement-level triggers are activated for only those tables that are involved in the delete operation.
  9. BEFORE INSERT triggers are activated for all rows being inserted (that is, the rows being moved). The new transition tables for the BEFORE INSERT triggers contain the input data provided by the user.
  10. The insert operation is processed.
  11. Constraints are processed for all inserted rows.
  12. AFTER INSERT triggers (both row-level and statement-level) are activated in creation order, for all inserted rows. Statement-level triggers are activated for only those tables that are involved in the insert operation.
- ***Nested UNION ALL views:*** A view defined with UNION ALL and based, either directly or indirectly, on a view that is also defined with UNION ALL cannot be updated if either view is defined using the WITH ROW MOVEMENT clause (SQLSTATE 429BK).

Examples:

Example 1: Create a view named MA_PROJ upon the PROJECT table that contains only those rows with a project number (PROJNO) starting with the letters 'MA'.

```
CREATE VIEW MA_PROJ  AS SELECT *
  FROM PROJECT
   WHERE SUBSTR(PROJNO, 1, 2) = 'MA'
```

Example 2: Create a view as in example 1, but select only the columns for project number (PROJNO), project name (PROJNAME) and employee in charge of the project (RESPEMP).

```
CREATE VIEW MA_PROJ
  AS SELECTPROJNO, PROJNAME, RESPEMP
  FROM PROJECT
  WHERE SUBSTR(PROJNO, 1, 2) = 'MA'
```

Example 3: Create a view as in example 2, but, in the view, call the column for the employee in charge of the project IN_CHARGE.

```
CREATE VIEW MA_PROJ
  (PROJNO, PROJNAME, IN_CHARGE)
  AS SELECTPROJNO, PROJNAME, RESPEMP
  FROM PROJECT
  WHERE SUBSTR(PROJNO, 1, 2) = 'MA'
```

Note: Even though only one of the column names is being changed, the names of all three columns in the view must be listed in the parentheses that follow MA_PROJ.

Example 4: Create a view named PRJ_LEADER that contains the first four columns (PROJNO, PROJNAME, DEPTNO, RESPEMP) from the PROJECT table together with the last name (LASTNAME) of the person who is responsible for the project (RESPEMP). Obtain the name from the EMPLOYEE table by matching EMPNO in EMPLOYEE to RESPEMP in PROJECT.

```
CREATE VIEW PRJ_LEADER
  AS SELECT PROJNO, PROJNAME, DEPTNO, RESPEMP, LASTNAME
  FROM PROJECT, EMPLOYEE
  WHERE RESPEMP = EMPNO
```

Example 5: Create a view as in example 4, but in addition to the columns PROJNO, PROJNAME, DEPTNO, RESPEMP, and LASTNAME, show the total pay (SALARY + BONUS + COMM) of the employee who is responsible. Also select only those projects with mean staffing (PRSTAFF) greater than one.

```
CREATE VIEW PRJ_LEADER
(PROJNO, PROJNAME, DEPTNO, RESPEMP, LASTNAME, TOTAL_PAY )
AS SELECT PROJNO, PROJNAME, DEPTNO, RESPEMP, LASTNAME, SALARY+BONUS+COMM
  FROM PROJECT, EMPLOYEE
  WHERE RESPEMP = EMPNO
  AND PRSTAFF > 1
```

Specifying the column name list could be avoided by naming the expression SALARY+BONUS+COMM as TOTAL_PAY in the fullselect.

```
CREATE VIEW PRJ_LEADER
  AS SELECT PROJNO, PROJNAME, DEPTNO, RESPEMP,
              LASTNAME, SALARY+BONUS+COMM AS TOTAL_PAY
    FROM PROJECT, EMPLOYEE
    WHERE RESPEMP = EMPNO AND PRSTAFF > 1
```

Example 6: Given the set of tables and views shown in the following figure:

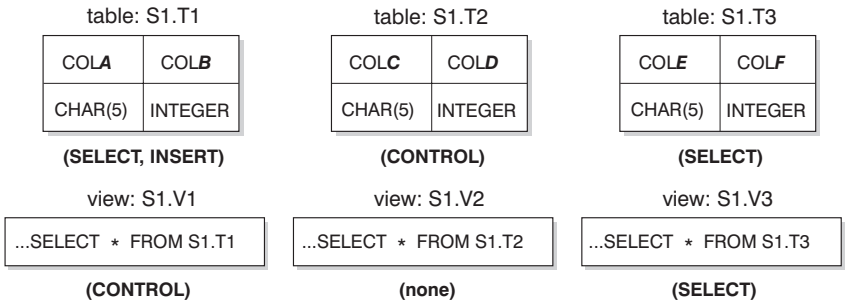


Figure 12. Tables and Views for Example 6

User ZORPIE (who does not have either DBADM or SYSADM authority) has been granted the privileges shown in brackets below each object:

1. ZORPIE will get CONTROL privilege on the view that she creates with:

   ```
   CREATE VIEW VA AS SELECT * FROM S1.V1
   ```

   because she has CONTROL on S1.V1. (CONTROL on S1.V1 must have been granted to ZORPIE by someone with DBADM or SYSADM authority.) It does not matter which, if any, privileges she has on the underlying base table.
2. ZORPIE will not be allowed to create the view:

   ```
   CREATE VIEW VB AS SELECT * FROM S1.V2
   ```

   because she has neither CONTROL nor SELECT on S1.V2. It does not matter that she has CONTROL on the underlying base table (S1.T2).
3. ZORPIE will get CONTROL privilege on the view that she creates with:

   ```
   CREATE VIEW VC (COLA, COLB, COLC, COLD)
     AS SELECT * FROM S1.V1, S1.T2
     WHERE COLA = COLC
   ```

   because the fullselect of ZORPIE.VC references view S1.V1 and table S1.T2 and she has CONTROL on both of these. Note that the view VC is read-only, so ZORPIE does not get INSERT, UPDATE or DELETE privileges.
4. ZORPIE will get SELECT privilege on the view that she creates with:

   ```
   CREATE VIEW VD (COLA,COLB, COLE, COLF)
     AS SELECT * FROM S1.V1, S1.V3
     WHERE COLA = COLE
   ```

   because the fullselect of ZORPIE.VD references the two views S1.V1 and S1.V3, one on which she has only SELECT privilege, and one on which she has CONTROL privilege. She is given the lesser of the two privileges, SELECT, on ZORPIE.VD.
5. ZORPIE will get INSERT, UPDATE and DELETE privilege WITH GRANT OPTION and SELECT privilege on the view VE in the following view definition.

   ```
   CREATE VIEW VE
      AS SELECT * FROM S1.V1
     WHERE COLA > ANY
           (SELECT COLE FROM S1.V3)
   ```

   ZORPIE's privileges on VE are determined primarily by her privileges on S1.V1. Since S1.V3 is only referenced in a subquery, she only needs SELECT privilege on S1.V3 to create the view VE. The definer of a view only gets

CONTROL on the view if they have CONTROL on all objects referenced in the view definition. ZORPIE does not have CONTROL on S1.V3, consequently she does not get CONTROL on VE.

Related reference:

- "CREATE FUNCTION (SQL Scalar, Table, or Row) statement" in the *SQL Reference, Volume 2*
- "SQL queries" in the *SQL Reference, Volume 1*

DELETE

The DELETE statement deletes rows from a table, nickname, or view, or the underlying tables, nicknames, or views of the specified fullselect. Deleting a row from a nickname deletes the row from the data source object to which the nickname refers. Deleting a row from a view deletes the row from the table on which the view is based if no INSTEAD OF trigger is defined for the delete operation on this view. If such a trigger is defined, the trigger will be executed instead.

There are two forms of this statement:

- The *Searched* DELETE form is used to delete one or more rows (optionally determined by a search condition).
- The *Positioned* DELETE form is used to delete exactly one row (as determined by the current position of a cursor).

Invocation:

A DELETE statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared.

Authorization:

To execute either form of this statement, the privileges held by the authorization ID of the statement must include at least one of the following:

- DELETE privilege on the table, view, or nickname for which rows are to be deleted
- CONTROL privilege on the table, view, or nickname for which rows are to be deleted
- SYSADM or DBADM authority.

To execute a Searched DELETE statement, the privileges held by the authorization ID of the statement must also include at least one of the following for each table, view, or nickname referenced by a subquery:

- SELECT privilege
- CONTROL privilege
- SYSADM or DBADM authority.

If the package used to process the statement is precompiled with SQL92 rules (option LANGLEVEL with a value of SQL92E or MIA), and the searched form of a DELETE statement includes a reference to a column of the table or view in the *search-condition*, the privileges held by the authorization ID of the statement must also include at least one of the following:

- SELECT privilege
- CONTROL privilege
- SYSADM or DBADM authority.

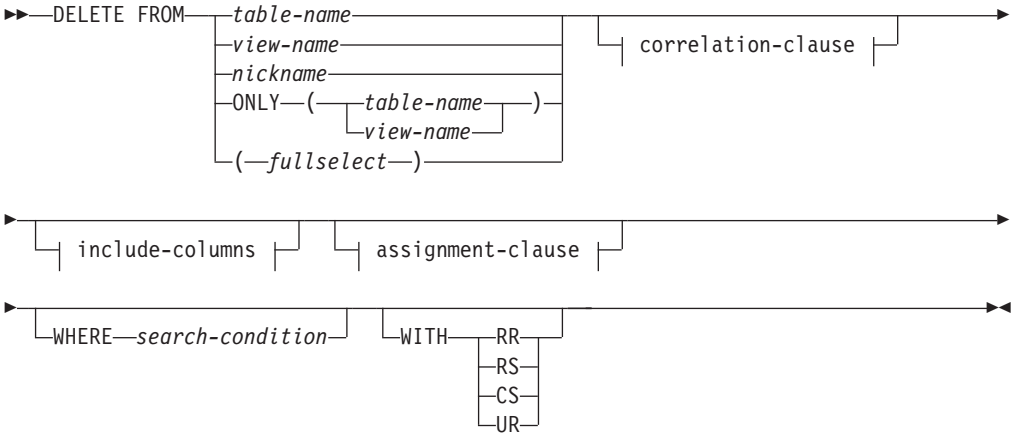
If the specified table or view is preceded by the ONLY keyword, the privileges held by the authorization ID of the statement must also include the SELECT privilege for every subtable or subview of the specified table or view.

Group privileges are not checked for static DELETE statements.

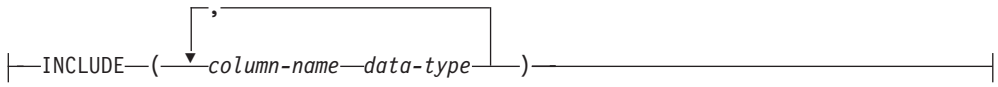
If the target of the delete operation is a nickname, the privileges on the object at the data source are not considered until the statement is executed at the data source. At this time, the authorization ID that is used to connect to the data source must have the privileges required for the operation on the object at the data source. The authorization ID of the statement may be mapped to a different authorization ID at the data source.

Syntax:

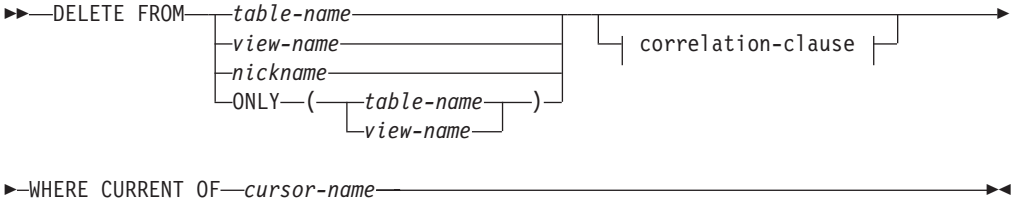
searched-delete:



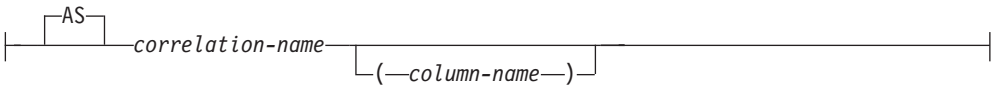
include-columns:



positioned-delete:



correlation-clause:



Description:

FROM *table-name*, *view-name*, *nickname*, **or** *(fullselect)*
Identifies the object of the delete operation. The name must identify a table or view that exists in the catalog, but it must not identify a catalog table, a catalog view, a system-maintained materialized query table, or a read-only view.

If *table-name* is a typed table, rows of the table or any of its proper subtables may get deleted by the statement.

If *view-name* is a typed view, rows of the underlying table or underlying tables of the view's proper subviews may get deleted by the statement. If *view-name* is a regular view with an underlying table that is a typed table, rows of the typed table or any of its proper subtables may get deleted by the statement.

If the object of the delete operation is a fullselect, the fullselect must be deletable, as defined in the "Deletable views" Notes item in the description of the CREATE VIEW statement.

Only the columns of the specified table can be referenced in the WHERE clause. For a positioned DELETE, the associated cursor must also have specified the table or view in the FROM clause without using ONLY.

FROM ONLY (*table-name***)**
Applicable to typed tables, the ONLY keyword specifies that the statement should apply only to data of the specified table and rows of proper subtables cannot be deleted by the statement. For a positioned DELETE, the associated cursor must also have specified the table in the FROM clause using ONLY. If *table-name* is not a typed table, the ONLY keyword has no effect on the statement.

FROM ONLY (*view-name***)**
Applicable to typed views, the ONLY keyword specifies that the statement should apply only to data of the specified view and rows of proper subviews cannot be deleted by the statement. For a positioned DELETE, the associated cursor must also have specified the view in the FROM clause using ONLY. If *view-name* is not a typed view, the ONLY keyword has no effect on the statement.

correlation-clause
Can be used within the *search-condition* to designate a table, view, nickname, or fullselect. For a description of *correlation-clause*, see "table-reference" in the description of "Subselect".

include-columns
Specifies a set of columns that are included, along with the columns of *table-name* or *view-name*, in the intermediate result table of the DELETE statement when it is nested in the FROM clause of a fullselect. The *include-columns* are appended at the end of the list of columns that are specified for *table-name* or *view-name*.

INCLUDE
Specifies a list of columns to be included in the intermediate result table of the DELETE statement.

column-name
Specifies a column of the intermediate result table of the DELETE statement. The name cannot be the same as the name of another include column or a column in *table-name* or *view-name* (SQLSTATE 42711).

data-type
Specifies the data type of the include column. The data type must be one that is supported by the CREATE TABLE statement.

assignment-clause
See the description of *assignment-clause* under the UPDATE statement. The same rules apply. The *include-columns* are the only columns that can be set using the *assignment-clause* (SQLSTATE 42703).

WHERE
Specifies a condition that selects the rows to be deleted. The clause can be omitted, a search condition specified, or a cursor named. If the clause is omitted, all rows of the table or view are deleted.

search-condition
Each *column-name* in the search condition, other than in a subquery must identify a column of the table or view.

The *search-condition* is applied to each row of the table, view, or nickname, and the deleted rows are those for which the result of the *search-condition* is true.

If the search condition contains a subquery, the subquery can be thought of as being executed each time the *search condition* is applied to a row, and the results used in applying the *search condition*. In actuality, a subquery with no correlated references is executed once, whereas a subquery with a correlated reference may have to be executed once for each row. If a subquery refers to the object table of a DELETE statement or a dependent table with a delete rule of CASCADE or SET NULL, the subquery is completely evaluated before any rows are deleted.

CURRENT OF *cursor-name*
Identifies a cursor that is defined in a DECLARE CURSOR statement of the program. The DECLARE CURSOR statement must precede the DELETE statement.

The table, view, or nickname named must also be named in the FROM clause of the SELECT statement of the cursor, and the result table of the cursor must not be read-only. (For an explanation of read-only result tables, see "DECLARE CURSOR".)

When the DELETE statement is executed, the cursor must be positioned on a row: that row is the one deleted. After the deletion, the cursor is positioned before the next row of its result table. If there is no next row, the cursor is positioned after the last row.

WITH
Specifies the isolation level used when locating the rows to be deleted.

RR
Repeatable Read

RS
Read Stability

CS
Cursor Stability

UR
Uncommitted Read

The default isolation level of the statement is the isolation level of the package in which the statement is bound.

Rules:

- ***Triggers:*** DELETE statements may cause triggers to be executed. A trigger may cause other statements to be executed, or may raise error conditions based on the deleted rows. If a DELETE statement on a view causes an INSTEAD OF trigger to fire, referential integrity will be checked against the updates performed in the trigger, and not against the underlying tables of the view that caused the trigger to fire.
- ***Referential Integrity:*** If the identified table or the base table of the identified view is a parent, the rows selected for delete must not have any dependents in a relationship with a delete rule of RESTRICT, and the DELETE must not cascade to descendent rows that have dependents in a relationship with a delete rule of RESTRICT.

  If the delete operation is not prevented by a RESTRICT delete rule, the selected rows are deleted. Any rows that are dependents of the selected rows are also affected:

  – The nullable columns of the foreign keys of any rows that are their dependents in a relationship with a delete rule of SET NULL are set to the null value.
  – Any rows that are their dependents in a relationship with a delete rule of CASCADE are also deleted, and the above rules apply, in turn, to those rows.

  The delete rule of NO ACTION is checked to enforce that any non-null foreign key refers to an existing parent row after the other referential constraints have been enforced.

Notes:

- If an error occurs during the execution of a multiple row DELETE, no changes are made to the database.
- Unless appropriate locks already exist, one or more exclusive locks are acquired during the execution of a successful DELETE statement. Issuing a COMMIT or ROLLBACK statement will release the locks. Until the locks are released by a commit or rollback operation, the effect of the delete operation can only be perceived by:

  – The application process that performed the deletion
  – Another application process using isolation level UR.

  The locks can prevent other application processes from performing operations on the table.
- If an application process deletes a row on which any of its cursors are positioned, those cursors are positioned before the next row of their result table. Let C be a cursor that is positioned before row R (as a result of an OPEN, a DELETE through C, a DELETE through some other cursor, or a searched DELETE). In the presence of INSERT, UPDATE, and DELETE operations that affect the base table from which R is derived, the next FETCH operation referencing C does not necessarily position C on R. For example, the operation can position C on R', where R' is a new row that is now the next row of the result table.

- SQLERRD(3) in the SQLCA shows the number of rows that qualified for the delete operation. In the context of an SQL procedure statement, the value can be retrieved using the ROW_COUNT variable of the GET DIAGNOSTICS statement. SQLERRD(5) in the SQLCA shows the number of rows affected by referential constraints and by triggered statements. It includes rows that were deleted as a result of a CASCADE delete rule and rows in which foreign keys were set to NULL as the result of a SET NULL delete rule. With regards to triggered statements, it includes the number of rows that were inserted, updated, or deleted.
- If an error occurs that prevents deleting all rows matching the search condition and all operations required by existing referential constraints, no changes are made to the table and the error is returned.
- For nicknames, the external server option `iud_app_svpt_enforce` poses an additional limitation. Refer to the Federated documentation for more information.
- For some data sources, the SQLCODE -20190 may be returned on a delete against a nickname because of potential data inconsistency. Refer to the Federated documentation for more information.
- For any deleted row that includes currently linked files through DATALINK columns, the files are unlinked, and will be either restored or deleted, depending on the datalink column definition.

  An error may occur when attempting to delete a DATALINK value if the file server referenced in the value is no longer registered with the database server (SQLSTATE 55022).

  An error may also occur when deleting a row that has a link to a server that is unavailable at the time of deletion (SQLSTATE 57050).

Examples:

Example 1: Delete department (DEPTNO) 'D11' from the DEPARTMENT table.

```
DELETE FROM DEPARTMENT
  WHERE DEPTNO = 'D11'
```

Example 2: Delete all the departments from the DEPARTMENT table (that is, empty the table).

```
DELETE FROM DEPARTMENT
```

Example 3: Delete from the EMPLOYEE table any sales rep or field rep who didn't make a sale in 1995.

```
DELETE FROM EMPLOYEE
  WHERE LASTNAME NOT IN
    (SELECT SALES_PERSON
      FROM SALES
      WHERE YEAR(SALES_DATE)=1995)
      AND JOB IN ('SALESREP','FIELDREP')
```

Example 4: Delete all the duplicate employee rows from the EMPLOYEE table. An employee row is considered to be a duplicate if the last names match. Keep the employee row with the smallest first name in lexical order.

```
DELETE FROM
  (SELECT ROWNUMBER() OVER (PARTITON BY LASTNAME ORDER BY>/ph> FIRSTNME)
    FROM EMPLOYEE) AS E(RN)
    WHERE RN = 1
```

Related reference:

- "Search conditions" in the *SQL Reference, Volume 1*
- "Subselect" on page 904
- "CREATE VIEW" on page 656
- "DECLARE CURSOR statement" in the *SQL Reference, Volume 2*
- "UPDATE" on page 757
- "SQLCA (SQL communications area)" on page 1004

Related samples:

- "dbuse.c -- How to use a database"
- "tbmod.c -- How to modify table data"
- "dbuse.sqc -- How to use a database (C)"
- "tbconstr.sqc -- How to create, use, and drop constraints (C)"
- "tbmod.sqc -- How to modify table data (C)"
- "dbuse.sqC -- How to use a database (C++)"
- "tbconstr.sqC -- How to create, use, and drop constraints (C++)"
- "tbmod.sqC -- How to modify table data (C++)"
- "delet.sqb -- How to delete table data (MF COBOL)"
- "updat.sqb -- How to update, delete and insert table data (MF COBOL)"
- "DbUse.java -- How to use a database (JDBC)"
- "TbConstr.java -- How to create, use and drop constraints (JDBC)"
- "TbMod.java -- How to modify table data (JDBC)"
- "DbUse.sqlj -- How to use a database (SQLj)"
- "TbConstr.sqlj -- How to create, use and drop constraints (SQLj)"
- "TbMod.sqlj -- How to modify table data (SQLj)"

DROP

The DROP statement deletes an object. Any objects that are directly or indirectly dependent on that object are either deleted or made inoperative. Whenever an object is deleted, its description is deleted from the catalog and any packages that reference the object are invalidated.

Invocation:

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization:

The privileges that must be held by the authorization ID of the DROP statement when dropping objects that allow two-part names must include one of the following or an error will result (SQLSTATE 42501):

- SYSADM or DBADM authority
- DROPIN privilege on the schema for the object
- definer of the object as recorded in the DEFINER column of the catalog view for the object

- CONTROL privilege on the object (applicable only to indexes, index specifications, nicknames, packages, tables, and views).
- definer of the user-defined type as recorded in the DEFINER column of the catalog view SYSCAT.DATATYPES (applicable only when dropping a method associated with a user-defined type)

The authorization ID of the DROP statement when dropping a table or view hierarchy must hold one of the above privileges for each of the tables or views in the hierarchy.

The authorization ID of the DROP statement when dropping a schema must have SYSADM or DBADM authority or be the schema owner as recorded in the OWNER column of SYSCAT.SCHEMATA.

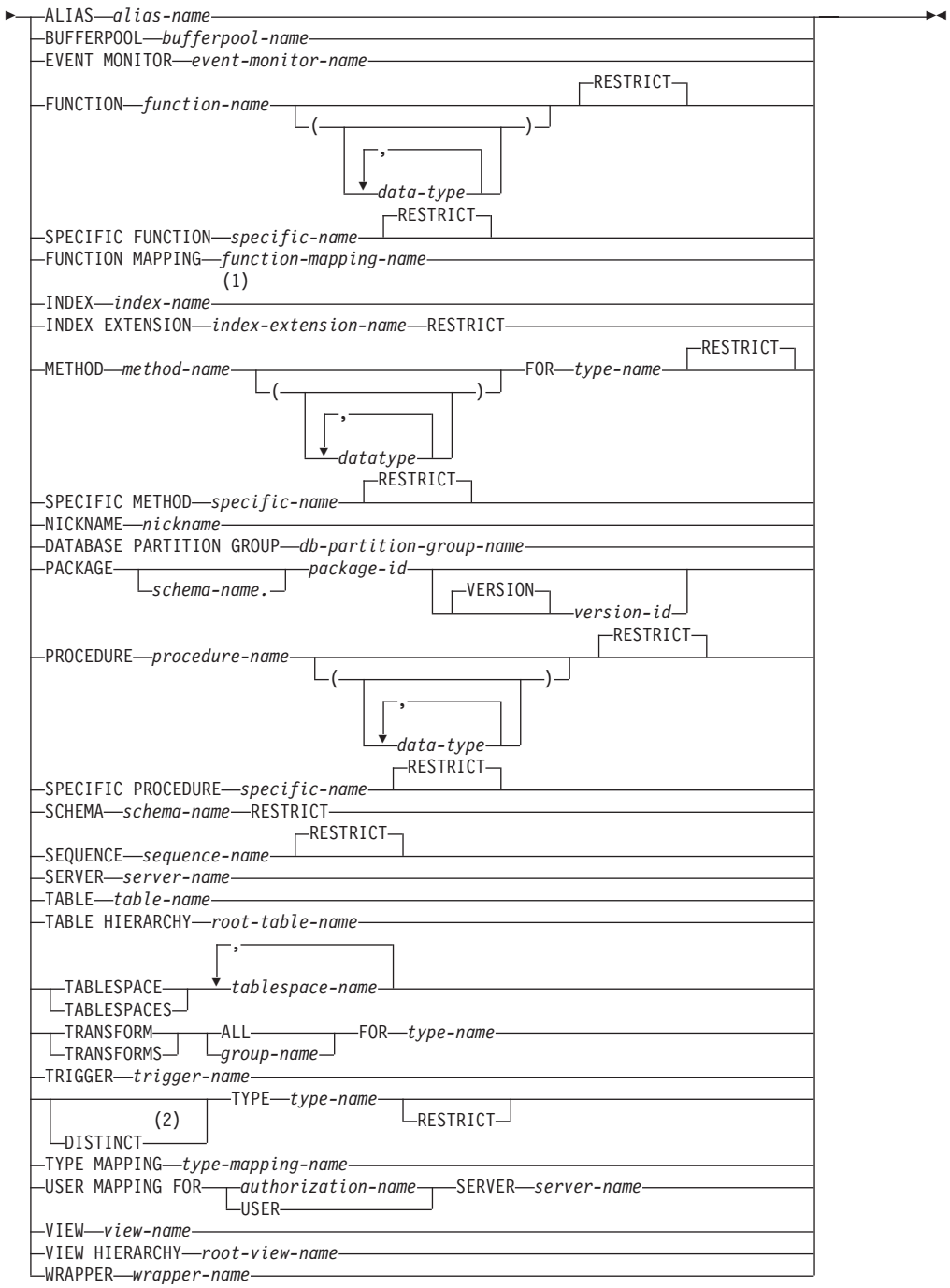
The authorization ID of the DROP statement when dropping a buffer pool, database partition group, or table space must have SYSADM or SYSCTRL authority.

The authorization ID of the DROP statement when dropping an event monitor, server definition, data type mapping, function mapping or a wrapper must have SYSADM or DBADM authority.

The authorization ID of the DROP statement when dropping a user mapping must have SYSADM or DBADM authority, if this authorization ID is different from the federated database authorization name within the mapping. Otherwise, if the authorization ID and the authorization name match, no authorities or privileges are required.

The authorization ID of the DROP statement when dropping a transform must hold SYSADM or DBADM authority, or must be the DEFINER of *type-name*.

Syntax:



Notes:

1 *Index-name* can be the name of either an index or an index specification.

2 DATA can also be used when dropping any user-defined type.

Description:

ALIAS *alias-name*
Identifies the alias that is to be dropped. The *alias-name* must identify an alias that is described in the catalog (SQLSTATE 42704). The specified alias is deleted.

All tables, views, and triggers that reference the alias are made inoperative. (This includes both the table referenced in the ON clause of the CREATE TRIGGER statement, and all tables referenced within the triggered SQL statements.)

BUFFERPOOL *bufferpool-name*
Identifies the buffer pool that is to be dropped. The *bufferpool-name* must identify a buffer pool that is described in the catalog (SQLSTATE 42704). There can be no table spaces assigned to the buffer pool (SQLSTATE 42893). The IBMDEFAULTBP buffer pool cannot be dropped (SQLSTATE 42832). Buffer pool memory is released immediately, to be used by DB2. Disk storage may not be released until the next connection to the database.

EVENT MONITOR *event-monitor-name*
Identifies the event monitor that is to be dropped. The *event-monitor-name* must identify an event monitor that is described in the catalog (SQLSTATE 42704).

If the identified event monitor is ON, an error (SQLSTATE 55034) is returned; otherwise, the event monitor is deleted.

If there are event files in the target path of the event monitor when the event monitor is dropped, the event files are not deleted. However, if a new event monitor that specifies the same target path is created, the event files are deleted.

When dropping WRITE TO TABLE event monitors, table information is removed from the SYSCAT.EVENTTABLES catalog view, but the tables themselves are not dropped.

FUNCTION
Identifies an instance of a user-defined function (either a complete function or a function template) that is to be dropped. The function instance specified must be a user-defined function described in the catalog. Functions implicitly generated by the CREATE DISTINCT TYPE statement cannot be dropped.

There are several different ways available to identify the function instance:

FUNCTION *function-name*
Identifies the particular function, and is valid only if there is exactly one function instance with the *function-name*. The function thus identified may have any number of parameters defined for it. In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified object names. If no function by this name exists in the named or implied schema, an error (SQLSTATE 42704) is raised. If there is more than one specific instance of the function in the named or implied schema, an error (SQLSTATE 42725) is raised.

FUNCTION *function-name* **(***data-type***,...)**
Provides the function signature, which uniquely identifies the function to be dropped. The function selection algorithm is not used.

function-name
Gives the function name of the function to be dropped. In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified object names.

(data-type,...)
Must match the data types that were specified on the CREATE FUNCTION statement in the corresponding position. The number of data types, and the logical concatenation of the data types is used to identify the specific function instance which is to be dropped.

If the *data-type* is unqualified, the type name is resolved by searching the schemas on the SQL path. This also applies to data type names specified for a REFERENCE type.

It is not necessary to specify the length, precision or scale for the parameterized data types. Instead, an empty set of parentheses may be coded to indicate that these attributes are to be ignored when looking for a data type match.

FLOAT() cannot be used (SQLSTATE 42601) since the parameter value indicates different data types (REAL or DOUBLE).

If length, precision, or scale is coded, the value must exactly match that specified in the CREATE FUNCTION statement.

A type of FLOAT(n) does not need to match the defined value for n since 0<n<25 means REAL and 24<n<54 means DOUBLE. Matching occurs based on whether the type is REAL or DOUBLE.

RESTRICT
The RESTRICT keyword enforces the rule that the function is not to be dropped if any of the following dependencies exists:

- Another routine is sourced on the function.
- A view uses the function.
- A trigger uses the function.
- A materialized query table uses the function in its definition.

RESTRICT is the default behavior.

If no function with the specified signature exists in named or implied schema, an error (SQLSTATE 42883) is raised.

SPECIFIC FUNCTION *specific-name*
Identifies the particular user-defined function that is to be dropped, using the specific name either specified or defaulted to at function creation time. In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified object names. The *specific-name* must identify a specific function instance in the named or implied schema; otherwise, an error (SQLSTATE 42704) is raised.

RESTRICT
The RESTRICT keyword enforces the rule that the function is not to be dropped if any of the following dependencies exists:

- Another routine is sourced on the function.
- A view uses the function.
- A trigger uses the function.

RESTRICT is the default behavior.

It is not possible to drop a function that is in the SYSIBM, SYSFUN, or the SYSPROC schema (SQLSTATE 42832).

Other objects can be dependent upon a function. All such dependencies must be removed before the function can be dropped, with the exception of packages which are marked inoperative. An attempt to drop a function with such dependencies will result in an error (SQLSTATE 42893). See 691 for a list of these dependencies.

If the function can be dropped, it is dropped.

Any package dependent on the specific function being dropped is marked as inoperative. Such a package is not implicitly rebound. It must either be rebound by use of the BIND or REBIND command, or it must be re-prepared by use of the PREP command.

FUNCTION MAPPING *function-mapping-name*
Identifies the function mapping that is to be dropped. The *function-mapping-name* must identify a user-defined function mapping that is described in the catalog (SQLSTATE 42704). The function mapping is deleted from the database.

Default function mappings cannot be dropped, but can be disabled by using the CREATE FUNCTION MAPPING statement. Dropping a user-defined function mapping that was created to override a default function mapping reinstates the default function mapping.

Packages having a dependency on a dropped function mapping are invalidated.

INDEX *index-name*
Identifies the index or index specification that is to be dropped. The *index-name* must identify an index or index specification that is described in the catalog (SQLSTATE 42704). It cannot be an index required by the system for a primary key or unique constraint or for a replicated materialized query table (SQLSTATE 42917). The specified index or index specification is deleted.

Packages having a dependency on a dropped index or index specification are invalidated.

INDEX EXTENSION *index-extension-name* **RESTRICT**
Identifies the index extension that is to be dropped. The *index-extension-name* must identify an index extension that is described in the catalog (SQLSTATE 42704). The RESTRICT keyword enforces the rule that no index can be defined that depends on this index extension definition (SQLSTATE 42893).

METHOD
Identifies a method body that is to be dropped. The method body specified must be a method described in the catalog (SQLSTATE 42704). Method bodies that are implicitly generated by the CREATE TYPE statement cannot be dropped.

DROP METHOD deletes the body of a method, but the method specification (signature) remains as a part of the definition of the subject type. After dropping the body of a method, the method specification can be removed from the subject type definition by ALTER TYPE DROP METHOD.

There are several ways available to identify the method body to be dropped:

METHOD *method-name*
Identifies the particular method to be dropped, and is valid only if there is exactly one method instance with name *method-name* and subject type *type-name*. Thus, the method identified may have any number of parameters. If no method by this name exists for the type *type-name*, an

error (SQLSTATE 42704) is raised. If there is more than one specific instance of the method for the named data type, an error (SQLSTATE 42725) is raised.

METHOD *method-name* **(***data-type***,...)**
Provides the method signature, which uniquely identifies the method to be dropped. The method selection algorithm is not used.

method-name
The method name of the method to be dropped for the specified type. The name must be an unqualified identifier.

(*data-type***, ...)**
Must match the data types that were specified in the corresponding positions of the method-specification of the CREATE TYPE or ALTER TYPE statement. The number of data types and the logical concatenation of the data types are used to identify the specific method instance which is to be dropped.

If the data-type is unqualified, the type name is resolved by searching the schemas on the SQL path.

It is not necessary to specify the length, precision or scale for the parameterized data types. Instead, an empty set of parentheses may be coded to indicate that these attributes are to be ignored when looking for a data type match.

FLOAT() cannot be used (SQLSTATE 42601) since the parameter value indicates different data types (REAL or DOUBLE).

However, if length, precision, or scale is coded, the value must exactly match that specified in the CREATE TYPE statement.

A type of FLOAT(n) does not need to match the defined value for n since 0<n<25 means REAL and 24<n<54 means DOUBLE. Matching occurs based on whether the type is REAL or DOUBLE.

If no method with the specified signature exists for the named data type, an error is raised (SQLSTATE 42883).

FOR *type-name*
Names the type for which the specified method is to be dropped. The name must identify a type already described in the catalog (SQLSTATE 42704). In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified type name. In static SQL statements, the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified type names.

RESTRICT
The RESTRICT keyword enforces the rule that the method is not to be dropped if any of the following dependencies exists:

- Another routine is sourced on the method.
- A view uses the method.
- A trigger uses the method.

RESTRICT is the default behavior.

SPECIFIC METHOD *specific-name*
Identifies the particular method that is to be dropped, using a name either specified or defaulted to at CREATE TYPE or ALTER TYPE time. If the specific name is unqualified, the CURRENT SCHEMA special register is used as a qualifier for an unqualified specific name in dynamic SQL. In

static SQL statements the QUALIFIER precompile/bind option implicitly specifies the qualifier for an unqualified specific name. The specific-name must identify a method; otherwise, an error is raised (SQLSTATE 42704).

RESTRICT
The RESTRICT keyword enforces the rule that the method is not to be dropped if any of the following dependencies exists:

- Another routine is sourced on the method.
- A view uses the method.
- A trigger uses the function.

RESTRICT is the default method.

Other objects can be dependent upon a method. All such dependencies must be removed before the method can be dropped, with the exception of packages which will be marked inoperative if the drop is successful. An attempt to drop a method with such dependencies will result in an error (SQLSTATE 42893).

If the method can be dropped, it will be dropped.

Any package dependent on the specific method being dropped is marked as inoperative. Such a package is not implicitly re-bound. Either it must be re-bound by use of the BIND or REBIND command, or it must be re-prepared by use of the PREP command.

If the specific method being dropped overrides another method, all packages dependent on the overridden method — and on methods that override this method in supertypes of the specific method being dropped — are invalidated.

NICKNAME *nickname*
Identifies the nickname that is to be dropped. The nickname must be listed in the catalog (SQLSTATE 42704). The nickname is deleted from the database.

All information about the columns and indexes associated with the nickname is deleted from the catalog. Any materialized query tables that are dependent on the nickname are dropped. Any index specifications that are dependent on the nickname are dropped. Any views that are dependent on the nickname are marked inoperative. Any packages that are dependent on the dropped index specifications or inoperative views are invalidated. The data source table that the nickname references is not affected.

If an SQL function or method is dependent on a nickname, that nickname cannot be dropped (SQLSTATE 42893).

DATABASE PARTITION GROUP *db-partition-group-name*
Identifies the database partition group that is to be dropped. The *db-partition-group-name* parameter must identify a database partition group that is described in the catalog (SQLSTATE 42704). This is a one-part name.

Dropping a database partition group drops all table spaces defined in the database partition group. All existing database objects with dependencies on the tables in the table spaces (such as packages, referential constraints, and so on) are dropped or invalidated (as appropriate), and dependent views and triggers are made inoperative.

System-defined database partition groups cannot be dropped (SQLSTATE 42832).

If a DROP DATABASE PARTITION GROUP statement is issued against a database partition group that is currently undergoing a data redistribution, the

drop database partition group operation fails, and an error is returned (SQLSTATE 55038). However, a partially redistributed database partition group can be dropped. A database partition group can become partially redistributed if a REDISTRIBUTE DATABASE PARTITION GROUP command does not execute to completion. This can happen if it is interrupted by either an error or a FORCE APPLICATION ALL command. (For a partially redistributed database partition group, the REBALANCE_PMAP_ID in the SYSCAT.DBPARTITIONGROUPS catalog is not –1.)

PACKAGE *schema-name.package-id*
Identifies the package that is to be dropped. If a schema name is not specified, the package identifier is implicitly qualified by the default schema. The schema name and package identifier, together with the implicitly or explicitly specified version identifier, must identify a package that is described in the catalog (SQLSTATE 42704). The specified package is deleted. If the package being dropped is the only package identified by *schema-name.package-id* (that is, there are no other versions), all privileges on the package are also deleted.

VERSION *version-id*
Identifies which package version is to be dropped. If a value is not specified, the version defaults to the empty string. If multiple packages with the same package name but different versions exist, only one package version can be dropped in one invocation of the DROP statement. Delimit the version identifier with double quotation marks when it:

- Is generated by the VERSION(AUTO) precompiler option
- Begins with a digit
- Contains lowercase or mixed-case letters

If the statement is invoked from an operating system command prompt, precede each double quotation mark delimiter with a back slash character to ensure that the operating system does not strip the delimiters.

PROCEDURE
Identifies an instance of a stored procedure that is to be dropped. The procedure instance specified must be a stored procedure described in the catalog.

There are several different ways available to identify the procedure instance:

PROCEDURE *procedure-name*
Identifies the particular procedure to be dropped, and is valid only if there is exactly one procedure instance with the *procedure-name* in the schema. The procedure thus identified may have any number of parameters defined for it. If no procedure by this name exists in the named or implied schema, an error (SQLSTATE 42704) is raised. In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified object names. If there is more than one specific instance of the procedure in the named or implied schema, an error (SQLSTATE 42725) is returned.

RESTRICT
The RESTRICT keyword prevents the procedure from being dropped if a trigger definition, an SQL function, or an SQL method contains a CALL statement with the name of the procedure. RESTRICT is the default behavior.

PROCEDURE *procedure-name* **(***data-type***,...)**
Provides the procedure signature, which uniquely identifies the procedure to be dropped. The procedure selection algorithm is not used.

procedure-name
Gives the procedure name of the procedure to be dropped. In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified object names.

(data-type,...)
Must match the data types that were specified on the CREATE PROCEDURE statement in the corresponding position. The number of data types, and the logical concatenation of the data types is used to identify the specific procedure instance which is to be dropped.

If the *data-type* is unqualified, the type name is resolved by searching the schemas on the SQL path. This also applies to data type names specified for a REFERENCE type.

It is not necessary to specify the length, precision or scale for the parameterized data types. Instead, an empty set of parentheses may be coded to indicate that these attributes are to be ignored when looking for a data type match.

FLOAT() cannot be used (SQLSTATE 42601) since the parameter value indicates different data types (REAL or DOUBLE).

However, if length, precision, or scale is coded, the value must exactly match that specified in the CREATE FUNCTION statement.

A type of FLOAT(n) does not need to match the defined value for n since 0<n<25 means REAL and 24<n<54 means DOUBLE. Matching occurs based on whether the type is REAL or DOUBLE.

If no procedure with the specified signature exists in named or implied schema, an error (SQLSTATE 42883) is returned.

SPECIFIC PROCEDURE *specific-name*
Identifies the particular stored procedure that is to be dropped, using the specific name either specified or defaulted to at procedure creation time. In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified object names. The *specific-name* must identify a specific procedure instance in the named or implied schema; otherwise, an error (SQLSTATE 42704) is raised.

RESTRICT
The RESTRICT keyword prevents the procedure from being dropped if a trigger definition, an SQL function, or an SQL method contains a CALL statement with the name of the procedure. RESTRICT is the default behavior.

It is not possible to drop a procedure that is in the SYSIBM, SYSFUN, or the SYSPROC schema (SQLSTATE 42832).

SCHEMA *schema-name* **RESTRICT**
Identifies the particular schema to be dropped. The *schema-name* must identify a schema that is described in the catalog (SQLSTATE 42704). The RESTRICT

keyword enforces the rule that no objects can be defined in the specified schema for the schema to be deleted from the database (SQLSTATE 42893).

SEQUENCE *sequence-name*
Identifies the particular sequence that is to be dropped. The *sequence-name*, along with the implicit or explicit schema name, must identify an existing sequence at the current server. If no sequence by this name exists in the explicitly or implicitly specified schema, an error (SQLSTATE 42704) is raised.

The RESTRICT option, which is the default, prevents the sequence from being dropped if any of the following dependencies exist:

- A trigger exists such that a NEXT VALUE or PREVIOUS VALUE expression in the trigger specifies the sequence (SQLSTATE 42893).
- An SQL function or an SQL method exists such that a NEXT VALUE expression in the routine body specifies the sequence (SQLSTATE 42893).

SERVER *server-name*
Identifies the data source whose definition is to be dropped from the catalog. The *server-name* must identify a data source that is described in the catalog (SQLSTATE 42704). The definition of the data source is deleted.

All nicknames for tables and views residing at the data source are dropped. Any index specifications dependent on these nicknames are dropped. Any user-defined function mappings, user-defined type mappings, and user mappings that are dependent on the dropped server definition are also dropped. All packages dependent on the dropped server definition, function mappings, nicknames, and index specifications are invalidated.

TABLE *table-name*
Identifies the base table, declared temporary table, or nickname that is to be dropped. The *table-name* must identify a table that is described in the catalog or, if it is a declared temporary table, the *table-name* must be qualified by the schema name SESSION and exist in the application (SQLSTATE 42704). The subtables of a typed table are dependent on their supertables. All subtables must be dropped before a supertable can be dropped (SQLSTATE 42893). The specified table is deleted from the database.

All indexes, primary keys, foreign keys, check constraints, materialized query tables, and staging tables referencing the table are dropped. All views and triggers that reference the table are made inoperative. (This includes both the table referenced in the ON clause of the CREATE TRIGGER statement, and all tables referenced within the triggered SQL statements.) All packages depending on any object dropped or marked inoperative will be invalidated. This includes packages dependent on any supertables above the subtable in the hierarchy. Any reference columns for which the dropped table is defined as the scope of the reference become unscoped.

Packages are not dependent on declared temporary tables, and therefore are not invalidated when such a table is dropped.

All files that are linked through any DATALINK columns are unlinked. The unlink operation is performed asynchronously so the files may not be immediately available for other operations.

In a federated system, a remote table that was created using transparent DDL can be dropped. Dropping a remote table also drops the nickname associated with that table, and invalidates any packages that are dependent on that nickname.

When a subtable is dropped from a table hierarchy, the columns associated with the subtable are no longer accessible although they continue to be considered with respect to limits on the number of columns and size of the row. Dropping a subtable has the effect of deleting all the rows of the subtable from the supertables. This may result in activation of triggers or referential integrity constraints defined on the supertables.

When a declared temporary table is dropped, and its creation preceded the active unit of work or savepoint, then the table will be functionally dropped and the application will not be able to access the table. However, the table will still reserve some space in its table space and will prevent that USER TEMPORARY table space from being dropped or the database partition group of the USER TEMPORARY table space from being redistributed until the unit of work is committed or savepoint is ended. Dropping a declared temporary table causes the data in the table to be destroyed, regardless of whether DROP is committed or rolled back.

A table cannot be dropped if it has the RESTRICT ON DROP attribute.

TABLE HIERARCHY *root-table-name*
Identifies the typed table hierarchy that is to be dropped. The *root-table-name* must identify a typed table that is the root table in the typed table hierarchy (SQLSTATE 428DR). The typed table identified by *root-table-name* and all of its subtables are deleted from the database.

All indexes, materialized query tables, staging tables, primary keys, foreign keys, and check constraints referencing the dropped tables are dropped. All views and triggers that reference the dropped tables are made inoperative. All packages depending on any object dropped or marked inoperative will be invalidated. Any reference columns for which one of the dropped tables is defined as the scope of the reference become unscoped.

All files that are linked through any DATALINK columns are unlinked. The unlink operation is performed asynchronously so the files may not be immediately available for other operations.

Unlike dropping a single subtable, dropping the table hierarchy does not result in the activation of delete triggers of any tables in the hierarchy nor does it log the deleted rows.

TABLESPACE or **TABLESPACES** *tablespace-name*
Identifies the table spaces that are to be dropped. *tablespace-name* must identify a table space that is described in the catalog (SQLSTATE 42704). This is a one-part name.

The table spaces will not be dropped (SQLSTATE 55024) if there is any table that stores at least one of its parts in a table space being dropped, and has one or more of its parts in another table space that is not being dropped (these tables would need to be dropped first), or if any table that resides in the table space has the RESTRICT ON DROP attribute. System table spaces cannot be dropped (SQLSTATE 42832). A SYSTEM TEMPORARY table space cannot be dropped (SQLSTATE 55026) if it is the only temporary table space that exists in the database. A USER TEMPORARY table space cannot be dropped if there is a declared temporary table created in it (SQLSTATE 55039). Even if a declared temporary table has been dropped, the USER TEMPORARY table space will still be considered to be in use until the unit of work containing the DROP TABLE has been committed.

Dropping a table space drops all objects defined in the table space. All existing database objects with dependencies on the table space, such as packages,

referential constraints, and so on, are dropped or invalidated (as appropriate), and dependent views and triggers are made inoperative.

Containers created by a user are not deleted. Any directories in the path of the container name that were created by the database manager on CREATE TABLESPACE are deleted. All containers that are below the database directory are deleted. When DROP TABLESPACE is committed, the DMS file containers or SMS containers for the specified table space are deleted, if possible. If the containers cannot be deleted (because they are being kept open by another agent, for example), the files are truncated to zero-length. After all connections are terminated, or the DEACTIVATE DATABASE command is issued, these zero-length files are deleted.

TRANSFORM ALL FOR *type-name*
Indicates that all transforms groups defined for the user-defined data type *type-name* are to be dropped. The transform functions referenced in these groups are not dropped. In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements, the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified object names. The *type-name* must identify a user-defined type described in the catalog (SQLSTATE 42704).

If there are not transforms defined for *type-name*, an error is raised (SQLSTATE 42740).

DROP TRANSFORM is the inverse of CREATE TRANSFORM. It causes the transform functions associated with certain groups, for a given datatype, to become undefined. The functions formerly associated with these groups still exist and can still be called explicitly, but they no longer have the transform property, and are no longer invoked implicitly for exchanging values with the host language environment.

The transform group is not dropped if there is a user-defined function (or method) written in a language other than SQL that has a dependency on one of the group's transform functions defined for the user-defined type *type-name* (SQLSTATE 42893). Such a function has a dependency on the transform function associated with the referenced transform group defined for type *type-name*. Packages that depend on a transform function associated with the named transform group are marked inoperative.

TRANSFORMS *group-name* **FOR** *type-name*
Indicates that the specified transform group for the user-defined data type *type-name* is to be dropped. The transform functions referenced in this group are not dropped. In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements, the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified object names. The *type-name* must identify a user-defined type described in the catalog (SQLSTATE 42704), and the *group-name* must identify an existing transform group for *type-name*.

TRIGGER *trigger-name*
Identifies the trigger that is to be dropped. The *trigger-name* must identify a trigger that is described in the catalog (SQLSTATE 42704). The specified trigger is deleted.

Dropping triggers causes certain packages to be marked invalid.

If *trigger-name* specifies an INSTEAD OF trigger on a view, another trigger may depend on that trigger through an update against the view.

TYPE *type-name*
Identifies the user-defined type to be dropped. In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified object names. For a structured type, the associated reference type is also dropped. The *type-name* must identify a user-defined type described in the catalog. If DISTINCT is specified, then the *type-name* must identify a distinct type described in the catalog.

RESTRICT
The type is not dropped (SQLSTATE 42893) if any of the following is true:

- The type is used as the type of a column of a table or view.
- The type has a subtype.
- The type is a structured type used as the data type of a typed table or a typed view.
- The type is an attribute of another structured type.
- There exists a column of a table whose type might contain an instance of *type-name*. This can occur if *type-name* is the type of the column or is used elsewhere in the column's associated type hierarchy. More formally, for any type T, T cannot be dropped if there exists a column of a table whose type directly or indirectly uses *type-name*.
- The type is the target type of a reference-type column of a table or view, or a reference-type attribute of another structured type.
- The type, or a reference to the type, is a parameter type or a return value type of a function or method.
- The type, or a reference to the type, is used in the body of an SQL function or method, but it is not a parameter type or a return value type.
- The type is used in a check constraint, trigger, view definition, or index extension.

If RESTRICT is not specified, the behavior is the same as RESTRICT, except for functions and methods that use the type.

Functions that use the type: If the user-defined type can be dropped, then for every function, F (with specific name SF), that has parameters or a return value of the type being dropped or a reference to the type being dropped, the following DROP FUNCTION statement is effectively executed:

```
DROP SPECIFIC FUNCTION SF
```

It is possible that this statement also would cascade to drop dependent functions. If all of these functions are also in the list to be dropped because of a dependency on the user-defined type, the drop of the user-defined type will succeed (otherwise it fails with SQLSTATE 42893).

Methods that use the type: If the user-defined type can be dropped, then for every method, M of type T1 (with specific name SM), that has parameters or a return value of the type being dropped or a reference to the type being dropped, the following statements are effectively executed:

```
DROP SPECIFIC METHOD SM
ALTER TYPE T1 DROP SPECIFIC METHOD SM
```

The existence of objects that are dependent on these methods may cause the DROP TYPE operation to fail.

All packages that are dependent on methods defined in supertypes of the type being dropped, and that are eligible for overriding, are invalidated.

TYPE MAPPING *type-mapping-name*
Identifies the user-defined data type mapping to be dropped. The *type-mapping-name* must identify a data type mapping that is described in the catalog (SQLSTATE 42704). The data type mapping is deleted from the database.

No additional objects are dropped.

USER MAPPING FOR *authorization-name* | **USER SERVER** *server-name*
Identifies the user mapping to be dropped. This mapping associates an authorization name that is used to access the federated database with an authorization name that is used to access a data source. The first of these two authorization names is either identified by the *authorization-name* or referenced by the special register USER. The *server-name* identifies the data source that the second authorization name is used to access.

The *authorization-name* must be listed in the catalog (SQLSTATE 42704). The *server-name* must identify a data source that is described in the catalog (SQLSTATE 42704). The user mapping is deleted.

No additional objects are dropped.

VIEW *view-name*
Identifies the view that is to be dropped. The *view-name* must identify a view that is described in the catalog (SQLSTATE 42704). The subviews of a typed view are dependent on their superviews. All subviews must be dropped before a superview can be dropped (SQLSTATE 42893).

The specified view is deleted. The definition of any view or trigger that is directly or indirectly dependent on that view is marked inoperative. Any materialized query table or staging table that is dependent on any view that is marked inoperative is dropped. Any packages dependent on a view that is dropped or marked inoperative will be invalidated. This includes packages dependent on any superviews above the subview in the hierarchy. Any reference columns for which the dropped view is defined as the scope of the reference become unscoped.

VIEW HIERARCHY *root-view-name*
Identifies the typed view hierarchy that is to be dropped. The *root-view-name* must identify a typed view that is the root view in the typed view hierarchy (SQLSTATE 428DR). The typed view identified by *root-view-name* and all of its subviews are deleted from the database.

The definition of any view or trigger that is directly or indirectly dependent on any of the dropped views is marked inoperative. Any packages dependent on any view or trigger that is dropped or marked inoperative will be invalidated. Any reference columns for which a dropped view or view marked inoperative is defined as the scope of the reference become unscoped.

WRAPPER *wrapper-name*
Identifies the wrapper to be dropped. The *wrapper-name* must identify a wrapper that is described in the catalog (SQLSTATE 42704). The wrapper is deleted.

All server definitions, user-defined function mappings, and user-defined data type mappings that are dependent on the wrapper are dropped. All user-defined function mappings, nicknames, user-defined data type mappings, and user mappings that are dependent on the dropped server definitions are

also dropped. Any index specifications dependent on the dropped nicknames are dropped, and any views dependent on these nicknames are marked inoperative. All packages dependent on the dropped objects and inoperative views are invalidated.

Rules:

Dependencies: Table 63 on page 692 shows the dependencies that objects have on each other. Not all dependencies are explicitly recorded in the catalog. For example, there is no record of the constraints on which a package has dependencies. Four different types of dependencies are shown:

R Restrict semantics. The underlying object cannot be dropped as long as the object that depends on it exists.

C Cascade semantics. Dropping the underlying object causes the object that depends on it (the depending object) to be dropped as well. However, if the depending object cannot be dropped because it has a Restrict dependency on some other object, the drop of the underlying object will fail.

X Inoperative semantics. Dropping the underlying object causes the object that depends on it to become inoperative. It remains inoperative until a user takes some explicit action.

A Automatic Invalidation/Revalidation semantics. Dropping the underlying object causes the object that depends on it to become invalid. The database manager attempts to revalidate the invalid object.

A package used by a function or a method, or by a procedure that is called directly or indirectly from a function or method, will only be automatically revalidated if the routine is defined as MODIFIES SQL DATA. If the routine is not MODIFIES SQL DATA, an error is returned (SQLSTATE 56098).

Some DROP statement parameters and objects are not shown in Table 63 on page 692 because they would result in blank rows or columns:

- EVENT MONITOR, PACKAGE, PROCEDURE, SCHEMA, TYPE MAPPING, and USER MAPPING DROP statements do not have object dependencies.
- Alias, bufferpool, partitioning key, privilege, and procedure object types do not have DROP statement dependencies.
- A DROP SERVER, DROP FUNCTION MAPPING, or DROP TYPE MAPPING statement in a given unit of work (UOW) cannot be processed under either of the following conditions:
  - The statement references a single data source, and the UOW already includes a SELECT statement that references a nickname for a table or view within this data source (SQLSTATE 55006).
  - The statement references a category of data sources (for example, all data sources of a specific type and version), and the UOW already includes a SELECT statement that references a nickname for a table or view within one of these data sources (SQLSTATE 55006).

Table 63. Dependencies

Object Type → / Statement ↓	CONSTRAINT	FUNCTION	FUNCMAPPING	INDEX	INDEXTENSION	METHOD	NICKNAME	NODEGROUP	PACKAGE[31]	SERVER	TABLE	TABLESPACE	TRIGGER	TYPE	TYPEMAPPING	USERMAPPING	VIEW
ALTER FUNCTION	-	-	-	-	-	-	-	-	A	-	-	-	-	-	-	-	-
ALTER METHOD	-	-	-	-	-	-	-	-	A	-	-	-	-	-	-	-	-
ALTER NICKNAME, altering the local name or the local type	R[33]	R	-	-	-	R	-	-	A	-	R	-	-	-	-	-	R
ALTER NICKNAME, altering a column option or a nickname option	-	-	-	-	-	-	-	-	A	-	R	-	-	-	-	-	-
ALTER NICKNAME, adding, altering, or dropping a constraint	-	-	-	-	-	-	-	-	A	-	-	-	-	-	-	-	-
ALTER PROCEDURE	-	-	-	-	-	-	-	-	A	-	-	-	-	-	-	-	-
ALTER SERVER	-	-	-	-	-	-	-	-	A	-	-	-	-	-	-	-	-
ALTER TABLE DROP CONSTRAINT	C	-	-	-	-	-	-	-	A[1]	-	-	-	-	-	-	-	-
ALTER TABLE DROP PARTITIONING KEY	-	-	-	-	-	-	-	R[20]	A[1]	-	-	-	-	-	-	-	-
ALTER TYPE ADD ATTRIBUTE	-	-	-	-	R	-	-	-	A[23]	-	R[24]	-	-	-	-	-	R[14]
ALTER TYPE ALTER METHOD	-	-	-	-	-	-	-	-	A	-	-	-	-	-	-	-	-
ALTER TYPE DROP ATTRIBUTE	-	-	-	-	R	-	-	-	A[23]	-	R[24]	-	-	-	-	-	R[14]
ALTER TYPE ADD METHOD	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
ALTER TYPE DROP METHOD	-	-	-	-	-	R[27]	-	-	-	-	-	-	-	-	-	-	-
CREATE METHOD	-	-	-	-	-	-	-	-	A[28]	-	-	-	-	-	-	-	-
CREATE TYPE	-	-	-	-	-	-	-	-	A[29]	-	-	-	-	-	-	-	-
DROP ALIAS	-	R	-	-	-	-	-	-	A[3]	-	R[3]	-	X[3]	-	-	-	X[3]
DROP BUFFERPOOL	-	-	-	-	-	-	-	-	-	-	-	R	-	-	-	-	-
DROP DATABASE PARTITION GROUP	-	-	-	-	-	-	-	-	-	-	-	C	-	-	-	-	-
DROP FUNCTION	R	R[7]	R	-	R	R[7]	-	-	X	-	R	-	R	-	-	-	R
DROP FUNCTION MAPPING	-	-	-	-	-	-	-	-	A	-	-	-	-	-	-	-	-
DROP INDEX	R	-	-	-	-	-	-	-	A	-	-	-	-	-	-	-	R[17]
DROP INDEX EXTENSION	-	R	-	R	-	-	-	-	-	-	-	-	-	-	-	-	-
DROP METHOD	R	R[7]	R	-	R	R	-	-	X/A[30]	-	R	-	R	-	-	-	R
DROP NICKNAME	-	R	-	C	-	R	-	-	A	-	C[11]	-	-	-	-	-	X[16]

Table 63. Dependencies (continued)

Object Type → / Statement ↓	CONSTRAINT	FUNCTION	FUNCMAPPING	INDEX	INDEXTENSION	METHOD	NICKNAME	NODEGROUP	PACKAGE[31]	SERVER	TABLE	TABLESPACE	TRIGGER	TYPE	TYPEMAPPING	USERMAPPING	VIEW
DROP PROCEDURE	-	R[7]	-	-	-	R[7]	-	-	A	-	-	-	R	-	-	-	-
DROP SEQUENCE	-	R	-	-	-	R	-	-	A	-	-	-	R	-	-	-	-
DROP SERVER	-	C[21]	C[19]	-	-	-	C	-	A	-	-	-	-	-	C[19]	C	-
DROP TABLE[32]	C	R	-	C	-	-	-	-	A[9]	-	RC[11]	-	X[16]	-	-	-	X[16]
DROP TABLE HIERARCHY	C	R	-	C	-	-	-	-	A[9]	-	RC[11]	-	X[16]	-	-	-	X[16]
DROP TABLESPACE	-	-	-	C[6]	-	-	-	-	-	-	CR[6]	-	-	-	-	-	-
DROP TRANSFORM	-	R	-	-	-	-	-	-	X	-	-	-	-	-	-	-	-
DROP TRIGGER	-	-	-	-	-	-	-	-	A[1]	-	-	-	X[26]	-	-	-	-
DROP TYPE	R[13]	R[5]	-	-	R	-	-	-	A[12]	-	R[18]	-	R[13]	R[4]	-	-	R[14]
DROP VIEW	-	R	-	-	-	-	-	-	A[2]	-	-	-	X[16]	-	-	-	X[15]
DROP VIEW HIERARCHY	-	R	-	-	-	-	-	-	A[2]	-	-	-	X[16]	-	-	-	X[16]
DROP WRAPPER	-	-	C	-	-	-	-	-	-	C	-	-	-	-	C	-	-
REVOKE a privilege[10]	-	CR[25]	-	-	-	CR[25]	-	-	A[1]	-	CX[8]	-	X	-	-	-	X[8]

1 This dependency is implicit in depending on a table with these constraints, triggers, or a partitioning key.

2 If a package has an INSERT, UPDATE, or DELETE statement acting upon a view, then the package has an insert, update or delete usage on the underlying base table of the view. In the case of UPDATE, the package has an update usage on each column of the underlying base table that is modified by the UPDATE.

If a package has a statement acting on a typed view, creating or dropping any view in the same view hierarchy will invalidate the package.

3 If a package, materialized query table, staging table, view, or trigger uses an alias, it becomes dependent both on the alias and the object that the alias references. If the alias is in a chain, a dependency is created on each alias in the chain.

Aliases themselves are not dependent on anything. It is possible for an alias to be defined on an object that does not exist.

4 A user-defined type T can depend on another user-defined type B, if T:

- names B as the data type of an attribute
- has an attribute of REF(B)
- has B as a supertype.

5 Dropping a data type cascades to drop the functions and methods that use that data type as a parameter or a result type, and methods defined on the data type. Dropping of these functions and methods will not be prevented

by the fact that they depend on each other. However, for functions or methods using the datatype within their bodies, restrict semantics apply.

6 Dropping a table space or a list of table spaces causes all the tables that are completely contained within the given table space or list to be dropped. However, if a table spans table spaces (indexes or long columns in different table spaces) and those table spaces are not in the list being dropped then the table space(s) cannot be dropped as long as the table exists.

7 A function can depend on another specific function if the depending function names the base function in a SOURCE clause. A function or method can also depend on another specific function or method if the depending routine is written in SQL and uses the base routine in its body. An external method, or an external function with a structured type parameter or returns type will also depend on one or more transform functions.

8 Only loss of SELECT privilege will cause a materialized query table to be dropped or a view to become inoperative. If the view that is made inoperative is included in a typed view hierarchy, all of its subviews also become inoperative.

9 If a package has an INSERT, UPDATE, or DELETE statement acting on table T, then the package has an insert, update or delete usage on T. In the case of UPDATE, the package has an update usage on each column of T that is modified by the UPDATE.

If a package has a statement acting on a typed table, creating or dropping any table in the same table hierarchy will invalidate the package.

10 Dependencies do not exist at the column level because privileges on columns cannot be revoked individually.

If a package, trigger or view includes the use of OUTER(*Z*) in the FROM clause, there is a dependency on the SELECT privilege on every subtable or subview of *Z*. Similarly, if a package, trigger, or view includes the use of DEREF(*Y*) where *Y* is a reference type with a target table or view *Z*, there is a dependency on the SELECT privilege on every subtable or subview of *Z*.

11 A materialized query table is dependent on the underlying tables or nicknames specified in the fullselect of the table definition.

Cascade semantics apply to dependent materialized query tables.

A subtable is dependent on its supertables up to the root table. A supertable cannot be dropped until all of its subtables are dropped.

12 A package can depend on structured types as a result of using the TYPE predicate or the subtype-treatment expression (TREAT *expression* AS *data-type*). The package has a dependency on the subtypes of each structured type specified in the right side of the TYPE predicate, or the right side of the TREAT expression. Dropping or creating a structured type that alters the subtypes on which the package is dependent causes invalidation.

All packages that are dependent on methods defined in supertypes of the type being dropped, and that are eligible for overriding, are invalidated.

13 A check constraint or trigger is dependent on a type if the type is used

anywhere in the constraint or trigger. There is no dependency on the subtypes of a structured type used in a TYPE predicate within a check constraint or trigger.

14 A view is dependent on a type if the type is used anywhere in the view definition (this includes the type of typed view). There is no dependency on the subtypes of a structured type used in a TYPE predicate within a view definition.

15 A subview is dependent on its superview up to the root view. A superview cannot be dropped until all its subviews are dropped. Refer to [16] for additional view dependencies.

16 A trigger or view is also dependent on the target table or target view of a dereference operation or DEREF function. A trigger or view with a FROM clause that includes OUTER(Z) is dependent on all the subtables or subviews of *Z* that existed at the time the trigger or view was created.

17 A typed view can depend on the existence of a unique index to ensure the uniqueness of the object identifier column.

18 A table may depend on a user defined data type (distinct or structured) because the type is:

- used as the type of a column
- used as the type of the table
- used as an attribute of the type of the table
- used as the target type of a reference type that is the type of a column of the table or an attribute of the type of the table
- directly or indirectly used by a type that is the column of the table.

19 Dropping a server cascades to drop the function mappings and type mappings created for that named server.

20 If the partitioning key is defined on a table in a multiple partition database partition group, the partitioning key is required.

21 If a dependent OLE DB table function has ″R″ dependent objects (see DROP FUNCTION), then the server cannot be dropped.

22 An SQL function or method can depend on the objects referenced by its body.

23 When an attribute A of type TA of *type-name* T is dropped, the following DROP statements are effectively executed:

```
Mutator method: DROP METHOD A (TA) FOR T
Observer method: DROP METHOD A () FOR T
ALTER TYPE T
  DROP METHOD A(TA)
  DROP METHOD A()
```

24 A table may depend on an attribute of a user-defined structured data type in the following cases:

1. The table is a typed table that is based on *type-name* or any of its subtypes.
2. The table has an existing column of a type that directly or indirectly refers to *type-name*.

25 A REVOKE of SELECT privilege on a table or view that is used in the body of an SQL function or method body causes an attempt to drop the function or method body, if the function or method body defined no longer has the SELECT privilege. If such a function or method body is used in a

view, trigger, function, or method body, it cannot be dropped, and the REVOKE is restricted as a result. Otherwise, the REVOKE cascades and drops such functions.

26 A trigger depends on an INSTEAD OF trigger when it modifies the view on which the INSTEAD OF trigger is defined, and the INSTEAD OF trigger fires.

27 A method declaration of an original method that is overridden by other methods cannot be dropped.(SQLSTATE -2).

28 If the method of the method body being created is declared to override another method, all packages dependent on the overridden method, and on methods that override this method in supertypes of the method being created, are invalidated.

29 When a new subtype of an existing type is created, all packages dependent on methods that are defined in supertypes of the type being created, and that are eligible for overriding (for example, no mutators or observers), are invalidated.

30 If the specific method of the method body being dropped is declared to override another method, all packages dependent on the overridden method, and on methods that override this method in supertypes of the specific method being dropped, are invalidated.

31 Cached dynamic SQL has the same semantics as packages.

32 When a remote base table is dropped using the DROP TABLE statement, both the nickname and the remote base table are dropped.

33 A primary key or unique keys that are not referenced by a foreign key do not restrict the altering of a nickname local name or local type.

Notes:

- *Compatibilities*
  - For compatibility with previous versions of DB2:
    - NODEGROUP can be specified in place of DATABASE PARTITION GROUP
  - For compatibility with DB2 UDB for OS/390 and z/OS:
    - SYNONYM can be specified in place of ALIAS
    - PROGRAM can be specified in place of PACKAGE
- It is valid to drop a user-defined function while it is in use. Also, a cursor can be open over a statement which contains a reference to a user-defined function, and while this cursor is open the function can be dropped without causing the cursor fetches to fail.
- If a package which depends on a user-defined function is executing, it is not possible for another authorization ID to drop the function until the package completes its current unit of work. At that point, the function is dropped and the package becomes inoperative. The next request for this package results in an error indicating that the package must be explicitly rebound.
- The removal of a function body (this is very different from dropping the function) can occur while an application which needs the function body is executing. This may or may not cause the statement to fail, depending on whether the function body still needs to be loaded into storage by the database manager on behalf of the statement.

- For any dropped table that includes currently linked files through DATALINK columns, the files are unlinked, and will be either restored or deleted, depending on the datalink column definition.
- If a table containing a DATALINK column is dropped while any DB2 Data Links Managers configured to the database are unavailable, either through DROP TABLE or DROP TABLESPACE, then the operation will fail (SQLSTATE 57050).
- In addition to the dependencies recorded for any explicitly specified UDF, the following dependencies are recorded when transforms are implicitly required:
  1. When the structured type parameter or result of a function or method requires a transform, a dependency is recorded for the function or method on the required TO SQL or FROM SQL transform function.
  2. When an SQL statement included in a package requires a transform function, a dependency is recorded for the package on the designated TO SQL or FROM SQL transform function.

  Since the above describes the only circumstances under which dependencies are recorded due to implicit invocation of transforms, no objects other than functions, methods, or packages can have a dependency on implicitly invoked transform functions. On the other hand, explicit calls to transform functions (in views and triggers, for example) do result in the usual dependencies of these other types of objects on transform functions. As a result, a DROP TRANSFORM statement may also fail due to these ″explicit″ type dependencies of objects on the transform(s) being dropped (SQLSTATE 42893).
- Since the dependency catalogs do not distinguish between depending on a function as a transform versus depending on a function by explicit function call, it is suggested that explicit calls to transform functions are not written. In such an instance, the transform property on the function cannot be dropped, or packages will be marked inoperative, simply because they contain explicit invocations in an SQL expression.
- System created sequences for IDENTITY columns cannot be dropped using the DROP SEQUENCE statement.
- When a sequence is dropped, all privileges on the sequence are also dropped and any packages that refer to the sequence are invalidated.
- For relational nicknames, the DROP NICKNAME statement within a given unit of work (UOW) cannot be processed under either of the following conditions (SQLSTATE 55007):
  - A nickname referenced in this statement has a cursor open on it in the same UOW
  - Either an INSERT, DELETE, or UPDATE statement is already issued in the same UOW against the nickname that is referenced in this statement
- For non-relational nicknames, the DROP NICKNAME statement within a given unit of work (UOW) cannot be processed under any of the following conditions (SQLSTATE 55007):
  - A nickname referenced in this statement has a cursor open on it in the same UOW
  - A nickname referenced in this statement is already referenced by a SELECT statement in the same UOW
  - Either an INSERT, DELETE, or UPDATE statement has already been issued in the same UOW against the nickname that is referenced in this statement
- A DROP SERVER statement (SQLSTATE 55006), or a DROP FUNCTION MAPPING or DROP TYPE MAPPING statement (SQLSTATE 55007) within a given unit of work (UOW) cannot be processed under either of the following conditions:

- The statement references a single data source, and the UOW already includes one of the following:
  - A SELECT statement that references a nickname for a table or view within this data source
  - An open cursor on a nickname for a table or view within this data source
  - Either an INSERT, DELETE, or UPDATE statement issued against a nickname for a table or view within this data source
- The statement references a category of data sources (for example, all data sources of a specific type and version), and the UOW already includes one of the following:
  - A SELECT statement that references a nickname for a table or view within one of these data sources
  - An open cursor on a nickname for a table or view within one of these data sources
  - Either an INSERT, DELETE, or UPDATE statement issued against a nickname for a table or view within one of these data sources

Examples:

Example 1: Drop table TDEPT.

```
DROP TABLE TDEPT
```

Example 2: Drop the view VDEPT.

```
DROP VIEW VDEPT
```

Example 3: The authorization ID HEDGES attempts to drop an alias.

```
DROP ALIAS A1
```

The alias HEDGES.A1 is removed from the catalogs.

Example 4: Hedges attempts to drop an alias, but specifies T1 as the alias-name, where T1 is the name of an existing table (not the name of an alias).

```
DROP ALIAS T1
```

This statement fails (SQLSTATE 42809).

Example 5:

Drop the BUSINESS_OPS database partition group. To drop the database partition group, the two table spaces (ACCOUNTING and PLANS) in the database partition group must first be dropped.

```
DROP TABLESPACE ACCOUNTING
DROP TABLESPACE PLANS
DROP DATABASE PARTITION GROUP BUSINESS_OPS
```

Example 6: Pellow wants to drop the CENTRE function, which he created in his PELLOW schema, using the signature to identify the function instance to be dropped.

```
DROP FUNCTION CENTRE (INT,FLOAT)
```

Example 7: McBride wants to drop the FOCUS92 function, which she created in the PELLOW schema, using the specific name to identify the function instance to be dropped.

```
DROP SPECIFIC FUNCTION PELLOW.FOCUS92
```

Example 8: Drop the function ATOMIC_WEIGHT from the CHEM schema, where it is known that there is only one function with that name.

```
DROP FUNCTION CHEM.ATOMIC_WEIGHT
```

Example 9: Drop the trigger SALARY_BONUS, which caused employees under a specified condition to receive a bonus to their salary.

```
DROP TRIGGER SALARY_BONUS
```

Example 10: Drop the distinct data type named shoesize, if it is not currently in use.

```
DROP DISTINCT TYPE SHOESIZE
```

Example 11: Drop the SMITHPAY event monitor.

```
DROP EVENT MONITOR SMITHPAY
```

Example 12: Drop the schema from Example 2 under CREATE SCHEMA using RESTRICT. Notice that the table called PART must be dropped first.

```
  DROP TABLE PART
DROP SCHEMA INVENTRY RESTRICT
```

Example 13: Macdonald wants to drop the DESTROY procedure, which he created in the EIGLER schema, using the specific name to identify the procedure instance to be dropped.

```
DROP SPECIFIC PROCEDURE  EIGLER.DESTROY
```

Example 14: Drop the procedure OSMOSIS from the BIOLOGY schema, where it is known that there is only one procedure with that name.

```
DROP PROCEDURE BIOLOGY.OSMOSIS
```

Example 15: User SHAWN used one authorization ID to access the federated database and another to access the database at an Oracle data source called ORACLE1. A mapping was created between the two authorizations, but SHAWN no longer needs to access the data source. Drop the mapping.

```
DROP USER MAPPING FOR  SHAWN SERVER ORACLE1
```

Example 16: An index of a data source table that a nickname references has been deleted. Drop the index specification that was created to let the optimizer know about this index.

```
DROP INDEX INDEXSPEC
```

Example 17: Drop the MYSTRUCT1 transform group.

```
DROP TRANSFORM MYSTRUCT1 FOR POLYGON
```

Example 18: Drop the method BONUS for the EMP data type in the PERSONNEL schema.

```
DROP METHOD BONUS (SALARY DECIMAL(10,2)) FOR PERSONNEL.EMP
```

Example 19: Drop the sequence ORG_SEQ, with restrictions.

```
DROP SEQUENCE ORG_SEQ
```

Example 20: A remote table EMPLOYEE was created in a federated system using transparent DDL. Access to the table is no longer needed. Drop the remote table EMPLOYEE.

```
DROP TABLE EMPLOYEE
```

Example 21: Drop the function mapping BONUS_CALC and reinstate the default function mapping (if one exists).

```
DROP FUNCTION MAPPING BONUS_CALC
```

Related tasks:

- "Disabling a default function mapping" in the *Federated Systems Guide*
- "Dropping a user-defined function mapping" in the *Federated Systems Guide*
- "Dropping remote tables using transparent DDL" in the *Federated Systems Guide*

Related reference:

- "CREATE TRIGGER statement" in the *SQL Reference, Volume 2*
- "CREATE VIEW" on page 656
- "CREATE FUNCTION MAPPING statement" in the *SQL Reference, Volume 2*

Related samples:

- "dbstat.sqb -- Reorganize table and run statistics (MF COBOL)"
- "dtstruct.sqC -- Create, use, drop a hierarchy of structured types and typed tables (C++)"
- "tbconstr.sqC -- How to create, use, and drop constraints (C++)"
- "tbcreate.sqC -- How to create and drop tables (C++)"
- "tbtrig.sqC -- How to use a trigger on a table (C++)"
- "DbSeq.java -- How to create, alter and drop a sequence in a database (JDBC)"
- "TbConstr.java -- How to create, use and drop constraints (JDBC)"
- "TbCreate.java -- How to create and drop tables (JDBC)"
- "TbTemp.java -- How to use Declared Temporary Table (JDBC)"
- "TbTrig.java -- How to use triggers (JDBC)"
- "UDFDrop.db2 -- How to uncatalog the Java UDFs contained in UDFsrv.java "
- "spdrop.db2 -- How to uncatalog the stored procedures contained in spserver.sqc (C)"
- "tbconstr.sqc -- How to create, use, and drop constraints (C)"
- "tbcreate.sqc -- How to create and drop tables (C)"
- "tbtemp.sqc -- How to use a declared temporary table (C)"
- "tbtrig.sqc -- How to use a trigger on a table (C)"
- "TbConstr.sqlj -- How to create, use and drop constraints (SQLj)"
- "TbCreate.sqlj -- How to create and drop tables (SQLj)"
- "TbTrig.sqlj -- How to use triggers (SQLj)"

GRANT (Database Authorities)

This form of the GRANT statement grants authorities that apply to the entire database (rather than privileges that apply to specific objects within the database).

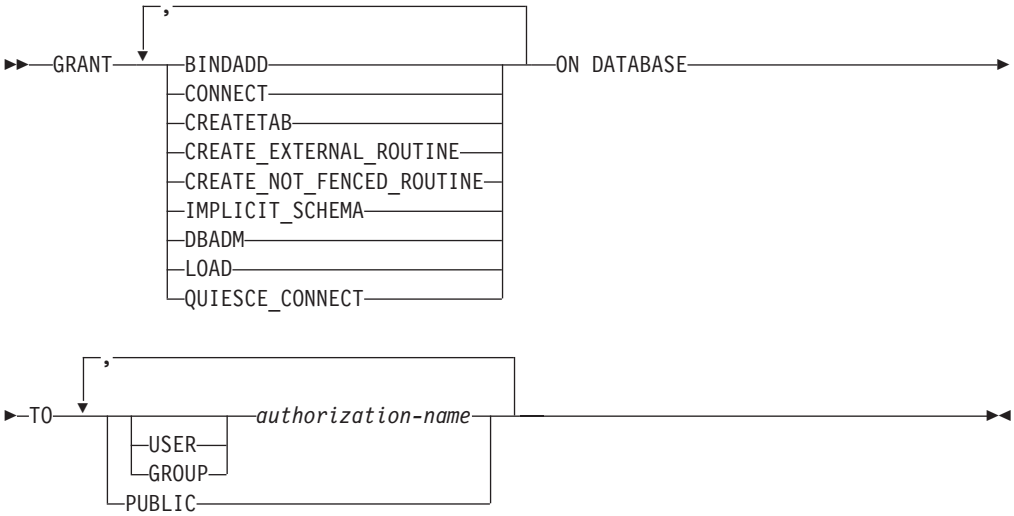
Invocation:

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization:

To grant DBADM authority, SYSADM authority is required. To grant other authorities, either DBADM or SYSADM authority is required.

Syntax:



Description:

BINDADD
Grants the authority to create packages. The creator of a package automatically has the CONTROL privilege on that package and retains this privilege even if the BINDADD authority is subsequently revoked.

CONNECT
Grants the authority to access the database.

CREATETAB
Grants the authority to create base tables. The creator of a base table automatically has the CONTROL privilege on that table. The creator retains this privilege even if the CREATETAB authority is subsequently revoked.

There is no explicit authority required for view creation. A view can be created at any time if the authorization ID of the statement used to create the view has either CONTROL or SELECT privilege on each base table of the view.

CREATE_EXTERNAL_ROUTINE
Grants the authority to register external routines. Care must be taken that routines so registered will not have adverse side effects. (For more information, see the description of the THREADSAFE clause on the CREATE or ALTER routine statements.)

Once an external routine has been registered, it continues to exist, even if CREATE_EXTERNAL_ROUTINE is subsequently revoked.

CREATE_NOT_FENCED_ROUTINE
Grants the authority to register routines that execute in the database manager's

process. Care must be taken that routines so registered will not have adverse side effects. (For more information, see the description of the FENCED clause on the CREATE or ALTER routine statements.)

Once a routine has been registered as not fenced, it continues to run in this manner, even if CREATE_NOT_FENCED_ROUTINE is subsequently revoked.

CREATE_EXTERNAL_ROUTINE is automatically granted to an *authorization-name* that is granted CREATE_NOT_FENCED_ROUTINE authority.

IMPLICIT_SCHEMA
Grants the authority to implicitly create a schema.

DBADM
Grants the database administrator authority and all other database authorities. A database administrator has all privileges against all objects in the database, and can grant these privileges to others.

Note: All other database authorities are implicitly and automatically granted to an *authorization-name* that is granted DBADM authority.

LOAD
Grants the authority to load in this database. This authority gives a user the right to use the LOAD utility in this database. SYSADM and DBADM also have this authority by default. However, if a user only has LOAD authority (not SYSADM or DBADM), the user is also required to have table-level privileges. In addition to LOAD privilege, the user is required to have:

- INSERT privilege on the table for LOAD with mode INSERT, TERMINATE (to terminate a previous LOAD INSERT), or RESTART (to restart a previous LOAD INSERT)
- INSERT and DELETE privilege on the table for LOAD with mode REPLACE, TERMINATE (to terminate a previous LOAD REPLACE), or RESTART (to restart a previous LOAD REPLACE)
- INSERT privilege on the exception table, if such a table is used as part of LOAD

QUIESCE_CONNECT
Grants the authority to access the database while it is quiesced.

TO
Specifies to whom the authorities are granted.

USER
Specifies that the *authorization-name* identifies a user.

GROUP
Specifies that the *authorization-name* identifies a group name.

authorization-name,...
Lists the authorization IDs of one or more users or groups.

The list of authorization IDs cannot include the authorization ID of the user issuing the statement (SQLSTATE 42502).

PUBLIC
Grants the authorities to all users. DBADM cannot be granted to PUBLIC.

Rules:

- If neither USER nor GROUP is specified, then

  - If the authorization-name is defined in the operating system only as GROUP, GROUP is assumed.
  - If the authorization-name is defined in the operating system only as USER, or if it is undefined, USER is assumed.
  - If the authorization-name is defined in the operating system as both, an error (SQLSTATE 56092) is returned.

Notes:

- *Compatibilities*
  - For compatibility with previous versions of DB2:
    - CREATE_NOT_FENCED can be specified in place of CREATE_NOT_FENCED_ROUTINE

Examples:

Example 1: Give the users WINKEN, BLINKEN, and NOD the authority to connect to the database.

```
GRANT CONNECT ON DATABASE TO USER WINKEN, USER BLINKEN, USER NOD
```

Example 2: GRANT BINDADD authority on the database to a group named D024. There is both a group and a user called D024 in the system.

```
GRANT BINDADD ON DATABASE TO GROUP D024
```

Observe that, the GROUP keyword must be specified; otherwise, an error will occur since both a user and a group named D024 exist. Any member of the D024 group will be allowed to bind packages in the database, but the D024 user will not be allowed (unless this user is also a member of the group D024, had been granted BINDADD authority previously, or BINDADD authority had been granted to another group of which D024 was a member).

Related reference:

- "GRANT (Index Privileges)" on page 704
- "GRANT (Package Privileges)" on page 705
- "GRANT (Schema Privileges)" on page 711
- "GRANT (Table, View, or Nickname Privileges)" on page 718
- "GRANT (Server Privileges)" on page 715
- "GRANT (Table Space Privileges)" on page 716
- "GRANT (Sequence Privileges)" on page 713
- "GRANT (Routine Privileges)" on page 708

Related samples:

- "dbauth.sqb -- How to grant and display authorities on a database (MF COBOL)"
- "dbauth.sqc -- How to grant, display, and revoke authorities at database level (C)"
- "dbauth.sqC -- How to grant, display, and revoke authorities at database level (C++)"
- "DbAuth.java -- Grant, display or revoke privileges on database (JDBC)"
- "DbAuth.sqlj -- Grant, display or revoke privileges on database (SQLj)"

GRANT (Index Privileges)

This form of the GRANT statement grants the CONTROL privilege on indexes.

Invocation:

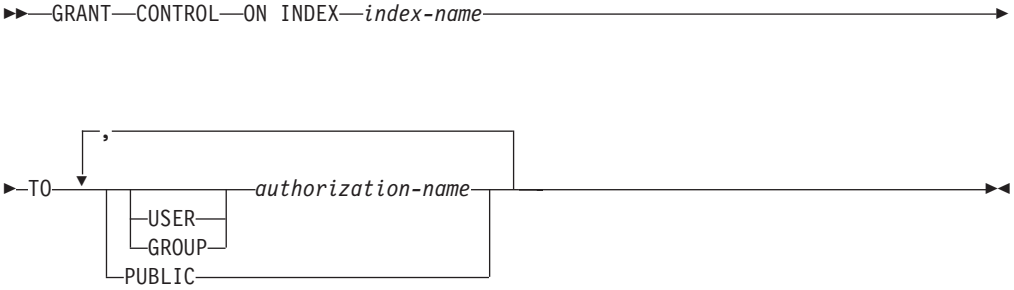
This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization:

The privileges held by the authorization ID of the statement must include at least one of the following:

- DBADM authority
- SYSADM authority.

Syntax:



Description:

CONTROL
Grants the privilege to drop the index. This is the CONTROL authority for indexes, which is automatically granted to creators of indexes.

ON INDEX *index-name*
Identifies the index for which the CONTROL privilege is to be granted.

TO
Specifies to whom the privileges are granted.

USER
Specifies that the *authorization-name* identifies a user.

GROUP
Specifies that the *authorization-name* identifies a group name.

authorization-name,...
Lists the authorization IDs of one or more users or groups.

The list of authorization IDs cannot include the authorization ID of the user issuing the statement (SQLSTATE 42502).

PUBLIC
Grants the privileges to all users.

Rules:

- If neither USER nor GROUP is specified, then
  - If the authorization-name is defined in the operating system only as GROUP, then GROUP is assumed.
  - If the authorization-name is defined in the operating system only as USER or if it is undefined, USER is assumed.
  - If the authorization-name is defined in the operating system as both, an error (SQLSTATE 56092) is returned.

Example:

```
GRANT CONTROL ON INDEX DEPTIDX TO USER USER4
```

Related reference:

- "GRANT (Database Authorities)" on page 700
- "GRANT (Package Privileges)" on page 705
- "GRANT (Schema Privileges)" on page 711
- "GRANT (Table, View, or Nickname Privileges)" on page 718
- "GRANT (Server Privileges)" on page 715
- "GRANT (Table Space Privileges)" on page 716
- "GRANT (Sequence Privileges)" on page 713
- "GRANT (Routine Privileges)" on page 708

GRANT (Package Privileges)

This form of the GRANT statement grants privileges on a package.

Invocation:

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

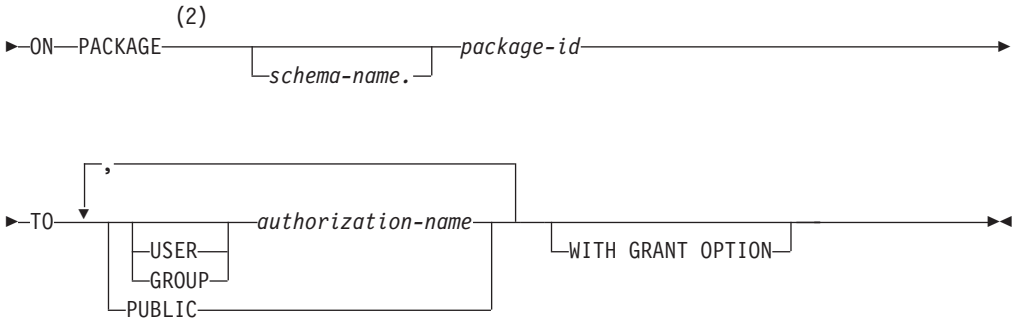
Authorization:

The privileges held by the authorization ID of the statement must include at least one of the following:

- CONTROL privilege on the referenced package
- The WITH GRANT OPTION for each identified privilege on *package-name*.
- SYSADM or DBADM authority.

To grant the CONTROL privilege, SYSADM or DBADM authority is required.

Syntax:



Notes:

1 RUN can be used as a synonym for EXECUTE.

2 PROGRAM can be used as a synonym for PACKAGE.

Description:

BIND
Grants the privilege to bind a package. The BIND privilege allows a user to re-issue the BIND command against that package, or to issue the REBIND command. It also allows a user to create a new version of an existing package.

In addition to the BIND privilege, a user must hold the necessary privileges on each table referenced by static DML statements contained in a program. This is necessary, because authorization on static DML statements is checked at bind time.

CONTROL
Grants the privilege to rebind, drop, or execute the package, and extend package privileges to other users. The CONTROL privilege for packages is automatically granted to creators of packages. A package owner is the package binder, or the ID specified with the OWNER option at bind/precompile time.

BIND and EXECUTE are automatically granted to an *authorization-name* that is granted CONTROL privilege.

CONTROL grants the ability to grant the above privileges (except for CONTROL) to others.

EXECUTE
Grants the privilege to execute the package.

ON PACKAGE *schema-name.package-id*
Specifies the name of the package on which privileges are to be granted. If a schema name is not specified, the package ID is implicitly qualified by the default schema. The granting of a package privilege applies to all versions of the package (that is, to all packages that share the same package ID and package schema).

TO
Specifies to whom the privileges are granted.

USER
Specifies that the *authorization-name* identifies a user.

GROUP
Specifies that the *authorization-name* identifies a group name.

authorization-name,...
Lists the authorization IDs of one or more users or groups.

The list of authorization IDs cannot include the authorization ID of the user issuing the statement (SQLSTATE 42502).

PUBLIC
Grants the privileges to all users.

WITH GRANT OPTION
Allows the specified *authorization-name* to GRANT the privileges to others.

If the specified privileges include CONTROL, the WITH GRANT OPTION applies to all of the applicable privileges except for CONTROL (SQLSTATE 01516).

Rules:

- If neither USER nor GROUP is specified, then
  - If the *authorization-name* is defined in the operating system only as GROUP, then GROUP is assumed.
  - If the *authorization-name* is defined in the operating system only as USER or if it is undefined, USER is assumed.
  - If the *authorization-name* is defined in the operating system as both, an error (SQLSTATE 56092) is returned.

Notes:

- Package privileges apply to all versions of a package (that is, all packages that share the same package ID and package schema). It is not possible to restrict access to only one version. Because CONTROL privilege is implicitly granted to the binder of a package, if two different users bind two versions of a package, then both users will implicitly be granted access to each other's package.

Examples:

Example 1: Grant the EXECUTE privilege on PACKAGE CORPDATA.PKGA to PUBLIC.

```
GRANT EXECUTE
  ON PACKAGE CORPDATA.PKGA
  TO PUBLIC
```

Example 2: GRANT EXECUTE privilege on package CORPDATA.PKGA to a user named EMPLOYEE. There is neither a group nor a user called EMPLOYEE.

```
GRANT EXECUTE ON PACKAGE
  CORPDATA.PKGA TO EMPLOYEE
```

or

```
GRANT EXECUTE ON PACKAGE
  CORPDATA.PKGA TO USER EMPLOYEE
```

Related reference:

- "GRANT (Database Authorities)" on page 700
- "GRANT (Index Privileges)" on page 704
- "GRANT (Schema Privileges)" on page 711
- "GRANT (Table, View, or Nickname Privileges)" on page 718
- "GRANT (Server Privileges)" on page 715
- "GRANT (Table Space Privileges)" on page 716
- "GRANT (Sequence Privileges)" on page 713

• "GRANT (Routine Privileges)" on page 708

GRANT (Routine Privileges)

This form of the GRANT statement grants privileges on a routine (function, method, or procedure).

Invocation:

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization:

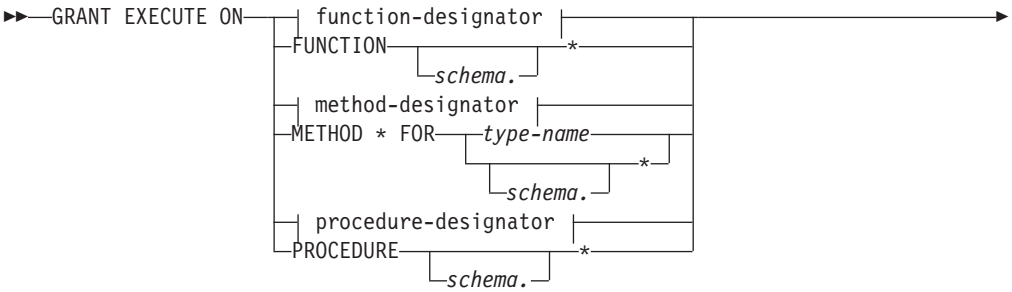
The privileges held by the authorization ID of the statement must include at least one of the following:

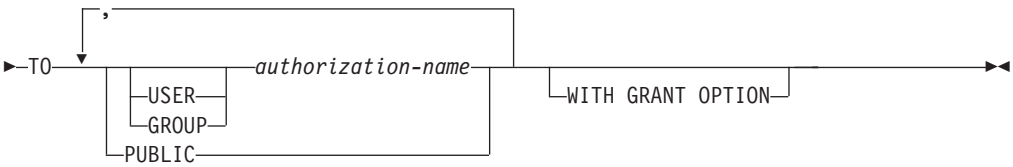
- WITH GRANT OPTION for EXECUTE on the routine
- SYSADM or DBADM authority

To grant all routine EXECUTE privileges in the schema or type, the privileges held by the authorization ID must include at least one of the following:

- WITH GRANT OPTION for EXECUTE on all existing and future routines (of the specified type) in the specified schema
- SYSADM or DBADM authority

Syntax:





Description:

EXECUTE
Grants the privilege to run the identified user-defined function, method, or stored procedure.

function-designator
Uniquely identifies the function.

FUNCTION *schema*.*
Identifies all the functions in the schema, including any functions that may be created in the future. In dynamic SQL statements, if a schema is not specified, the schema in the CURRENT SCHEMA special register will be used. In static SQL statements, if a schema is not specified, the schema in the QUALIFIER precompile/bind option will be used.

method-designator
Uniquely identifies the method.

METHOD *
Identifies all the methods for the type *type-name*, including any methods that may be created in the future.

FOR *type-name*
Names the type in which the specified method is found. The name must identify a type already described in the catalog (SQLSTATE 42704). In dynamic SQL statements, the value of the CURRENT SCHEMA special register is used as a qualifier for an unqualified type name. In static SQL statements, the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified type names. An asterisk (*) can be used in place of *type-name* to identify all types in the schema, including any types that may be created in the future.

procedure-designator
Uniquely identifies the procedure.

PROCEDURE *schema*.*
Identifies all the procedures in the schema, including any procedures that may be created in the future. In dynamic SQL statements, if a schema is not specified, the schema in the CURRENT SCHEMA special register will be used. In static SQL statements, if a schema is not specified, the schema in the QUALIFIER precompile/bind option will be used.

TO
Specifies to whom the EXECUTE privilege is granted.

USER
Specifies that the *authorization-name* identifies a user.

GROUP
Specifies that the *authorization-name* identifies a group name.

authorization-name,...
Lists the authorization IDs of one or more users or groups.

PUBLIC
Grants the EXECUTE privilege to all users.

WITH GRANT OPTION
Allows the specified *authorization-name*s to GRANT the EXECUTE privilege to others.

If the WITH GRANT OPTION is omitted, the specified *authorization-name* can only grant the EXECUTE privilege to others if they:

- have SYSADM or DBADM authority or
- received the ability to grant the EXECUTE privilege from some other source.

Rules:

- It is not possible to grant the EXECUTE privilege on a function or method defined with schema 'SYSIBM' or 'SYSFUN' (SQLSTATE 42832).

- If neither USER nor GROUP is specified, then:
  - If the *authorization-name* is defined in the operating system only as GROUP, then GROUP is assumed.
  - If the *authorization-name* is defined in the operating system only as USER or if it is undefined, USER is assumed.
  - If the *authorization-name* is defined in the operating system as both USER and GROUP, an error (SQLSTATE 56092) is returned.
- In general, the GRANT statement will process the granting of privileges that the authorization ID of the statement is allowed to grant, returning a warning (SQLSTATE 01007) if one or more privileges was not granted. If the package used for processing the statement was precompiled with LANGLEVEL set to SQL92E or MIA, and no privileges were granted, a warning is returned (SQLSTATE 01007). If the grantor has no privileges on the object of the grant operation, an error is returned (SQLSTATE 42501).

Examples:

Example 1: Grant the EXECUTE privilege on function CALC_SALARY to user JONES. Assume that there is only one function in the schema with function name CALC_SALARY.

```
GRANT EXECUTE ON FUNCTION CALC_SALARY TO JONES
```

Example 2: Grant the EXECUTE privilege on procedure VACATION_ACCR to all users at the current server.

```
GRANT EXECUTE ON PROCEDURE VACATION_ACCR TO PUBLIC
```

Example 3: Grant the EXECUTE privilege on function DEPT_TOTALS to the administrative assistant and give the assistant the ability to grant the EXECUTE privilege on this function to others. The function has the specific name DEPT85_TOT. Assume that the schema has more than one function named DEPT_TOTALS.

```
GRANT EXECUTE ON SPECIFIC FUNCTION DEPT85_TOT
  TO ADMIN_A WITH GRANT OPTION
```

Example 4: Grant the EXECUTE privilege on function NEW_DEPT_HIRES to HR (Human Resources). The function has two input parameters of type INTEGER and CHAR(10), respectively. Assume that the schema has more than one function named NEW_DEPT_HIRES.

```
GRANT EXECUTE ON FUNCTION NEW_DEPT_HIRES (INTEGER, CHAR(10)) TO HR
```

Example 5: Grant the EXECUTE privilege on method SET_SALARY of type EMPLOYEE to user JONES.

```
GRANT EXECUTE ON METHOD SET_SALARY FOR EMPLOYEE TO JONES
```

Related reference:

- “GRANT (Database Authorities)” on page 700
- “GRANT (Index Privileges)” on page 704
- “GRANT (Package Privileges)” on page 705
- “GRANT (Schema Privileges)” on page 711
- “GRANT (Table, View, or Nickname Privileges)” on page 718
- “GRANT (Server Privileges)” on page 715
- “GRANT (Table Space Privileges)” on page 716

- "GRANT (Sequence Privileges)" on page 713
- "Common syntax elements" in the *SQL Reference, Volume 2*

GRANT (Schema Privileges)

This form of the GRANT statement grants privileges on a schema.

Invocation:

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

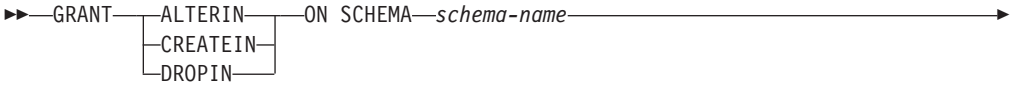
Authorization:

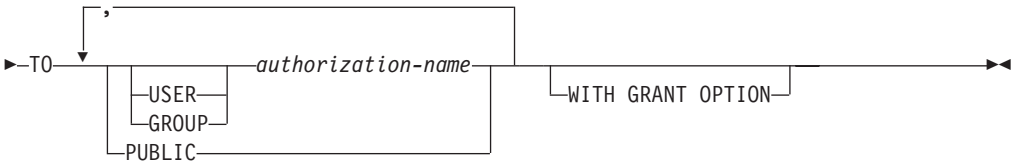
The privileges held by the authorization ID of the statement must include at least one of the following:

- WITH GRANT OPTION for each identified privilege on *schema-name*
- SYSADM or DBADM authority

Privileges cannot be granted on schema names SYSIBM, SYSCAT, SYSFUN and SYSSTAT by any user (SQLSTATE 42501).

Syntax:





Description:

ALTERIN
Grants the privilege to alter or comment on all objects in the schema. The owner of an explicitly created schema automatically receives ALTERIN privilege.

CREATEIN
Grants the privilege to create objects in the schema. Other authorities or privileges required to create the object (such as CREATETAB) are still required. The owner of an explicitly created schema automatically receives CREATEIN privilege. An implicitly created schema has CREATEIN privilege automatically granted to PUBLIC.

DROPIN
Grants the privilege to drop all objects in the schema. The owner of an explicitly created schema automatically receives DROPIN privilege.

ON SCHEMA *schema-name*
Identifies the schema on which the privileges are to be granted.

TO
Specifies to whom the privileges are granted.

USER
Specifies that the *authorization-name* identifies a user.

GROUP
Specifies that the *authorization-name* identifies a group name.

authorization-name,...
Lists the authorization IDs of one or more users or groups.

The list of authorization IDs cannot include the authorization ID of the user issuing the statement (SQLSTATE 42502).

PUBLIC
Grants the privileges to all users.

WITH GRANT OPTION
Allows the specified *authorization-name*s to GRANT the privileges to others.

If the WITH GRANT OPTION is omitted, the specified *authorization-name*s can only grant the privileges to others if they:

- have DBADM authority or
- received the ability to grant privileges from some other source.

Rules:

- If neither USER nor GROUP is specified, then
  - If the authorization-name is defined in the operating system only as GROUP, then GROUP is assumed.
  - If the authorization-name is defined in the operating system only as USER or if it is undefined, USER is assumed.
  - If the authorization-name is defined in the operating system as both, an error (SQLSTATE 56092) is returned.
- In general, the GRANT statement will process the granting of privileges that the authorization ID of the statement is allowed to grant, returning a warning (SQLSTATE 01007) if one or more privileges was not granted. If no privileges were granted, an error is returned (SQLSTATE 42501). (If the package used for processing the statement was precompiled with LANGLEVEL set to SQL92E for MIA, a warning is returned (SQLSTATE 01007), unless the grantor has no privileges on the object of the grant operation.)

Examples:

Example 1: Grant user JSINGLETON to the ability to create objects in schema CORPDATA.

```
GRANT CREATEIN ON SCHEMA CORPDATA TO JSINGLETON
```

Example 2: Grant user IHAKES the ability to create and drop objects in schema CORPDATA.

```
GRANT CREATEIN, DROPIN ON SCHEMA CORPDATA TO IHAKES
```

Related reference:

- "GRANT (Database Authorities)" on page 700

- "GRANT (Index Privileges)" on page 704
- "GRANT (Package Privileges)" on page 705
- "GRANT (Table, View, or Nickname Privileges)" on page 718
- "GRANT (Server Privileges)" on page 715
- "GRANT (Table Space Privileges)" on page 716
- "GRANT (Sequence Privileges)" on page 713
- "GRANT (Routine Privileges)" on page 708

GRANT (Sequence Privileges)

This form of the GRANT statement grants privileges on a sequence.

Invocation:

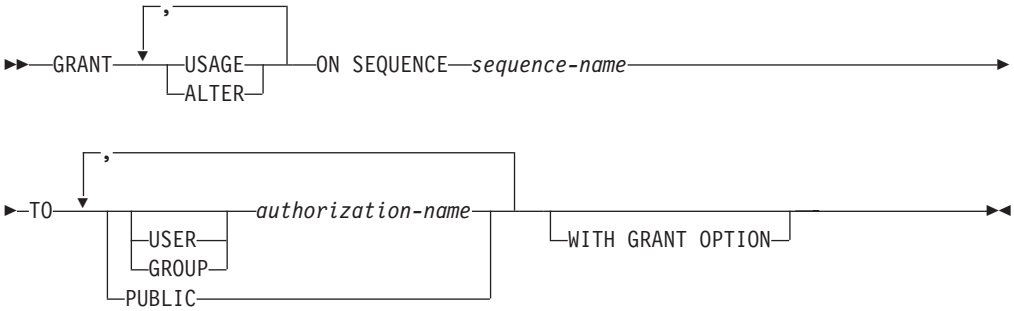
This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization:

The privileges held by the authorization ID of the statement must include at least one of the following:

- WITH GRANT OPTION for each identified privilege on *sequence-name*
- SYSADM or DBADM authority

Syntax:



Description:

USAGE
Grants the privilege to reference a sequence using *nextval-expression* or *prevval-expression*.

ALTER
Grants the privilege to alter sequence properties using the ALTER SEQUENCE statement.

ON SEQUENCE *sequence-name*
Identifies the sequence on which the specified privileges are to be granted. The sequence name, including an implicit or explicit schema qualifier, must uniquely identify an existing sequence at the current server. If no sequence by this name exists, an error (SQLSTATE 42704) is returned.

TO
Specifies to whom the specified privileges are granted.

USER
Specifies that the *authorization-name* identifies a user.

GROUP
Specifies that the *authorization-name* identifies a group.

authorization-name,...
Lists the authorization IDs of one or more users or groups.

PUBLIC
Grants the specified privileges to all users.

WITH GRANT OPTION
Allows the specified *authorization-name* to grant the specified privileges to others.

If the WITH GRANT OPTION is omitted, the specified *authorization-name* can only grant the specified privileges to others if they:

- have SYSADM or DBADM authority or
- received the ability to grant the specified privileges from some other source.

Rules:

- If neither USER nor GROUP is specified, then:
  - If the *authorization-name* is defined in the operating system only as GROUP, then GROUP is assumed.
  - If the *authorization-name* is defined in the operating system only as USER or if it is undefined, then USER is assumed.
  - If the *authorization-name* is defined in the operating system as both USER and GROUP, an error (SQLSTATE 56092) is returned.
- In general, the GRANT statement will process the granting of privileges that the authorization ID of the statement is allowed to grant, returning a warning (SQLSTATE 01007) if one or more privileges is not granted. If no privileges are granted, an error is returned (SQLSTATE 42501). (If the package used for processing the statement was precompiled with LANGLEVEL set to SQL92E or MIA, a warning is returned (SQLSTATE 01007), unless the grantor has no privileges on the object of the grant operation.)

Example:

Example 1: Grant any user the USAGE privilege on a sequence called ORG_SEQ.

```
GRANT USAGE ON SEQUENCE ORG_SEQ TO PUBLIC
```

Example 2: Grant user BOBBY the ability to alter a sequence called GENERATE_ID, and to grant this privilege to others.

```
GRANT ALTER ON SEQUENCE GENERATE_ID TO BOBBY WITH GRANT OPTION
```

Related reference:

- "GRANT (Database Authorities)" on page 700
- "GRANT (Index Privileges)" on page 704
- "GRANT (Package Privileges)" on page 705
- "GRANT (Schema Privileges)" on page 711
- "GRANT (Table, View, or Nickname Privileges)" on page 718

- "GRANT (Server Privileges)" on page 715
- "GRANT (Table Space Privileges)" on page 716
- "GRANT (Routine Privileges)" on page 708

GRANT (Server Privileges)

This form of the GRANT statement grants the privilege to access and use a specified data source in pass-through mode.

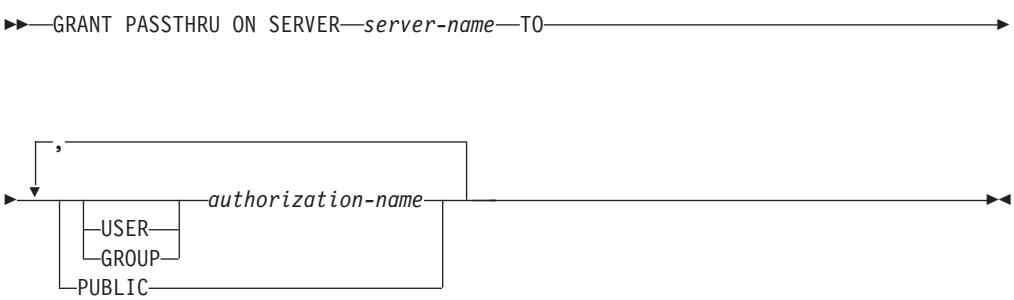
Invocation:

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization:

The authorization ID of the statement must have either SYSADM or DBADM authority.

Syntax:



Description:

server-name
Names the data source for which the privilege to use in pass-through mode is being granted. *server-name* must identify a data source that is described in the catalog.

TO
Specifies to whom the privilege is granted.

USER
Specifies that the *authorization-name* identifies a user.

GROUP
Specifies that the *authorization-name* identifies a group name.

authorization-name,...
Lists the authorization IDs of one or more users or groups.

The list of authorization IDs cannot include the authorization ID of the user issuing the statement (SQLSTATE 42502).

PUBLIC
Grants to all users the privilege to pass through to *server-name.*

Examples:

Example 1: Give R. Smith and J. Jones the privilege to pass through to data source SERVALL. Their authorization IDs are RSMITH and JJONES.

```
   GRANT PASSTHRU ON SERVER SERVALL
TO USER RSMITH,
USER JJONES
```

Example 2: Grant the privilege to pass through to data source EASTWING to a group whose authorization ID is D024. There is a user whose authorization ID is also D024.

```
   GRANT PASSTHRU ON SERVER EASTWING TO GROUP D024
```

The GROUP keyword must be specified; otherwise, an error will occur because D024 is a user's ID as well as the specified group's ID (SQLSTATE 56092). Any member of group D024 will be allowed to pass through to EASTWING. Therefore, if user D024 belongs to the group, this user will be able to pass through to EASTWING.

Related reference:

- "GRANT (Database Authorities)" on page 700
- "GRANT (Index Privileges)" on page 704
- "GRANT (Package Privileges)" on page 705
- "GRANT (Schema Privileges)" on page 711
- "GRANT (Table, View, or Nickname Privileges)" on page 718
- "GRANT (Table Space Privileges)" on page 716
- "GRANT (Sequence Privileges)" on page 713
- "GRANT (Routine Privileges)" on page 708

GRANT (Table Space Privileges)

This form of the GRANT statement grants privileges on a table space.

Invocation:

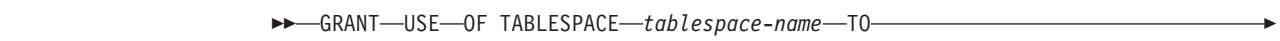
This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

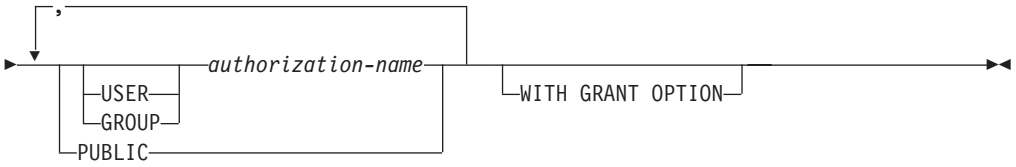
Authorization:

The privileges held by the authorization ID of the statement must include at least one of the following:

- WITH GRANT OPTION for use of the table space
- SYSADM, SYSCTRL, or DBADM authority

Syntax:





Description:

USE
Grants the privilege to specify or default to the table space when creating a table. The creator of a table space automatically receives USE privilege with grant option.

OF TABLESPACE *tablespace-name*
Identifies the table space on which the USE privilege is to be granted. The table space cannot be SYSCATSPACE (SQLSTATE 42838) or a system temporary table space (SQLSTATE 42809).

TO
Specifies to whom the USE privilege is granted.

USER
Specifies that the *authorization-name* identifies a user.

GROUP
Specifies that the *authorization-name* identifies a group name.

authorization-name
Lists the authorization IDs of one or more users or groups.

The list of authorization IDs cannot include the authorization ID of the user issuing the statement (SQLSTATE 42502).

PUBLIC
Grants the USE privilege to all users.

WITH GRANT OPTION
Allows the specified *authorization-name* to GRANT the USE privilege to others.

If the WITH GRANT OPTION is omitted, the specified *authorization-name* can only GRANT the USE privilege to others if they:

- have SYSADM or DBADM authority or
- received the ability to GRANT the USE privilege from some other source.

Notes:

If neither USER nor GROUP is specified, then

- If the *authorization-name* is defined in the operating system only as GROUP, then GROUP is assumed.
- If the *authorization-name* is defined in the operating system only as USER, or if it is undefined, then USER is assumed.
- If the *authorization-name* is defined in the operating system as both, an error (SQLSTATE 56092) is returned.

Examples:

Example 1: Grant user BOBBY the ability to create tables in table space PLANS and to grant this privilege to others.

```
GRANT USE OF TABLESPACE PLANS TO BOBBY WITH GRANT OPTION
```

Related reference:

- "GRANT (Database Authorities)" on page 700
- "GRANT (Index Privileges)" on page 704
- "GRANT (Package Privileges)" on page 705
- "GRANT (Schema Privileges)" on page 711
- "GRANT (Table, View, or Nickname Privileges)" on page 718
- "GRANT (Server Privileges)" on page 715
- "GRANT (Sequence Privileges)" on page 713
- "GRANT (Routine Privileges)" on page 708

GRANT (Table, View, or Nickname Privileges)

This form of the GRANT statement grants privileges on a table, view, or nickname.

Invocation:

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization:

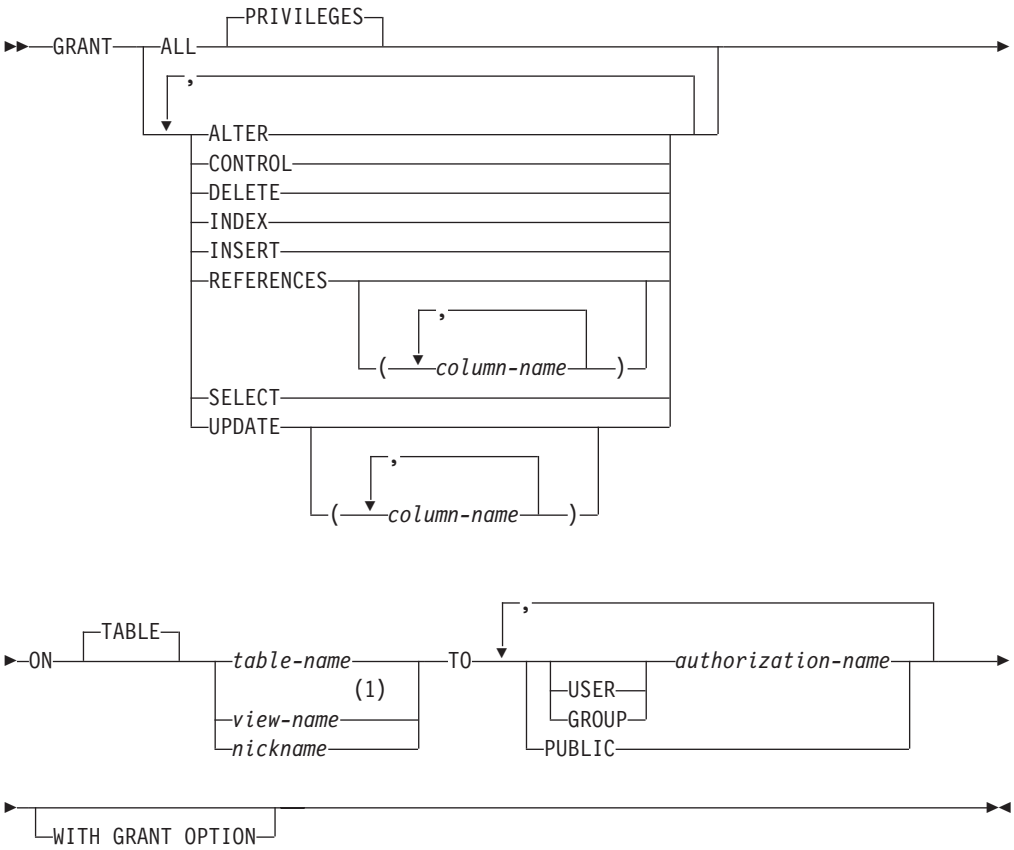
The privileges held by the authorization ID of the statement must include at least one of the following:

- CONTROL privilege on the referenced table, view, or nickname
- The WITH GRANT OPTION for each identified privilege. If ALL is specified, the authorization ID must have some grantable privilege on the identified table, view, or nickname.
- SYSADM or DBADM authority.

To grant the CONTROL privilege, SYSADM or DBADM authority is required.

To grant privileges on catalog tables and views, either SYSADM or DBADM authority is required.

Syntax:



Notes:

1 ALTER, INDEX, and REFERENCES privileges are not applicable to views.

Description:

ALL or **ALL PRIVILEGES**
Grants all the appropriate privileges, except CONTROL, on the base table, view, or nickname named in the ON clause.

If the authorization ID of the statement has CONTROL privilege on the table, view, or nickname, or DBADM or SYSADM authority, then all the privileges applicable to the object (except CONTROL) are granted. Otherwise, the privileges granted are all those grantable privileges that the authorization ID of the statement has on the identified table, view, or nickname.

If ALL is not specified, one or more of the keywords in the list of privileges must be specified.

ALTER
Grants the privilege to:

- Add columns to a base table definition.
- Create or drop a primary key or unique constraint on a base table.
- Create or drop a foreign key on a base table.

  The REFERENCES privilege on each column of the parent table is also required.
- Create or drop a check constraint on a base table.

- Create a trigger on a base table.
- Add, reset, or drop a column option for a nickname.
- Change a nickname column name or data type.
- Add or change a comment on a base table or a nickname.

CONTROL
Grants:

- All of the appropriate privileges in the list, that is:
  - ALTER, CONTROL, DELETE, INSERT, INDEX, REFERENCES, SELECT, and UPDATE to base tables
  - CONTROL, DELETE, INSERT, SELECT, and UPDATE to views
  - ALTER, CONTROL, INDEX, and REFERENCES to nicknames
- The ability to grant the above privileges (except for CONTROL) to others.
- The ability to drop the base table, view, or nickname.

  This ability cannot be extended to others on the basis of holding CONTROL privilege. The only way that it can be extended is by granting the CONTROL privilege itself and that can only be done by someone with SYSADM or DBADM authority.
- The ability to execute the RUNSTATS utility on the table and indexes.
- The ability to execute the REORG utility on the table.
- The ability to issue the SET INTEGRITY statement against a base table, materialized query table, or staging table.

The definer of a base table, materialized query table, staging table, or nickname automatically receives the CONTROL privilege.

The definer of a view automatically receives the CONTROL privilege if the definer holds the CONTROL privilege on all tables, views, and nicknames identified in the fullselect.

DELETE
Grants the privilege to delete rows from the table or updatable view.

INDEX
Grants the privilege to create an index on a table, or an index specification on a nickname. This privilege cannot be granted on a view. The creator of an index or index specification automatically has the CONTROL privilege on the index or index specification (authorizing the creator to drop the index or index specification). In addition, the creator retains the CONTROL privilege even if the INDEX privilege is revoked.

INSERT
Grants the privilege to insert rows into the table or updatable view and to run the IMPORT utility.

REFERENCES
Grants the privilege to create and drop a foreign key referencing the table as the parent.

If the authorization ID of the statement has one of:

- DBADM or SYSADM authority
- CONTROL privilege on the table
- REFERENCES WITH GRANT OPTION on the table

then the grantee(s) can create referential constraints using all columns of the table as parent key, even those added later using the ALTER TABLE statement.

Otherwise, the privileges granted are all those grantable column REFERENCES privileges that the authorization ID of the statement has on the identified table.

The privilege can be granted on a nickname, although foreign keys cannot be defined to reference nicknames.

REFERENCES (*column-name***,...)**
Grants the privilege to create and drop a foreign key using only those columns specified in the column list as a parent key. Each *column-name* must be an unqualified name that identifies a column of the table identified in the ON clause. Column level REFERENCES privilege cannot be granted on typed tables, typed views, or nicknames (SQLSTATE 42997).

SELECT
Grants the privilege to:

- Retrieve rows from the table or view.
- Create views on the table.
- Run the EXPORT utility against the table or view.

UPDATE
Grants the privilege to use the UPDATE statement on the table or updatable view identified in the ON clause.

If the authorization ID of the statement has one of:

- DBADM or SYSADM authority
- CONTROL privilege on the table or view
- UPDATE WITH GRANT OPTION on the table or view

then the grantee(s) can update all updatable columns of the table or view on which the grantor has with grant privilege as well as those columns added later using the ALTER TABLE statement. Otherwise, the privileges granted are all those grantable column UPDATE privileges that the authorization ID of the statement has on the identified table or view.

UPDATE (*column-name***,...)**
Grants the privilege to use the UPDATE statement to update only those columns specified in the column list. Each *column-name* must be an unqualified name that identifies a column of the table or view identified in the ON clause. Column level UPDATE privilege cannot be granted on typed tables, typed views, or nicknames (SQLSTATE 42997).

ON TABLE *table-name* or *view-name* or *nickname*
Specifies the table, view, or nickname on which privileges are to be granted.

No privileges may be granted on an inoperative view or an inoperative materialized query table (SQLSTATE 51024). No privileges may be granted on a declared temporary table (SQLSTATE 42995).

TO
Specifies to whom the privileges are granted.

USER
Specifies that the *authorization-name* identifies a user.

GROUP
Specifies that the *authorization-name* identifies a group name.

authorization-name,...
Lists the authorization IDs of one or more users or groups. (Previous restrictions on grants to the authorization ID of the user issuing the statement have been removed.)

A privilege that is granted to a group is not used for authorization checking:

- On static DML statements in a package
- On a base table while processing a CREATE VIEW statement
- On a base table while processing a CREATE TABLE statement for a materialized query table

In DB2 Universal Database, table privileges granted to groups only apply to statements that are dynamically prepared. For example, if the INSERT privilege on the PROJECT table has been granted to group D204 but not UBIQUITY (a member of D204) UBIQUITY could issue the statement:

```
EXEC SQL EXECUTE IMMEDIATE :INSERT_STRING;
```

where the content of the string is:

```
INSERT INTO PROJECT (PROJNO, PROJNAME, DEPTNO, RESPEMP)
VALUES ('AD3114', 'TOOL PROGRAMMING', 'D21', '000260');
```

but could not precompile or bind a program with the statement:

```
EXEC SQL INSERT INTO PROJECT (PROJNO, PROJNAME, DEPTNO, RESPEMP)
VALUES ('AD3114', 'TOOL PROGRAMMING', 'D21', '000260');
```

PUBLIC
Grants the privileges to all users. (Previous restrictions on the use of privileges granted to PUBLIC for static SQL statements and the CREATE VIEW statement have been removed.)

WITH GRANT OPTION
Allows the specified *authorization-name*s to GRANT the privileges to others.

If the specified privileges include CONTROL, the WITH GRANT OPTION applies to all the applicable privileges except for CONTROL (SQLSTATE 01516).

Rules:

- If neither USER nor GROUP is specified, then
  - If the *authorization-name* is defined in the operating system only as GROUP, then GROUP is assumed.
  - If the *authorization-name* is defined in the operating system only as USER or if it is undefined, USER is assumed.
  - If the *authorization-name* is defined in the operating system as both, an error (SQLSTATE 56092) is returned.
- In general, the GRANT statement will process the granting of privileges that the authorization ID of the statement is allowed to grant, returning a warning (SQLSTATE 01007) if one or more privileges was not granted. If no privileges were granted, an error is returned (SQLSTATE 42501). (If the package used for processing the statement was precompiled with LANGLEVEL set to SQL92E or MIA, a warning is returned (SQLSTATE 01007), unless the grantor has no privileges on the object of the grant operation.) If CONTROL privilege is specified, privileges will only be granted if the authorization ID of the statement has SYSADM or DBADM authority (SQLSTATE 42501).

Notes:

- *Compatibilities*
  - For compatibility with DB2 UDB for OS/390 and z/OS:
    - The following syntax is tolerated and ignored:
      - PUBLIC AT ALL LOCATIONS
- Privileges may be granted independently at every level of a table hierarchy. A user with a privilege on a supertable may affect the subtables. For example, an update specifying the supertable *T* may show up as a change to a row in the subtable *S* of *T* done by a user with UPDATE privilege on *T* but without UPDATE privilege on *S*. A user can only operate directly on the subtable if the necessary privilege is held on the subtable.
- Granting nickname privileges has no effect on data source object (table or view) privileges. Typically, data source privileges are required for the table or view that a nickname references when attempting to retrieve data.

Examples:

Example 1: Grant all privileges on the table WESTERN_CR to PUBLIC.

```
GRANT ALL ON WESTERN_CR
   TO PUBLIC
```

Example 2: Grant the appropriate privileges on the CALENDAR table so that users PHIL and CLAIRE can read it and insert new entries into it. Do not allow them to change or remove any existing entries.

```
GRANT SELECT, INSERT ON CALENDAR
   TO USER  PHIL, USER CLAIRE
```

Example 3: Grant all privileges on the COUNCIL table to user FRANK and the ability to extend all privileges to others.

```
GRANT ALL ON COUNCIL
   TO USER FRANK WITH GRANT OPTION
```

Example 4: GRANT SELECT privilege on table CORPDATA.EMPLOYEE to a user named JOHN. There is a user called JOHN and no group called JOHN.

```
GRANT SELECT ON CORPDATA.EMPLOYEE TO JOHN
```

or

```
GRANT SELECT
  ON CORPDATA.EMPLOYEE TO USER JOHN
```

Example 5: GRANT SELECT privilege on table CORPDATA.EMPLOYEE to a group named JOHN. There is a group called JOHN and no user called JOHN.

```
GRANT SELECT ON CORPDATA.EMPLOYEE TO JOHN
```

or

```
GRANT SELECT ON CORPDATA.EMPLOYEE TO GROUP JOHN
```

Example 6: GRANT INSERT and SELECT on table T1 to both a group named D024 and a user named D024.

```
GRANT INSERT, SELECT ON TABLE T1
  TO GROUP D024, USER D024
```

In this case, both the members of the D024 group and the user D024 would be allowed to INSERT into and SELECT from the table T1. Also, there would be two rows added to the SYSCAT.TABAUTH catalog view.

Example 7: GRANT INSERT, SELECT, and CONTROL on the CALENDAR table to user FRANK. FRANK must be able to pass the privileges on to others.

```
GRANT CONTROL ON TABLE CALENDAR
  TO FRANK WITH GRANT OPTION
```

The result of this statement is a warning (SQLSTATE 01516) that CONTROL was not given the WITH GRANT OPTION. Frank now has the ability to grant any privilege on CALENDAR including INSERT and SELECT as required. FRANK cannot grant CONTROL on CALENDAR to other users unless he has SYSADM or DBADM authority.

Example 8: User JON created a nickname for an Oracle table that had no index. The nickname is ORAREM1. Later, the Oracle DBA defined an index for this table. User SHAWN now wants DB2 to know that this index exists, so that the optimizer can devise strategies to access the table more efficiently. SHAWN can inform DB2 of the index by creating an index specification for ORAREM1. Give SHAWN the index privilege on this nickname, so that he can create the index specification.

```
GRANT INDEX ON NICKNAME ORAREM1
  TO USER SHAWN
```

Related reference:

- "ALTER TABLE" on page 525
- "GRANT (Database Authorities)" on page 700
- "GRANT (Index Privileges)" on page 704
- "GRANT (Package Privileges)" on page 705
- "GRANT (Schema Privileges)" on page 711
- "GRANT (Server Privileges)" on page 715
- "GRANT (Table Space Privileges)" on page 716
- "GRANT (Sequence Privileges)" on page 713
- "GRANT (Routine Privileges)" on page 708

Related samples:

- "tbpriv.sqc -- How to grant, display, and revoke privileges (C)"
- "tbpriv.sqC -- How to grant, display, and revoke privileges (C++)"
- "TbPriv.java -- How to grant, display and revoke privileges on a table (JDBC)"
- "TbPriv.sqlj -- How to grant, display and revoke privileges on a table (SQLj)"

INSERT

The INSERT statement inserts rows into a table, nickname, or view, or the underlying tables, nicknames, or views of the specified fullselect. Inserting a row into a nickname inserts the row into the data source object to which the nickname refers. Inserting a row into a view also inserts the row into the table on which the view is based, if no INSTEAD OF trigger is defined for the insert operation on this view. If such a trigger is defined, the trigger will be executed instead.

Invocation:

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared.

Authorization:

To execute this statement, the privileges held by the authorization ID of the statement must include at least one of the following:

- INSERT privilege on the table, view or nickname where rows are to be inserted
- CONTROL privilege on the table, view or nickname where rows are to be inserted
- SYSADM or DBADM authority.

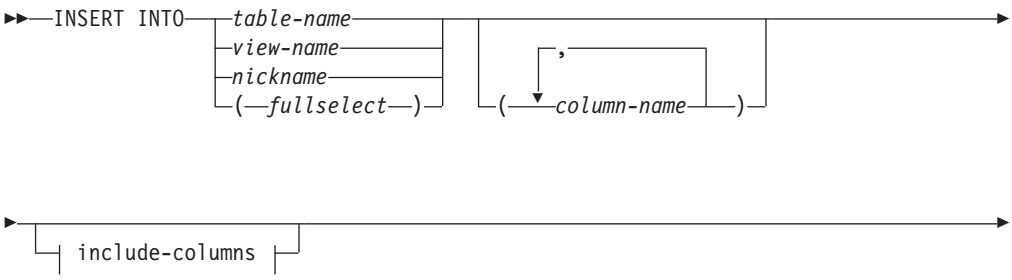
In addition, for each table, view or nickname referenced in any fullselect used in the INSERT statement, the privileges held by the authorization ID of the statement must include at least one of the following:

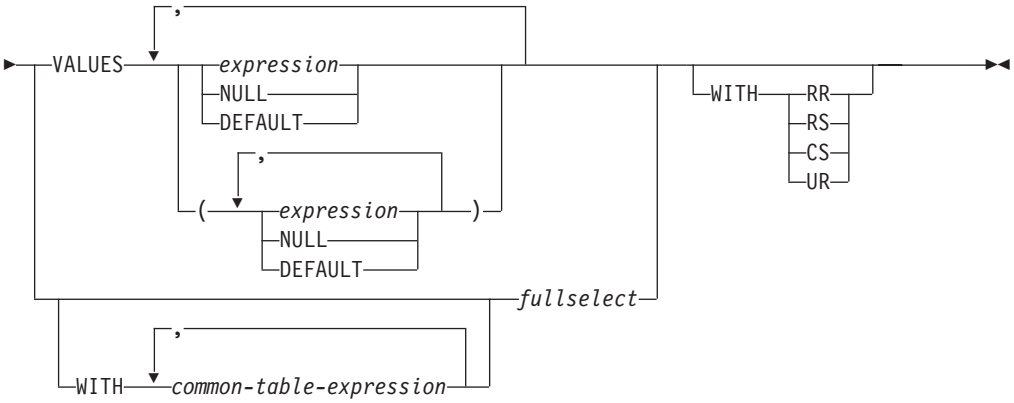
- SELECT privilege
- CONTROL privilege
- SYSADM or DBADM authority.

GROUP privileges are not checked for static INSERT statements.

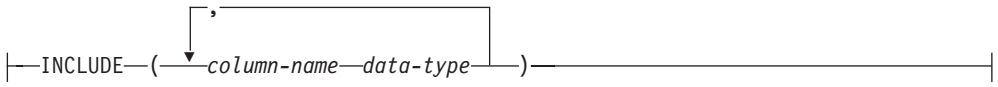
If the target of the insert operation is a nickname, the privileges on the object at the data source are not considered until the statement is executed at the data source. At this time, the authorization ID that is used to connect to the data source must have the privileges required for the operation on the object at the data source. The authorization ID of the statement may be mapped to a different authorization ID at the data source.

Syntax:





include-columns:



Description:

INTO *table-name*, *view-name*, *nickname*, **or** *(fullselect)*
Identifies the object of the insert operation. The name must identify a table, view or nickname that exists at the application server, but it must not identify a catalog table, a system-maintained materialized query table, a view of a catalog table, or a read-only view, unless an INSTEAD OF trigger is defined for the insert operation on the subject view. Rows inserted into a nickname are placed in the data source object to which the nickname refers.

If the object of the insert operation is a fullselect, the fullselect must be insertable, as defined in the "Insertable views" Notes item in the description of the CREATE VIEW statement.

If no INSTEAD OF trigger exists for the insert operation on this view, a value cannot be inserted into a view column that is derived from:

- A constant, expression, or scalar function
- The same base table column as some other column of the view

If the object of the insert operation is a view with such columns, a list of column names must be specified, and the list must not identify these columns.

A row can be inserted into a view or a fullselect that is defined using a UNION ALL if the row satisfies the check constraints of exactly one of the underlying base tables. If a row satisfies the check constraints of more than one table, or no table at all, an error is returned (SQLSTATE 23513).

(*column-name***,...)**
Specifies the columns for which insert values are provided. Each name must be an unqualified name that identifies a column of the table or view, or a column in the fullselect. The same column must not be identified more than once. A column that cannot accept inserted values (for example, a column based on an expression) must not be identified.

Omission of the column list is an implicit specification of a list in which every column of the table or view, or every item in the select-list of the fullselect is

identified in left-to-right order. This list is established when the statement is prepared and, therefore, does not include columns that were added to a table after the statement was prepared.

include-columns
Specifies a set of columns that are included, along with the columns of *table-name* or *view-name*, in the intermediate result table of the INSERT statement when it is nested in the FROM clause of a fullselect. The *include-columns* are appended at the end of the list of columns that are specified for *table-name* or *view-name*.

INCLUDE
Specifies a list of columns to be included in the intermediate result table of the INSERT statement. This clause can only be specified if the INSERT statement is nested in the FROM clause of a fullselect.

column-name
Specifies a column of the intermediate result table of the INSERT statement. The name cannot be the same as the name of another include column or a column in *table-name* or *view-name* (SQLSTATE 42711).

data-type
Specifies the data type of the include column. The data type must be one that is supported by the CREATE TABLE statement.

VALUES
Introduces one or more rows of values to be inserted.

Each host variable named must be described in the program in accordance with the rules for declaring host variables.

The number of values for each row must equal the number of names in the implicit or explicit column list and the columns identified in the INCLUDE clause. The first value is inserted in the first column in the list, the second value in the second column, and so on.

expression
An *expression* can be any expression defined in "Expressions".

NULL
Specifies the null value and should only be specified for nullable columns.

DEFAULT
Specifies that the default value is to be used. The result of specifying DEFAULT depends on how the column was defined, as follows:

- If the column was defined as a generated column based on an expression, the column value is generated by the system, based on that expression.
- If the IDENTITY clause is used, the value is generated by the database manager.
- If the WITH DEFAULT clause is used, the value inserted is as defined for the column (see *default-clause* in "CREATE TABLE").
- If the NOT NULL clause is used and the GENERATED clause is not used, or the WITH DEFAULT clause is not used or DEFAULT NULL is used, the DEFAULT keyword cannot be specified for that column (SQLSTATE 23502).
- When inserting into a nickname, the DEFAULT keyword will be passed through the INSERT statement to the data source only if the data source supports the DEFAULT keyword in its query language syntax.

WITH *common-table-expression*
Defines a common table expression for use with the fullselect that follows.

fullselect
Specifies a set of new rows in the form of the result table of a fullselect. There may be one, more than one, or none. If the result table is empty, SQLCODE is set to +100 and SQLSTATE is set to '02000'.

When the base object of the INSERT and the base object of the fullselect or any subquery of the fullselect, are the same table, the fullselect is completely evaluated before any rows are inserted.

The number of columns in the result table must equal the number of names in the column list. The value of the first column of the result is inserted in the first column in the list, the second value in the second column, and so on.

WITH
Specifies the isolation level at which the fullselect is executed.

RR
Repeatable Read

RS
Read Stability

CS
Cursor Stability

UR
Uncommitted Read

The default isolation level of the statement is the isolation level of the package in which the statement is bound.

Rules:

- ***Triggers:*** INSERT statements may cause triggers to be executed. A trigger may cause other statements to be executed, or may raise error conditions based on the inserted values. If an insert operation into a view causes an INSTEAD OF trigger to fire, validity, referential integrity, and constraints will be checked against the updates that are performed in the trigger, and not against the view that caused the trigger to fire, or its underlying tables.
- ***Default values:*** The value inserted in any column that is not in the column list is either the default value of the column or null. Columns that do not allow null values and are not defined with NOT NULL WITH DEFAULT must be included in the column list. Similarly, if you insert into a view, the value inserted into any column of the base table that is not in the view is either the default value of the column or null. Hence, all columns of the base table that are not in the view must have either a default value or allow null values. The only value that can be inserted into a generated column defined with the GENERATED ALWAYS clause is DEFAULT (SQLSTATE 428C9).
- ***Length:*** If the insert value of a column is a number, the column must be a numeric column with the capacity to represent the integral part of the number. If the insert value of a column is a string, the column must either be a string column with a length attribute at least as great as the length of the string, or a datetime column if the string represents a date, time, or timestamp.
- ***Assignment:*** Insert values are assigned to columns in accordance with specific assignment rules.
- ***Validity:*** If the table named, or the base table of the view named, has one or more unique indexes, each row inserted into the table must conform to the

constraints imposed by those indexes. If a view whose definition includes WITH CHECK OPTION is named, each row inserted into the view must conform to the definition of the view. For an explanation of the rules governing this situation, see "CREATE VIEW".

- ***Referential Integrity:*** For each constraint defined on a table, each non-null insert value of the foreign key must be equal to a primary key value of the parent table.
- ***Check Constraint:*** Insert values must satisfy the check conditions of the check constraints defined on the table. An INSERT to a table with check constraints defined has the constraint conditions evaluated once for each row that is inserted.
- ***Datalinks:*** Insert statements that include DATALINK values will result in an attempt to link the file if a URL value is included (not empty string or blanks) and the column is defined with FILE LINK CONTROL. Errors in the DATALINK value or in linking the file will cause the insert to fail (SQLSTATE 428D1 or 57050).

Notes:

- After execution of an INSERT statement, the value of the third variable of the SQLERRD(3) portion of the SQLCA indicates the number of rows that were passed to the insert operation. In the context of an SQL procedure statement, the value can be retrieved using the ROW_COUNT variable of the GET DIAGNOSTICS statement. SQLERRD(5) contains the count of all triggered insert, update and delete operations.
- Unless appropriate locks already exist, one or more exclusive locks are acquired at the execution of a successful INSERT statement. Until the locks are released, an inserted row can only be accessed by:
  - The application process that performed the insert.
  - Another application process using isolation level UR through a read-only cursor, SELECT INTO statement, or subselect used in a subquery.
- For further information about locking, see the description of the COMMIT, ROLLBACK, and LOCK TABLE statements.
- If an application is running against a partitioned database, and it is bound with option INSERT BUF, then INSERT with VALUES statements which are not processed using EXECUTE IMMEDIATE may be buffered. DB2 assumes that such an INSERT statement is being processed inside a loop in the application's logic. Rather than execute the statement to completion, it attempts to buffer the new row values in one or more buffers. As a result the actual insertions of the rows into the table are performed later, asynchronous with the application's INSERT logic. Be aware that this asynchronous insertion may cause an error related to an INSERT to be returned on some other SQL statement that follows the INSERT in the application.

  This has the potential to dramatically improve INSERT performance, but is best used with clean data, due to the asynchronous nature of the error handling.
- When a row is inserted into a table that has an identity column, DB2 generates a value for the identity column.
  - For a GENERATED ALWAYS identity column, DB2 always generates the value.
  - For a GENERATED BY DEFAULT column, if a value is not explicitly specified (with a VALUES clause, or subselect), DB2 generates a value.

  The first value generated by DB2 is the value of the START WITH specification for the identity column.

- When a value is inserted for a user-defined distinct type identity column, the entire computation is done in the source type, and the result is cast to the distinct type before the value is actually assigned to the column. (There is no casting of the previous value to the source type prior to the computation.)
- When inserting into a GENERATED ALWAYS identity column, DB2 will always generate a value for the column, and users must not specify a value at insertion time. If a GENERATED ALWAYS identity column is listed in the column-list of the INSERT statement, with a non-DEFAULT value in the VALUES clause, an error occurs (SQLSTATE 428C9).

  For example, assuming that EMPID is defined as an identity column that is GENERATED ALWAYS, then the command:

  ```
  INSERT INTO T2 (EMPID, EMPNAME, EMPADDR)
    VALUES (:hv_valid_emp_id, :hv_name, :hv_addr)
  ```

  will result in an error.
- When inserting into a GENERATED BY DEFAULT column, DB2 will allow an actual value for the column to be specified within the VALUES clause, or from a subselect. However, when a value is specified in the VALUES clause, DB2 does not perform any verification of the value. In order to guarantee uniqueness of the values, a unique index on the identity column must be created.

  When inserting into a table with a GENERATED BY DEFAULT identity column, without specifying a column list, the VALUES clause can specify the DEFAULT keyword to represent the value for the identity column. DB2 will generate the value for the identity column.

  ```
  INSERT INTO T2 (EMPID, EMPNAME, EMPADDR)
    VALUES (DEFAULT, :hv_name, :hv_addr)
  ```

  In this example, EMPID is defined as an identity column, and thus the value inserted into this column is generated by DB2.
- The rules for inserting into an identity column with a subselect are similar to those for an insert with a VALUES clause. A value for an identity column may only be specified if the identity column is defined as GENERATED BY DEFAULT.

  For example, assume T1 and T2 are tables with the same definition, both containing columns *intcol1* and *identcol2* (both are type INTEGER and the second column has the identity attribute). Consider the following insert:

  ```
  INSERT INTO T2
    SELECT *
    FROM T1
  ```

  This example is logically equivalent to:

  ```
  INSERT INTO T2 (intcol1,identcol2)
    SELECT intcol1, identcol2
    FROM T1
  ```

  In both cases, the INSERT statement is providing an explicit value for the identity column of T2. This explicit specification can be given a value for the identity column, but the identity column in T2 must be defined as GENERATED BY DEFAULT. Otherwise, an error will result (SQLSTATE 428C9).

  If there is a table with a column defined as a GENERATED ALWAYS identity, it is still possible to propagate all other columns from a table with the same definition. For example, given the example tables T1 and T2 described above, the intcol1 values from T1 to T2 can be propagated with the following SQL:

```
INSERT INTO T2 (intcol1)
  SELECT intcol1
  FROM T1
```

Note that, because identcol2 is not specified in the column-list, it will be filled in with its default (generated) value.

- When inserting a row into a single column table where the column is defined as a GENERATED ALWAYS identity column, it is possible to specify a VALUES clause with the DEFAULT keyword. In this case, the application does not provide any value for the table, and DB2 generates the value for the identity column.

  ```
  INSERT INTO IDTABLE
    VALUES(DEFAULT)
  ```

  Assuming the same single column table for which the column has the identity attribute, to insert multiple rows with a single INSERT statement, the following INSERT statement could be used:

  ```
  INSERT INTO IDTABLE
    VALUES (DEFAULT), (DEFAULT), (DEFAULT), (DEFAULT)
  ```

- When DB2 generates a value for an identity column, that generated value is consumed; the next time that a value is needed, DB2 will generate a new value. This is true even when an INSERT statement involving an identity column fails or is rolled back.

  For example, assume that a unique index has been created on the identity column. If a duplicate key violation is detected in generating a value for an identity column, an error occurs (SQLSTATE 23505) and the value generated for the identity column is considered to be consumed. This can occur when the identity column is defined as GENERATED BY DEFAULT and the system tries to generate a new value, but the user has explicitly specified values for the identity column in previous INSERT statements. Reissuing the same INSERT statement in this case can lead to success. DB2 will generate the next value for the identity column, and it is possible that this next value will be unique, and that this INSERT statement will be successful.

- If the maximum value for the identity column is exceeded (or minimum value for a descending sequence) in generating a value for an identity column, an error occurs (SQLSTATE 23522). In this situation, the user would have to DROP and CREATE a new table with an identity column having a larger range (that is, change the data type or increment value for the column to allow for a larger range of values).

  For example, an identity column may have been defined with a data type of SMALLINT, and eventually the column runs out of assignable values. To redefine the identity column as INTEGER, the data would need to be unloaded, the table would have to be dropped and recreated with a new definition for the column, and then the data would be reloaded. When the table is redefined, it needs to specify a START WITH value for the identity column such that the next value generated by DB2 will be the next value in the original sequence. To determine the end value, issue a query using MAX of the identity column (for an ascending sequence), or MIN of the identity column (for a descending sequence), before unloading the data.

Examples:

Example 1: Insert a new department with the following specifications into the DEPARTMENT table:

- Department number (DEPTNO) is 'E31'

- Department name (DEPTNAME) is 'ARCHITECTURE'
- Managed by (MGRNO) a person with number '00390'
- Reports to (ADMRDEPT) department 'E01'.

```
INSERT INTO DEPARTMENT
  VALUES ('E31', 'ARCHITECTURE', '00390', 'E01')
```

Example 2: Insert a new department into the DEPARTMENT table as in example 1, but do not assign a manager to the new department.

```
INSERT INTO DEPARTMENT (DEPTNO, DEPTNAME, ADMRDEPT )
  VALUES ('E31', 'ARCHITECTURE', 'E01')
```

Example 3: Insert two new departments using one statement into the DEPARTMENT table as in example 2, but do not assign a manager to the new department.

```
INSERT INTO DEPARTMENT (DEPTNO, DEPTNAME, ADMRDEPT)
  VALUES ('B11', 'PURCHASING', 'B01'),
         ('E41', 'DATABASE ADMINISTRATION', 'E01')
```

Example 4: Create a temporary table MA_EMP_ACT with the same columns as the EMP_ACT table. Load MA_EMP_ACT with the rows from the EMP_ACT table with a project number (PROJNO) starting with the letters 'MA'.

```
CREATE TABLE MA_EMP_ACT
         ( EMPNO CHAR(6)  NOT NULL,
          PROJNO CHAR(6)  NOT NULL,
          ACTNO SMALLINT  NOT NULL,
          EMPTIME DEC(5,2),
          EMSTDATE DATE,
          EMENDATE  DATE )
INSERT INTO MA_EMP_ACT
  SELECT * FROM EMP_ACT
    WHERE SUBSTR(PROJNO, 1, 2) = 'MA'
```

Example 5: Use a C program statement to add a skeleton project to the PROJECT table. Obtain the project number (PROJNO), project name (PROJNAME), department number (DEPTNO), and responsible employee (RESPEMP) from host variables. Use the current date as the project start date (PRSTDATE). Assign a NULL value to the remaining columns in the table.

```
EXEC SQL INSERT INTO PROJECT (PROJNO, PROJNAME, DEPTNO, RESPEMP, PRSTDATE)
  VALUES (:PRJNO, :PRJNM, :DPTNO, :REMP, CURRENT DATE);
```

Example 6: Specify an INSERT statement as the *data-change-table-reference* within a SELECT statement. Define an extra include column whose values are specified in the VALUES clause, which is then used as an ordering column for the inserted rows.

```
SELECT inorder.ordernum
  FROM (INSERT INTO orders(custno)INCLUDE (insertnum integer)
    VALUES(:cnum1, 1), (:cnum2, 2)) InsertedOrders
  ORDER BY insertnum;
```

Related reference:

- "Expressions" in the *SQL Reference, Volume 1*
- "CREATE TABLE" on page 591
- "CREATE VIEW" on page 656
- "SQL queries" in the *SQL Reference, Volume 1*
- "Assignments and comparisons" in the *SQL Reference, Volume 1*

Related samples:

- "dtlob.sqc -- How to use the LOB data type (C)"
- "tbident.sqc -- How to use identity columns (C)"
- "tbmod.sqc -- How to modify table data (C)"
- "tbtrig.sqc -- How to use a trigger on a table (C)"
- "dtlob.sqC -- How to use the LOB data type (C++)"
- "tbmod.sqC -- How to modify table data (C++)"
- "tbtrig.sqC -- How to use a trigger on a table (C++)"
- "DtLob.java -- How to use LOB data type (JDBC)"
- "TbIdent.java -- How to use Identity Columns (JDBC)"
- "TbMod.java -- How to modify table data (JDBC)"
- "TbTrig.java -- How to use triggers (JDBC)"
- "TbIdent.sqlj -- How to use Identity Columns (SQLj)"
- "TbMod.sqlj -- How to modify table data (SQLj)"
- "TbTrig.sqlj -- How to use triggers (SQLj)"
- "updat.sqb -- How to update, delete and insert table data (MF COBOL)"

REVOKE (Database Authorities)

This form of the REVOKE statement revokes authorities that apply to the entire database.

Invocation:

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

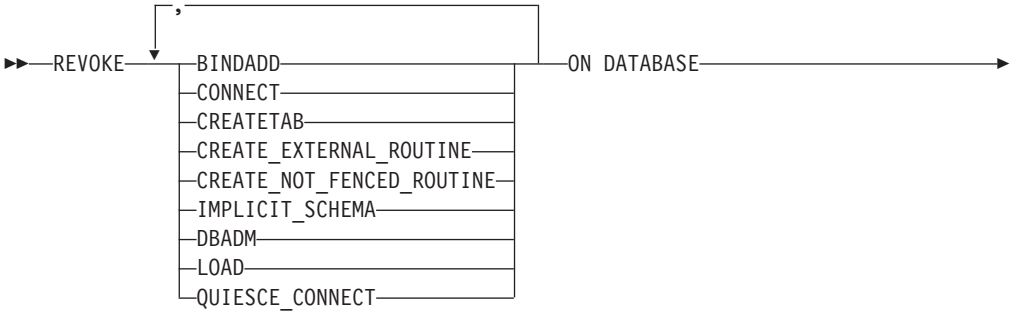
Authorization:

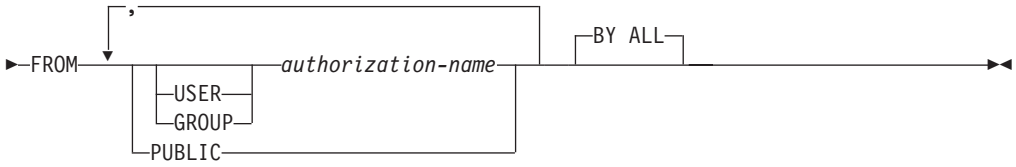
The privileges held by the authorization ID of the statement must include at least one of the following:

- DBADM authority
- SYSADM authority

To revoke DBADM authority, SYSADM authority is required.

Syntax:





Description:

BINDADD
Revokes the authority to create packages. The creator of a package automatically has the CONTROL privilege on that package and retains this privilege even if his BINDADD authority is subsequently revoked.

The BINDADD authority cannot be revoked from an *authorization-name* holding DBADM authority without also revoking the DBADM authority.

CONNECT
Revokes the authority to access the database.

Revoking the CONNECT authority from a user does not affect any privileges that were granted to that user on objects in the database. If the user is subsequently granted the CONNECT authority again, all previously held privileges are still valid (assuming they were not explicitly revoked).

The CONNECT authority cannot be revoked from an *authorization-name* holding DBADM authority without also revoking the DBADM authority (SQLSTATE 42504).

CREATETAB
Revokes the authority to create tables. The creator of a table automatically has the CONTROL privilege on that table, and retains this privilege even if his CREATETAB authority is subsequently revoked.

The CREATETAB authority cannot be revoked from an *authorization-name* holding DBADM authority without also revoking the DBADM authority (SQLSTATE 42504).

CREATE_EXTERNAL_ROUTINE
Revokes the authority to register external routines. Once an external routine has been registered, it continues to exist, even if CREATE_EXTERNAL_ROUTINE is subsequently revoked from the authorization ID that registered the routine.

CREATE_EXTERNAL_ROUTINE authority cannot be revoked from an *authorization-name* holding DBADM or CREATE_NOT_FENCED_ROUTINE authority without also revoking DBADM or CREATE_NOT_FENCED_ROUTINE authority (SQLSTATE 42504).

CREATE_NOT_FENCED_ROUTINE
Revokes the authority to register routines that execute in the database manager's process. Once a routine has been registered as not fenced, it continues to run in this manner, even if CREATE_NOT_FENCED_ROUTINE is subsequently revoked from the authorization ID that registered the routine.

CREATE_NOT_FENCED_ROUTINE authority cannot be revoked from an *authorization-name* holding DBADM authority without also revoking the DBADM authority (SQLSTATE 42504).

IMPLICIT_SCHEMA
Revokes the authority to implicitly create a schema. It does not affect the ability to create objects in existing schemas or to process a CREATE SCHEMA statement.

DBADM
Revokes the DBADM authority.

DBADM authority cannot be revoked from PUBLIC (because it cannot be granted to PUBLIC).

CAUTION:
Revoking DBADM authority does not automatically revoke any privileges that were held by the *authorization-name* **on objects in the database, nor does it revoke any of the other database authorities that were implicitly and automatically granted when DBADM authority was originally granted.**

LOAD
Revokes the authority to LOAD in this database.

QUIESCE_CONNECT
Revokes the authority to access the database while it is quiesced.

FROM
Indicates from whom the authorities are revoked.

USER
Specifies that the *authorization-name* identifies a user.

GROUP
Specifies that the *authorization-name* identifies a group name.

authorization-name,...
Lists one or more authorization IDs.

The authorization ID of the REVOKE statement itself cannot be used (SQLSTATE 42502). It is not possible to revoke the authorities from an *authorization-name* that is the same as the authorization ID of the REVOKE statement.

PUBLIC
Revokes the authorities from PUBLIC.

BY ALL
Revokes each named privilege from all named users who were explicitly granted those privileges, regardless of who granted them. This is the default behavior.

Rules:

- If neither USER nor GROUP is specified, then:
  - If all rows for the grantee in the SYSCAT.DBAUTH catalog view have a GRANTEETYPE of U, then USER will be assumed.
  - If all rows have a GRANTEETYPE of G, then GROUP will be assumed.
  - If some rows have U and some rows have G, then an error (SQLSTATE 56092) is raised.

Notes:

- ***Compatibilities***

For compatibility with versions earlier than Version 8, the option CREATE_NOT_FENCED can be substituted for CREATE_NOT_FENCED_ROUTINE.

- Revoking a specific privilege does not necessarily revoke the ability to perform an action. A user may proceed with a task if other privileges are held by PUBLIC or a group, or if the user has a higher level authority, such as DBADM.

Examples:

Example 1: Given that USER6 is only a user and not a group, revoke the privilege to create tables from the user USER6.

```
REVOKE CREATETAB ON DATABASE FROM USER6
```

Example 2: Revoke BINDADD authority on the database from a group named D024. There are two rows in the SYSCAT.DBAUTH catalog view for this grantee; one with a GRANTEETYPE of U and one with a GRANTEETYPE of G.

```
REVOKE BINDADD ON DATABASE FROM GROUP D024
```

In this case, the GROUP keyword must be specified; otherwise an error will occur (SQLSTATE 56092).

Related reference:

- "REVOKE (Index Privileges)" on page 738
- "REVOKE (Package Privileges)" on page 740
- "REVOKE (Schema Privileges)" on page 745
- "REVOKE (Table, View, or Nickname Privileges)" on page 752
- "REVOKE (Server Privileges)" on page 749
- "REVOKE (Table Space Privileges)" on page 750
- "REVOKE (Routine Privileges)" on page 742

Related samples:

- "dbauth.sqc -- How to grant, display, and revoke authorities at database level (C)"
- "dbauth.sqC -- How to grant, display, and revoke authorities at database level (C++)"
- "DbAuth.java -- Grant, display or revoke privileges on database (JDBC)"
- "DbAuth.sqlj -- Grant, display or revoke privileges on database (SQLj)"

RENAME

The RENAME statement renames an existing table or index.

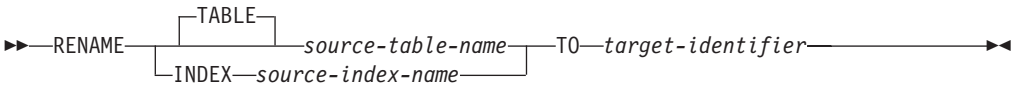
Invocation:

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization:

The privileges held by the authorization ID of the statement must include either SYSADM or DBADM authority, CONTROL privilege on the table or index, or ALTERIN privilege on the schema.

Syntax:



Description:

TABLE *source-table-name*
Names the existing table that is to be renamed. The name, including the schema name, must identify a table that already exists in the database (SQLSTATE 42704). It must not be the name of a catalog table (SQLSTATE 42832), a materialized query table, a typed table (SQLSTATE 42997), a declared global temporary table (SQLSTATE 42995), a nickname, or an object other than a table or an alias (SQLSTATE 42809). The TABLE keyword is optional.

INDEX *source-index-name*
Names the existing index that is to be renamed. The name, including the schema name, must identify an index that already exists in the database (SQLSTATE 42704). It must not be the name of an index on a declared global temporary table (SQLSTATE 42995). The schema name must not be SYSIBM, SYSCAT, SYSFUN, or SYSSTAT (SQLSTATE 42832).

target-identifier
Specifies the new name for the table or index without a schema name. The schema name of the source object is used to qualify the new name for the object. The qualified name must *not* identify a table, view, alias, or index that already exists in the database (SQLSTATE 42710).

Rules:

When renaming a table, the source table must not:

- Be referenced in any existing view definitions or materialized query table definitions
- Be referenced in any triggered SQL statements in existing triggers or be the subject table of an existing trigger
- Be referenced in an SQL function
- Have any check constraints
- Have any generated columns other than the identity column
- Be a parent or dependent table in any referential integrity constraints
- Be the scope of any existing reference column.

An error (SQLSTATE 42986) is returned if the source table violates one or more of these conditions.

When renaming an index:

- The source index must not be a system-generated index for an implementation table on which a typed table is based (SQLSTATE 42858).

Notes:

- Catalog entries are updated to reflect the new table or index name.

- *All* authorizations associated with the source table or index name are *transferred* to the new table or index name (the authorization catalog tables are updated appropriately).
- Indexes defined over the source table are *transferred* to the new table (the index catalog tables are updated appropriately).
- RENAME TABLE invalidates any packages that are dependent on the source table. RENAME INDEX invalidates any packages that are dependent on the source index.
- If an alias is used for the *source-table-name*, it must resolve to a table name. The table is renamed within the schema of this table. The alias is not changed by the RENAME statement and continues to refer to the old table name.
- A table with primary key or unique constraints can be renamed if none of the primary key or unique constraints are referenced by any foreign key.

Examples:

Change the name of the EMP table to EMPLOYEE.

```
RENAME TABLE EMP TO EMPLOYEE
RENAME TABLE ABC.EMP TO EMPLOYEE
```

Change the name of the index NEW-IND to IND.

```
RENAME INDEX NEW-IND TO IND
RENAME INDEX ABC.NEW-IND TO IND
```

REVOKE (Index Privileges)

This form of the REVOKE statement revokes the CONTROL privilege on an index.

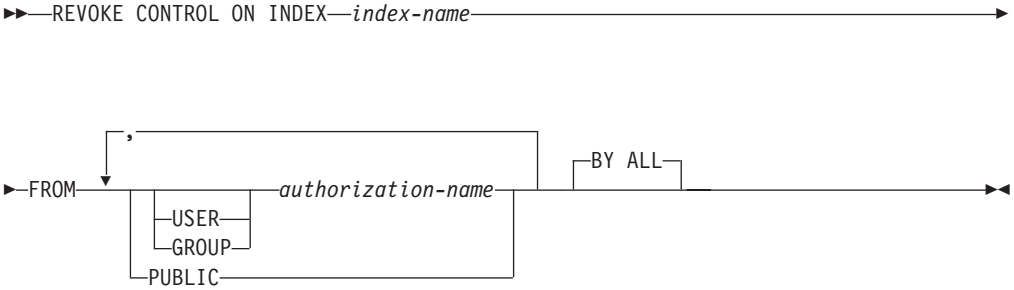
Invocation:

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization:

The authorization ID of the statement must hold either SYSADM or DBADM authority (SQLSTATE 42501).

Syntax:



Description:

CONTROL
Revokes the privilege to drop the index. This is the CONTROL privilege for indexes, which is automatically granted to creators of indexes.

ON INDEX *index-name*
Specifies the name of the index on which the CONTROL privilege is to be revoked.

FROM
Indicates from whom the privileges are revoked.

USER
Specifies that the *authorization-name* identifies a user.

GROUP
Specifies that the *authorization-name* identifies a group name.

authorization-name,...
Lists one or more authorization IDs.

The authorization ID of the REVOKE statement itself cannot be used (SQLSTATE 42502). It is not possible to revoke the privileges from an *authorization-name* that is the same as the authorization ID of the REVOKE statement.

PUBLIC
Revokes the privileges from PUBLIC.

BY ALL
Revokes the privilege from all named users who were explicitly granted that privilege, regardless of who granted it. This is the default behavior.

Rules:

- If neither USER nor GROUP is specified, then:
  - If all rows for the grantee in the SYSCAT.INDEXAUTH catalog view have a GRANTEETYPE of U, then USER will be assumed.
  - If all rows have a GRANTEETYPE of G, then GROUP will be assumed.
  - If some rows have U and some rows have G, then an error (SQLSTATE 56092) is raised.

Notes:

- Revoking a specific privilege does not necessarily revoke the ability to perform the action. A user may proceed with their task if other privileges are held by PUBLIC or a group, or if they have authorities such as ALTERIN on the schema of an index.

Examples:

Example 1: Given that USER4 is only a user and not a group, revoke the privilege to drop an index DEPTIDX from the user USER4.

```
REVOKE CONTROL ON INDEX DEPTIDX FROM USER4
```

Example 2: Revoke the privilege to drop an index LUNCHITEMS from the user CHEF and the group WAITERS.

```
REVOKE CONTROL ON INDEX LUNCHITEMS
  FROM USER CHEF, GROUP WAITERS
```

Related reference:

- "REVOKE (Database Authorities)" on page 733
- "REVOKE (Package Privileges)" on page 740
- "REVOKE (Schema Privileges)" on page 745
- "REVOKE (Table, View, or Nickname Privileges)" on page 752
- "REVOKE (Server Privileges)" on page 749
- "REVOKE (Table Space Privileges)" on page 750
- "REVOKE (Routine Privileges)" on page 742

REVOKE (Package Privileges)

This form of the REVOKE statement revokes CONTROL, BIND, and EXECUTE privileges against a package.

Invocation:

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

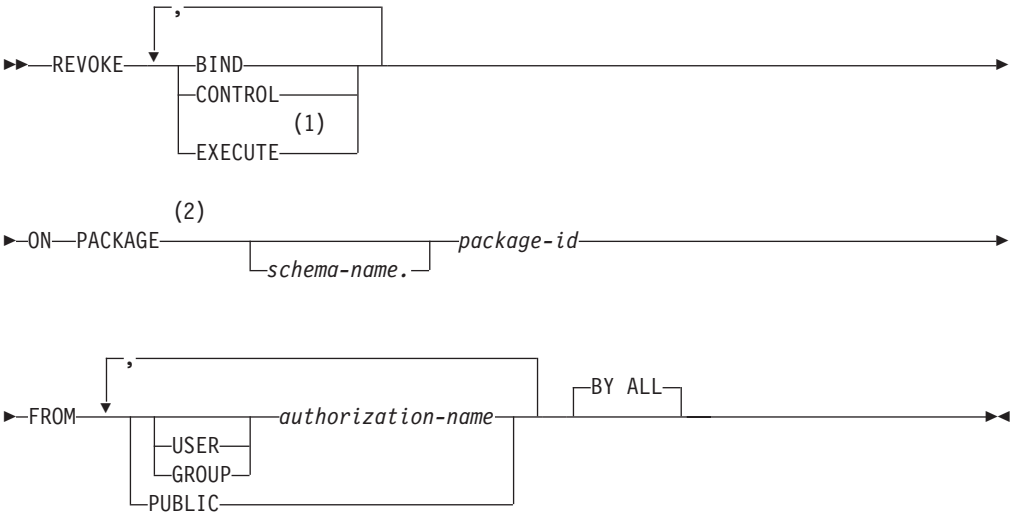
Authorization:

The privileges held by the authorization ID of the statement must include at least one of the following:

- CONTROL privilege on the referenced package
- SYSADM or DBADM authority.

To revoke the CONTROL privilege, SYSADM or DBADM authority are required.

Syntax:



Notes:

1 RUN can be used as a synonym for EXECUTE.

2 PROGRAM can be used as a synonym for PACKAGE.

Description:

BIND
Revokes the privilege to execute BIND or REBIND on—or to add a new version of— the referenced package.

The BIND privilege cannot be revoked from an *authorization-name* that holds CONTROL privilege on the package, without also revoking the CONTROL privilege.

CONTROL
Revokes the privilege to drop the package and to extend package privileges to other users.

Revoking CONTROL does not revoke the other package privileges.

EXECUTE
Revokes the privilege to execute the package.

The EXECUTE privilege cannot be revoked from an *authorization-name* that holds CONTROL privilege on the package without also revoking the CONTROL privilege.

ON PACKAGE *schema-name.package-id*
Specifies the name of the package on which privileges are to be revoked. If a schema name is not specified, the package ID is implicitly qualified by the default schema. The revoking of a package privilege applies to all versions of the package.

FROM
Indicates from whom the privileges are revoked.

USER
Specifies that the *authorization-name* identifies a user.

GROUP
Specifies that the *authorization-name* identifies a group name.

authorization-name,...
Lists one or more authorization IDs.

The authorization ID of the REVOKE statement itself cannot be used (SQLSTATE 42502). It is not possible to revoke the privileges from an *authorization-name* that is the same as the authorization ID of the REVOKE statement.

PUBLIC
Revokes the privileges from PUBLIC.

BY ALL
Revokes each named privilege from all named users who were explicitly granted those privileges, regardless of who granted them. This is the default behavior.

Rules:

- If neither USER nor GROUP is specified, then:
  - If all rows for the grantee in the SYSCAT.PACKAGEAUTH catalog view have a GRANTEETYPE of U, then USER will be assumed.
  - If all rows have a GRANTEETYPE of G, then GROUP will be assumed.
  - If some rows have U and some rows have G, then an error (SQLSTATE 56092) is raised.

Notes:

- Revoking a specific privilege does not necessarily revoke the ability to perform the action. A user may proceed with their task if other privileges are held by PUBLIC or a group, or if they have privileges such as ALTERIN on the schema of a package.

Examples:

Example 1: Revoke the EXECUTE privilege on package CORPDATA.PKGA from PUBLIC.

```
REVOKE EXECUTE
  ON PACKAGE CORPDATA.PKGA
  FROM PUBLIC
```

Example 2: Revoke CONTROL authority on the RRSP_PKG package for the user FRANK and for PUBLIC.

```
REVOKE CONTROL
  ON PACKAGE RRSP_PKG
  FROM USER FRANK, PUBLIC
```

Related reference:

- "REVOKE (Database Authorities)" on page 733
- "REVOKE (Index Privileges)" on page 738
- "REVOKE (Schema Privileges)" on page 745
- "REVOKE (Table, View, or Nickname Privileges)" on page 752
- "REVOKE (Server Privileges)" on page 749
- "REVOKE (Table Space Privileges)" on page 750
- "REVOKE (Routine Privileges)" on page 742

REVOKE (Routine Privileges)

This form of the REVOKE statement revokes privileges on a routine (function, method, or procedure).

Invocation:

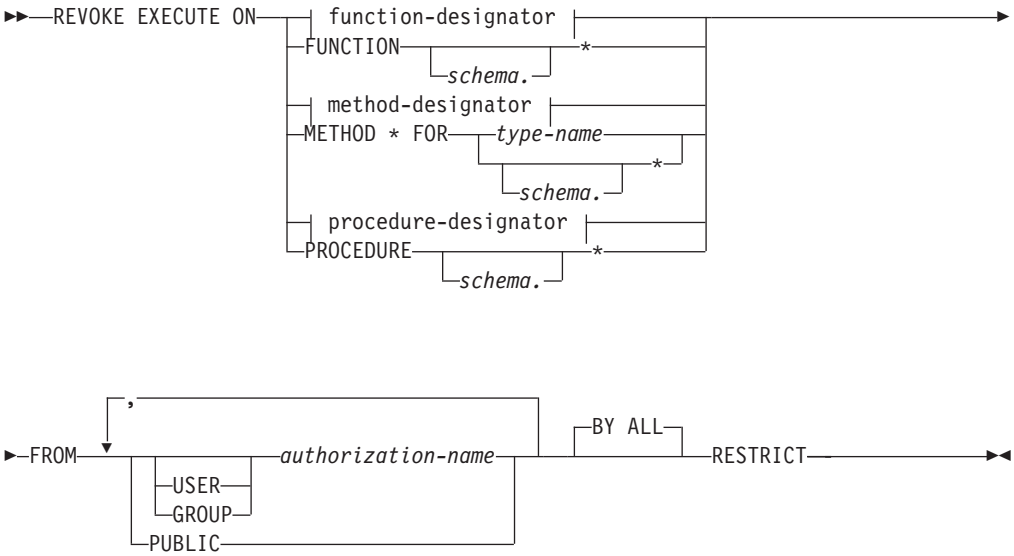
This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization:

The privileges held by the authorization ID of the statement must include at least one of the following:

- SYSADM or DBADM authority

Syntax:



Description:

EXECUTE
Revokes the privilege to run the identified user-defined function, method, or stored procedure.

function-designator
Uniquely identifies the function.

FUNCTION *schema.**
Identifies the explicit grant for all the existing and future functions in the schema. Revoking the *schema*.* privilege does not revoke any privileges that were granted on a specific function. In dynamic SQL statements, if a schema is not specified, the schema in the CURRENT SCHEMA special register will be used. In static SQL statements, if a schema is not specified, the schema in the QUALIFIER precompile/bind option will be used.

method-designator
Uniquely identifies the method.

METHOD *
Identifies the explicit grant for all the existing and future methods for the type *type-name*. Revoking the * privilege does not revoke any privileges that were granted on a specific method.

FOR *type-name*
Names the type in which the specified method is found. The name must identify a type already described in the catalog (SQLSTATE 42704). In dynamic SQL statements, the value of the CURRENT SCHEMA special register is used as a qualifier for an unqualified type name. In static SQL statements, the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified type names. An asterisk (*) can be used in place of *type-name* to identify the explicit grant on all existing and future methods for all existing and future types in the schema. Revoking the privilege using an asterisk for method and *type-name* does not revoke any privileges that were granted on a specific method or on all methods for a specific type.

procedure-designator
Uniquely identifies the procedure.

PROCEDURE *schema.**
Identifies the explicit grant for all the existing and future procedures in the schema. Revoking the *schema*.* privilege does not revoke any privileges that were granted on a specific procedure. In dynamic SQL statements, if a schema is not specified, the schema in the CURRENT SCHEMA special register will be used. In static SQL statements, if a schema is not specified, the schema in the QUALIFIER precompile/bind option will be used.

FROM
Specifies from whom the EXECUTE privilege is revoked.

USER
Specifies that the *authorization-name* identifies a user.

GROUP
Specifies that the *authorization-name* identifies a group name.

authorization-name,...
Lists the authorization IDs of one or more users or groups. The list of authorization IDs cannot include the authorization ID of the user issuing the statement (SQLSTATE 42502).

PUBLIC
Revokes the EXECUTE privilege from all users.

BY ALL
Revokes the EXECUTE privilege from all named users who were explicitly granted the privilege, regardless of who granted it. This is the default behavior.

RESTRICT
Specifies that the EXECUTE privilege cannot be revoked if both of the following are true (SQLSTATE 42893):

- The specified routine is used in a view, trigger, constraint, index extension, SQL function, SQL method, transform group, or is referenced as the SOURCE of a sourced function.
- The loss of the EXECUTE privilege would cause the definer of the view, trigger, constraint, index extension, SQL function, SQL method, transform group, or sourced function to no longer be able to execute the specified routine.

Rules:

- It is not possible to revoke the EXECUTE privilege on a function or method defined with schema 'SYSIBM' or 'SYSFUN' (SQLSTATE 42832).
- If neither USER nor GROUP is specified, then:
  - If all rows for the grantee in the SYSCAT.ROUTINEAUTH catalog views have a GRANTEETYPE of U, then USER is assumed.
  - If all rows have a GRANTEETYPE of G, then GROUP is assumed.
  - If some rows have U and some rows have G, an error (SQLSTATE 56092) is raised.

Examples:

Example 1: Revoke the EXECUTE privilege on function CALC_SALARY from user JONES. Assume that there is only one function in the schema with function name CALC_SALARY.

```
REVOKE EXECUTE ON FUNCTION CALC_SALARY FROM JONES RESTRICT
```

Example 2: Revoke the EXECUTE privilege on procedure VACATION_ACCR from all users at the current server.

```
REVOKE EXECUTE ON PROCEDURE VACATION_ACCR FROM PUBLIC RESTRICT
```

Example 3: Revoke the EXECUTE privilege on function NEW_DEPT_HIRES from HR (Human Resources). The function has two input parameters of type INTEGER and CHAR(10), respectively. Assume that the schema has more than one function named NEW_DEPT_HIRES.

```
REVOKE EXECUTE ON FUNCTION NEW_DEPT_HIRES (INTEGER, CHAR(10))
  FROM HR RESTRICT
```

Example 4: Revoke the EXECUTE privilege on method SET_SALARY for type EMPLOYEE from user Jones.

```
REVOKE EXECUTE ON METHOD SET_SALARY FOR EMPLOYEE FROM JONES RESTRICT
```

Related reference:

- "REVOKE (Database Authorities)" on page 733
- "REVOKE (Index Privileges)" on page 738
- "REVOKE (Package Privileges)" on page 740
- "REVOKE (Schema Privileges)" on page 745
- "REVOKE (Table, View, or Nickname Privileges)" on page 752
- "REVOKE (Server Privileges)" on page 749
- "REVOKE (Table Space Privileges)" on page 750
- "Common syntax elements" in the *SQL Reference, Volume 2*

REVOKE (Schema Privileges)

This form of the REVOKE statement revokes the privileges on a schema.

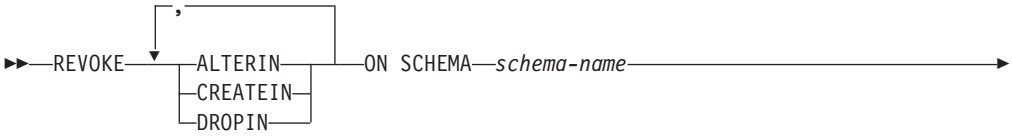
Invocation:

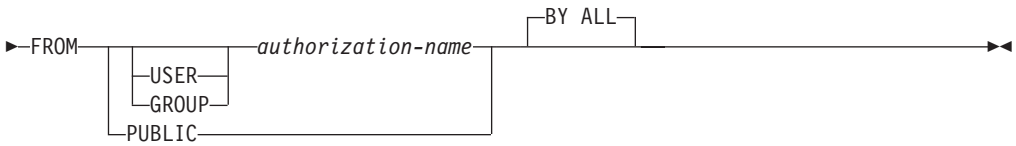
This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization:

The authorization ID of the statement must hold either SYSADM or DBADM authority (SQLSTATE 42501).

Syntax:





Description:

ALTERIN
Revokes the privilege to alter or comment on objects in the schema.

CREATEIN
Revokes the privilege to create objects in the schema.

DROPIN
Revokes the privilege to drop objects in the schema.

ON SCHEMA *schema-name*
Specifies the name of the schema on which privileges are to be revoked.

FROM
Indicates from whom the privileges are revoked.

USER
Specifies that the *authorization-name* identifies a user.

GROUP
Specifies that the *authorization-name* identifies a group name.

authorization-name,...
Lists one or more authorization IDs.

The authorization ID of the REVOKE statement itself cannot be used (SQLSTATE 42502). It is not possible to revoke the privileges from an *authorization-name* that is the same as the authorization ID of the REVOKE statement.

PUBLIC
Revokes the privileges from PUBLIC.

BY ALL
Revokes each named privilege from all named users who were explicitly granted those privileges, regardless of who granted them. This is the default behavior.

Rules:

- If neither USER nor GROUP is specified, then:
  - If all rows for the grantee in the SYSCAT.SCHEMAAUTH catalog view have a GRANTEETYPE of U, then USER will be assumed.
  - If all rows have a GRANTEETYPE of G, then GROUP will be assumed.
  - If some rows have U and some rows have G, then an error (SQLSTATE 56092) is raised.

Notes:

- Revoking a specific privilege does not necessarily revoke the ability to perform the action. A user may proceed with their task if other privileges are held by PUBLIC or a group, or if they have a higher level authority such as DBADM.

Examples:

Example 1: Given that USER4 is only a user and not a group, revoke the privilege to create objects in schema DEPTIDX from the user USER4.

```
REVOKE CREATEIN ON SCHEMA DEPTIDX FROM USER4
```

Example 2: Revoke the privilege to drop objects in schema LUNCH from the user CHEF and the group WAITERS.

```
REVOKE DROPIN ON SCHEMA LUNCH
  FROM USER CHEF, GROUP WAITERS
```

Related reference:

- "REVOKE (Database Authorities)" on page 733
- "REVOKE (Index Privileges)" on page 738
- "REVOKE (Package Privileges)" on page 740
- "REVOKE (Table, View, or Nickname Privileges)" on page 752
- "REVOKE (Server Privileges)" on page 749
- "REVOKE (Table Space Privileges)" on page 750
- "REVOKE (Routine Privileges)" on page 742

REVOKE (Sequence Privileges)

This form of the REVOKE statement revokes privileges on a sequence.

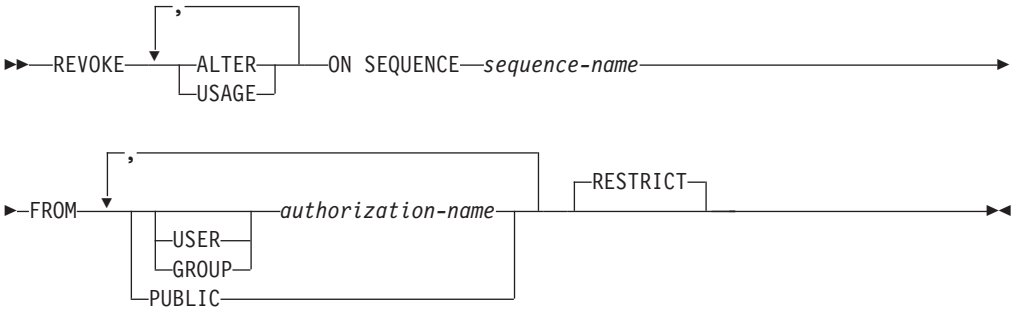
Invocation:

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared. However, if the bind option DYNAMICRULES BIND applies, the statement cannot be dynamically prepared (SQLSTATE 42509).

Authorization:

The privileges held by the authorization ID of the statement must include SYSADM or DBADM authority.

Syntax:



Description:

ALTER
Revokes the privilege to change the properties of a sequence or to restart sequence number generation using the ALTER SEQUENCE statement.

USAGE
Revokes the privilege to reference a sequence using *nextval-expression* or *prevval-expression*.

ON SEQUENCE *sequence-name*
Identifies the sequence on which the specified privileges are to be revoked. The sequence name, including an implicit or explicit schema qualifier, must uniquely identify an existing sequence at the current server. If no sequence by this name exists, an error is returned (SQLSTATE 42704).

FROM
Specifies from whom the privileges are revoked.

USER
Specifies that the *authorization-name* identifies a user.

GROUP
Specifies that the *authorization-name* identifies a group.

authorization-name,...
Lists the authorization IDs of one or more users or groups. The authorization ID of the REVOKE statement itself cannot be specified (SQLSTATE 42502).

PUBLIC
Revokes the specified privileges from all users.

RESTRICT
This optional keyword indicates that the statement will fail if any objects depend on the privilege being revoked.

Rules:

- If neither USER nor GROUP is specified, then:
  - If all rows for the grantee in the SYSCAT.SEQUENCEAUTH catalog view have a GRANTEETYPE of U, then USER is assumed.
  - If all rows have a GRANTEETYPE of G, then GROUP is assumed.
  - If some rows have U and some rows have G, an error is returned (SQLSTATE 56092).

Notes:

- Revoking a specific privilege does not necessarily remove the ability to perform an action. A user can proceed if other privileges are held by PUBLIC or by a group to which the user belongs, or if the user has a higher level of authority, such as DBADM.

Examples:

Example 1: Revoke the USAGE privilege on a sequence called GENERATE_ID from user ENGLES. There is one row in the SYSCAT.SEQUENCEAUTH catalog view for this sequence and grantee, and the GRANTEETYPE value is U.

```
REVOKE USAGE ON SEQUENCE GENERATE_ID FROM ENGLES
```

Example 2: Revoke alter privileges on sequence GENERATE_ID that were previously granted to all local users. (Grants to specific users are not affected.)

```
REVOKE ALTER ON SEQUENCE GENERATE_ID FROM PUBLIC
```

Example 3: Revoke all privileges on sequence GENERATE_ID from users PELLOW and MLI, and from group PLANNERS.

```
REVOKE ALTER, USAGE ON SEQUENCE GENERATE_ID
  FROM USER PELLOW, USER MLI, GROUP PLANNERS
```

Related reference:

- "REVOKE (Database Authorities)" on page 733
- "REVOKE (Index Privileges)" on page 738
- "REVOKE (Package Privileges)" on page 740
- "REVOKE (Schema Privileges)" on page 745
- "REVOKE (Table, View, or Nickname Privileges)" on page 752
- "REVOKE (Server Privileges)" on page 749
- "REVOKE (Table Space Privileges)" on page 750
- "GRANT (Sequence Privileges)" on page 713
- "REVOKE (Routine Privileges)" on page 742

REVOKE (Server Privileges)

This form of the REVOKE statement revokes the privilege to access and use a specified data source in pass-through mode.

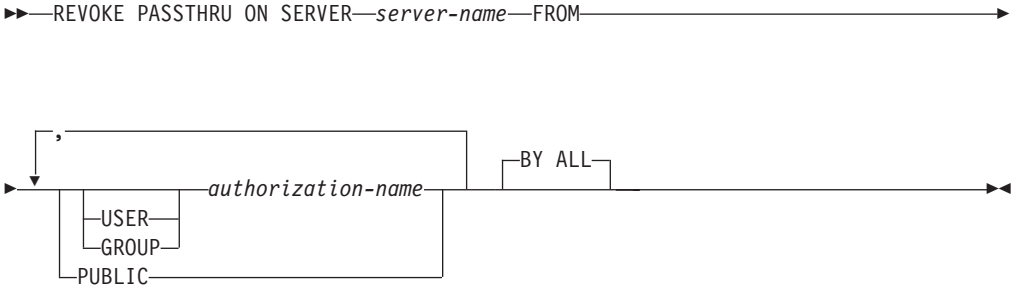
Invocation:

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization:

The authorization ID of the statement must have SYSADM or DBADM authority.

Syntax:



Description:

SERVER *server-name*
Names the data source for which the privilege to use in pass-through mode is being revoked. *server-name* must identify a data source that is described in the catalog.

FROM
Specifies from whom the privilege is revoked.

USER
Specifies that the *authorization-name* identifies a user.

GROUP
Specifies that the *authorization-name* identifies a group name.

authorization-name,...
Lists the authorization IDs of one or more users or groups.

The authorization ID of the REVOKE statement itself cannot be used (SQLSTATE 42502). It is not possible to revoke the privileges from an *authorization-name* that is the same as the authorization ID of the REVOKE statement.

PUBLIC
Revokes from all users the privilege to pass through to *server-name*.

BY ALL
Revokes the privilege from all named users who were explicitly granted that privilege, regardless of who granted it. This is the default behavior.

Examples:

Example 1: Revoke USER6's privilege to pass through to data source MOUNTAIN.

```
REVOKE PASSTHRU ON SERVER MOUNTAIN FROM USER USER6
```

Example 2: Revoke group D024's privilege to pass through to data source EASTWING.

```
REVOKE PASSTHRU ON SERVER EASTWING FROM GROUP D024
```

The members of group D024 will no longer be able to use their group ID to pass through to EASTWING. But if any members have the privilege to pass through to EASTWING under their own user IDs, they will retain this privilege.

Related reference:
- "REVOKE (Database Authorities)" on page 733
- "REVOKE (Index Privileges)" on page 738
- "REVOKE (Package Privileges)" on page 740
- "REVOKE (Schema Privileges)" on page 745
- "REVOKE (Table, View, or Nickname Privileges)" on page 752
- "REVOKE (Table Space Privileges)" on page 750
- "REVOKE (Routine Privileges)" on page 742

REVOKE (Table Space Privileges)

This form of the REVOKE statement revokes the USE privilege on a table space.

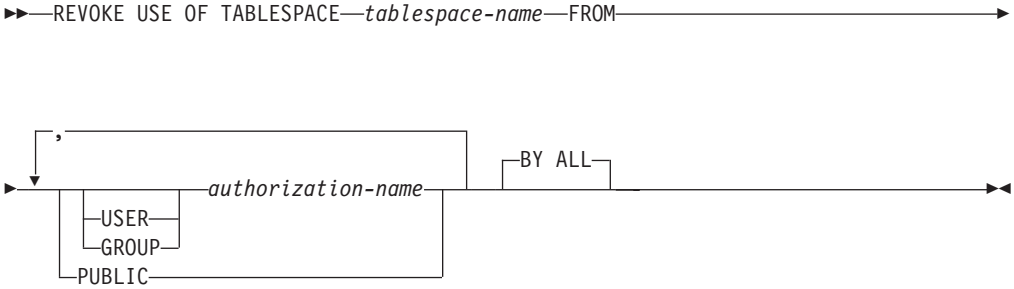
Invocation:

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization:

The authorization ID of the statement must hold either SYSADM, SYSCTRL or DBADM authority (SQLSTATE 42501).

Syntax:



Description:

USE
Revokes the privilege to specify or default to the table space when creating a table.

OF TABLESPACE *tablespace-name*
Specifies the table space on which the USE privilege is to be revoked. The table space cannot be SYSCATSPACE (SQLSTATE 42838) or a SYSTEM TEMPORARY table space (SQLSTATE 42809).

FROM
Indicates from whom the USE privilege is revoked.

USER
Specifies that the *authorization-name* identifies a user.

GROUP
Specifies that the *authorization-name* identifies a group name.

authorization-name
Lists one or more authorization IDs.

The authorization ID of the REVOKE statement itself cannot be used (SQLSTATE 42502). It is not possible to revoke the privileges from an *authorization-name* that is the same as the authorization ID of the REVOKE statement.

PUBLIC
Revokes the USE privilege from PUBLIC.

BY ALL
Revokes the privilege from all named users who were explicitly granted that privilege, regardless of who granted it. This is the default behavior.

Rules:

- If neither USER nor GROUP is specified, then:
  - If all rows for the grantee in the SYSCAT.TBSPACEAUTH catalog view have a GRANTEETYPE of U, then USER will be assumed.
  - If all rows have a GRANTEETYPE of G, then GROUP will be assumed.
  - If some rows have U and some rows have G, then an error results (SQLSTATE 56092).

Notes:

- Revoking the USE privilege does not necessarily revoke the ability to create tables in that table space. A user may still be able to create tables in that table space if the USE privilege is held by PUBLIC or a group, or if the user has a higher level authority, such as DBADM.

Examples:

Example 1: Revoke the privilege to create tables in table space PLANS from the user BOBBY.

```
REVOKE USE OF TABLESPACE PLANS FROM USER BOBBY
```

Related reference:

- "REVOKE (Database Authorities)" on page 733
- "REVOKE (Index Privileges)" on page 738
- "REVOKE (Package Privileges)" on page 740
- "REVOKE (Schema Privileges)" on page 745
- "REVOKE (Table, View, or Nickname Privileges)" on page 752
- "REVOKE (Server Privileges)" on page 749
- "REVOKE (Routine Privileges)" on page 742

REVOKE (Table, View, or Nickname Privileges)

This form of the REVOKE statement revokes privileges on a table, view, or nickname.

Invocation:

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization:

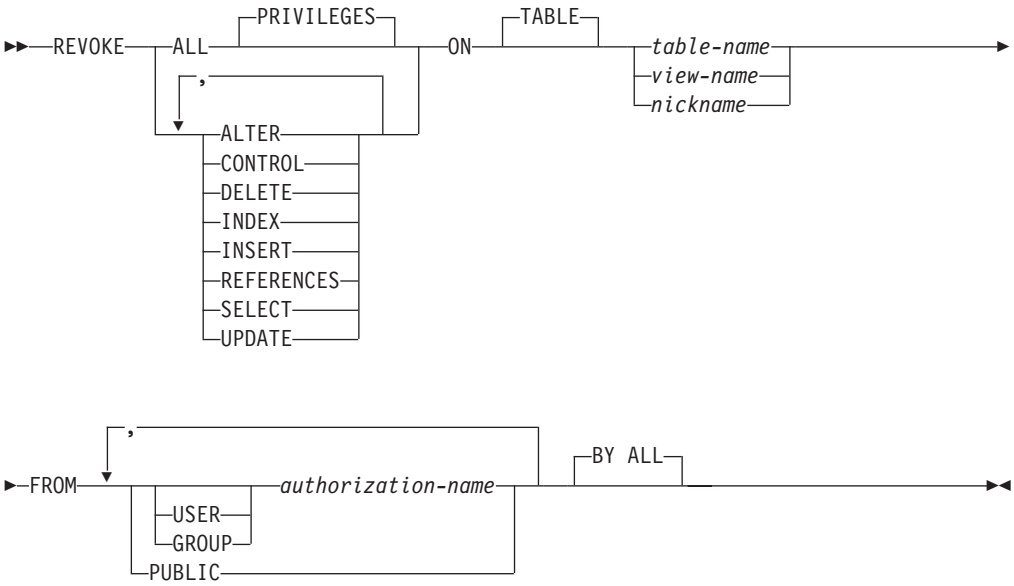
The privileges held by the authorization ID of the statement must include at least one of the following:

- SYSADM or DBADM authority
- CONTROL privilege on the referenced table, view, or nickname.

To revoke the CONTROL privilege, either SYSADM or DBADM authority is required.

To revoke the privileges on catalog tables and views, either SYSADM or DBADM authority is required.

Syntax:



Description:

ALL or **ALL PRIVILEGES**
Revokes all privileges (except CONTROL) held by an authorization-name for the specified tables, views, or nicknames.

If ALL is not used, one or more of the keywords listed below must be used. Each keyword revokes the privilege described, but only as it applies to the tables, views, or nicknames named in the ON clause. The same keyword must not be specified more than once.

ALTER
Revokes the privilege to add columns to the base table definition; create or drop a primary key or unique constraint on the table; create or drop a foreign key on the table; add/change a comment on the table, view, or nickname; create or drop a check constraint; create a trigger; add, reset, or drop a column option for a nickname; or, change nickname column names or data types.

CONTROL
Revokes the ability to drop the table, view, or nickname, and the ability to execute the RUNSTATS utility on the table and indexes.

Revoking CONTROL privilege from an *authorization-name* does not revoke other privileges granted to the user on that object.

DELETE
Revokes the privilege to delete rows from the table, updatable view, or nickname.

INDEX
Revokes the privilege to create an index on the table or an index specification on the nickname. The creator of an index or index specification automatically has the CONTROL privilege over the index or index specification (authorizing the creator to drop the index or index specification). In addition, the creator retains this privilege even if the INDEX privilege is revoked.

INSERT
Revokes the privileges to insert rows into the table, updatable view, or nickname, and to run the IMPORT utility.

REFERENCES
Revokes the privilege to create or drop a foreign key referencing the table as the parent. Any column level REFERENCES privileges are also revoked.

SELECT
Revokes the privilege to retrieve rows from the table or view, to create a view on a table, and to run the EXPORT utility against the table or view.

Revoking SELECT privilege may cause some views to be marked inoperative. (For information on inoperative views, see "CREATE VIEW".)

UPDATE
Revokes the privilege to update rows in the table, updatable view, or nickname. Any column level UPDATE privileges are also revoked.

ON TABLE *table-name* or *view-name* or *nickname*
Specifies the table, view, or nickname on which privileges are to be revoked. The *table-name* cannot be a declared temporary table (SQLSTATE 42995).

FROM
Indicates from whom the privileges are revoked.

USER
Specifies that the *authorization-name* identifies a user.

GROUP
Specifies that the *authorization-name* identifies a group name.

authorization-name,...
Lists one or more authorization IDs.

The ID of the REVOKE statement itself cannot be used (SQLSTATE 42502). It is not possible to revoke the privileges from an *authorization-name* that is the same as the authorization ID of the REVOKE statement.

PUBLIC
Revokes the privileges from PUBLIC.

BY ALL
Revokes each named privilege from all named users who were explicitly granted those privileges, regardless of who granted them. This is the default behavior.

Rules:

- If neither USER nor GROUP is specified, then:
  - If all rows for the grantee in the SYSCAT.TABAUTH and SYSCAT.COLAUTH catalog views have a GRANTEETYPE of U, then USER will be assumed.
  - If all rows have a GRANTEETYPE of G, then GROUP will be assumed.
  - If some rows have U and some rows have G, then an error (SQLSTATE 56092) is raised.

Notes:

- If a privilege is revoked from the *authorization-name* used to create a view (this is called the view's DEFINER in SYSCAT.VIEWS), that privilege is also revoked from any dependent views.
- If the DEFINER of the view loses a SELECT privilege on some object on which the view definition depends (or an object upon which the view definition depends is dropped, or made inoperative in the case of another view), the view will be made inoperative.

However, if a DBADM or SYSADM explicitly revokes all privileges on the view from the DEFINER, then the record of the DEFINER will not appear in SYSCAT.TABAUTH but nothing will happen to the view - it remains operative.

- Privileges on inoperative views cannot be revoked.
- All packages dependent upon an object for which a privilege is revoked are marked invalid. A package remains invalid until a bind or rebind operation on the application is successfully executed, or the application is executed and the database manager successfully rebinds the application (using information stored in the catalogs). Packages marked invalid due to a revoke may be successfully rebound without any additional grants.

  For example, if a package owned by USER1 contains a SELECT from table T1 and the SELECT privilege for table T1 is revoked from USER1, then the package will be marked invalid. If SELECT authority is re-granted, or if the user holds DBADM authority, the package is successfully rebound when executed.
- Packages, triggers or views that include the use of OUTER(*Z*) in the FROM clause, are dependent on having SELECT privilege on every subtable or subview of *Z*. Similarly, packages, triggers, or views that include the use of DEREF(*Y*) where *Y* is a reference type with a target table or view *Z*, are dependent on having SELECT privilege on every subtable or subview of *Z*. If one of these SELECT privileges is revoked, such packages are invalidated and such triggers or views are made inoperative.
- Table, view, or nickname privileges cannot be revoked from an *authorization-name* with CONTROL on the object without also revoking the CONTROL privilege (SQLSTATE 42504).
- Revoking a specific privilege does not necessarily revoke the ability to perform the action. A user may proceed with their task if other privileges are held by PUBLIC or a group, or if they have privileges such as ALTERIN on the schema of a table or a view.
- If the DEFINER of the materialized query table loses a SELECT privilege on a table on which the materialized query table definition depends (or a table upon which the materialized query table definition depends is dropped), the materialized query table will be made inoperative.

  However, if a DBADM or SYSADM explicitly revokes all privileges on the materialized query table from the DEFINER, then the record in SYSTABAUTH for the DEFINER will be deleted, but nothing will happen to the materialized query table - it remains operative.
- Revoking nickname privileges has no affect on data source object (table or view) privileges.
- Revoking the SELECT privilege for a table or view that is directly or indirectly referenced in an SQL function or method body may fail if the SQL function or method body cannot be dropped because some other object is dependent on it (SQLSTATE 42893).
- If the DEFINER of the SQL function or method body loses the SELECT privilege on some object on which the function or method body definition depends (or if an object upon which the function or method body definition depends is dropped), the function or method body will be dropped, unless another object depends on the function or method (SQLSTATE 42893).

Examples:

Example 1: Revoke SELECT privilege on table EMPLOYEE from user ENGLES. There is one row in the SYSCAT.TABAUTH catalog view for this table and grantee and the GRANTEETYPE value is U.

```
REVOKE SELECT
  ON TABLE EMPLOYEE
  FROM ENGLES
```

Example 2: Revoke update privileges on table EMPLOYEE previously granted to all local users. Note that grants to specific users are not affected.

```
REVOKE UPDATE
  ON EMPLOYEE
  FROM PUBLIC
```

Example 3: Revoke all privileges on table EMPLOYEE from users PELLOW and MLI and from group PLANNERS.

```
REVOKE ALL
  ON EMPLOYEE
  FROM USER PELLOW, USER MLI, GROUP PLANNERS
```

Example 4: Revoke SELECT privilege on table CORPDATA.EMPLOYEE from a user named JOHN. There is one row in the SYSCAT.TABAUTH catalog view for this table and grantee and the GRANTEETYPE value is U.

```
REVOKE SELECT
  ON CORPDATA.EMPLOYEE FROM JOHN
```

or

```
REVOKE SELECT
  ON CORPDATA.EMPLOYEE FROM USER JOHN
```

Note that an attempt to revoke the privilege from GROUP JOHN would result in an error, since the privilege was not previously granted to GROUP JOHN.

Example 5: Revoke SELECT privilege on table CORPDATA.EMPLOYEE from a group named JOHN. There is one row in the SYSCAT.TABAUTH catalog view for this table and grantee and the GRANTEETYPE value is G.

```
REVOKE SELECT
  ON CORPDATA.EMPLOYEE FROM JOHN
```

or

```
REVOKE SELECT
  ON CORPDATA.EMPLOYEE FROM GROUP JOHN
```

Example 6: Revoke user SHAWN's privilege to create an index specification on nickname ORAREM1.

```
REVOKE INDEX
  ON ORAREM1 FROM USER SHAWN
```

Related reference:

- "CREATE TABLE" on page 591
- "CREATE VIEW" on page 656
- "DROP" on page 676
- "REVOKE (Database Authorities)" on page 733
- "REVOKE (Index Privileges)" on page 738
- "REVOKE (Package Privileges)" on page 740
- "REVOKE (Schema Privileges)" on page 745
- "REVOKE (Server Privileges)" on page 749
- "REVOKE (Table Space Privileges)" on page 750

- "REVOKE (Routine Privileges)" on page 742

Related samples:
- "tbpriv.sqc -- How to grant, display, and revoke privileges (C)"
- "tbpriv.sqC -- How to grant, display, and revoke privileges (C++)"
- "TbPriv.java -- How to grant, display and revoke privileges on a table (JDBC)"
- "TbPriv.sqlj -- How to grant, display and revoke privileges on a table (SQLj)"

UPDATE

The UPDATE statement updates the values of specified columns in rows of a table, view or nickname, or the underlying tables, nicknames, or views of the specified fullselect. Updating a row of a view updates a row of its base table, if no INSTEAD OF trigger is defined for the update operation on this view. If such a trigger is defined, the trigger will be executed instead. Updating a row using a nickname updates a row in the data source object to which the nickname refers.

The forms of this statement are:
- The *Searched* UPDATE form is used to update one or more rows (optionally determined by a search condition).
- The *Positioned* UPDATE form is used to update exactly one row (as determined by the current position of a cursor).

Invocation:

An UPDATE statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared.

Authorization:

The privileges held by the authorization ID of the statement must include at least one of the following:
- UPDATE privilege on the table, view or nickname where rows are to be updated
- UPDATE privilege on each of the columns to be updated.
- CONTROL privilege on the table, view or nickname where rows are to be updated
- SYSADM or DBADM authority.
- If a *row-fullselect* is included in the assignment, at least one of the following for each referenced table, view or nickname:
  - SELECT privilege
  - CONTROL privilege
  - SYSADM or DBADM authority.

For each table, view or nickname referenced by a subquery, the privileges held by the authorization ID of the statement must also include at least one of the following:
- SELECT privilege
- CONTROL privilege
- SYSADM or DBADM authority.

If the package used to process the statement is precompiled with SQL92 rules (option LANGLEVEL with a value of SQL92E or MIA), and the searched form of an UPDATE statement includes a reference to a column of the table, view or nickname in the right side of the *assignment-clause*, or anywhere in the *search-condition*, the privileges held by the authorization ID of the statement must also include at least one of the following:

- SELECT privilege
- CONTROL privilege
- SYSADM or DBADM authority.

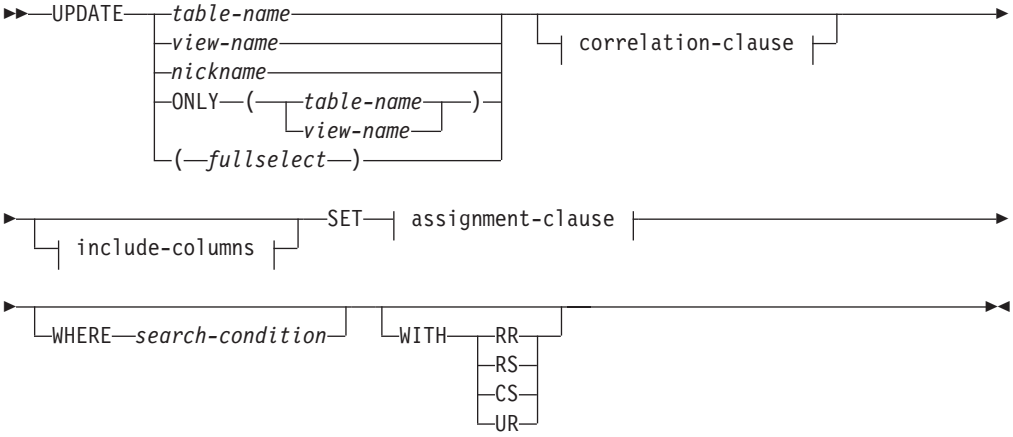
If the specified table or view is preceded by the ONLY keyword, the privileges held by the authorization ID of the statement must also include the SELECT privilege for every subtable or subview of the specified table or view.

GROUP privileges are not checked for static UPDATE statements.

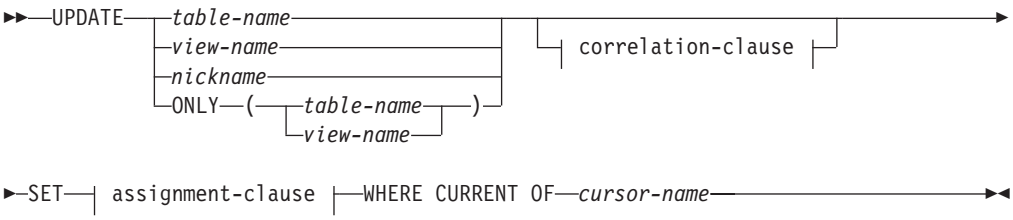
If the target of the update operation is a nickname, the privileges on the object at the data source are not considered until the statement is executed at the data source. At this time, the authorization ID that is used to connect to the data source must have the privileges required for the operation on the object at the data source. The authorization ID of the statement may be mapped to a different authorization ID at the data source.

Syntax:

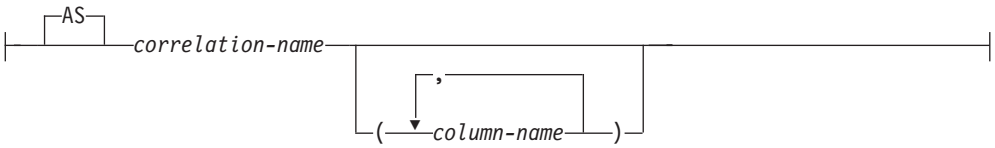
searched-update:



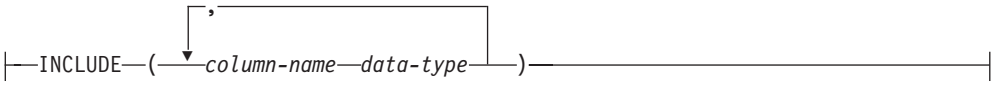
positioned-update:



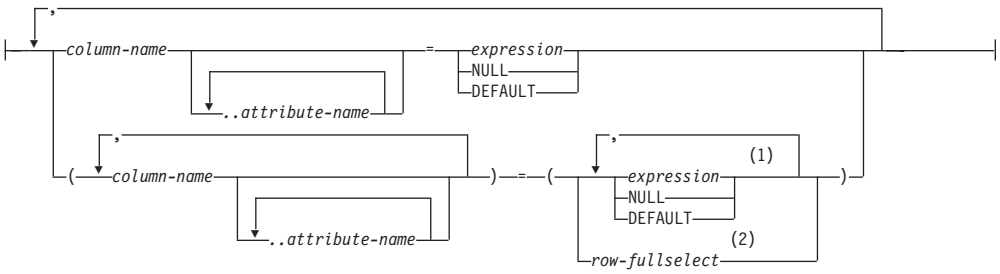
correlation-clause:



include-columns:



assignment-clause:



Notes:

1 The number of expressions, NULLs and DEFAULTs must match the number of column names.

2 The number of columns in the select list must match the number of column names.

Description:

table-name, *view-name*, *nickname*, **or** *(fullselect)*
Identifies the object of the update operation. The name must identify a table, view, or nickname described in the catalog, but not a catalog table, a view of a catalog table (unless it is one of the updatable SYSSTAT views), a system-maintained materialized query table, or a read-only view that has no INSTEAD OF trigger defined for its update operations.

If *table-name* is a typed table, rows of the table or any of its proper subtables may get updated by the statement. Only the columns of the specified table may be set or referenced in the WHERE clause. For a positioned UPDATE, the associated cursor must also have specified the same table, view or nickname in the FROM clause without using ONLY.

If the object of the update operation is a fullselect, the fullselect must be updatable, as defined in the "Updatable views" Notes item in the description of the CREATE VIEW statement.

ONLY (*table-name***)**
Applicable to typed tables, the ONLY keyword specifies that the statement should apply only to data of the specified table and rows of proper subtables cannot be updated by the statement. For a positioned UPDATE, the associated

cursor must also have specified the table in the FROM clause using ONLY. If *table-name* is not a typed table, the ONLY keyword has no effect on the statement.

ONLY (*view-name***)**
Applicable to typed views, the ONLY keyword specifies that the statement should apply only to data of the specified view and rows of proper subviews cannot be updated by the statement. For a positioned UPDATE, the associated cursor must also have specified the view in the FROM clause using ONLY. If *view-name* is not a typed view, the ONLY keyword has no effect on the statement.

correlation-clause
Can be used within *search-condition* or *assignment-clause* to designate a table, view, nickname, or fullselect. For a description of *correlation-clause*, see "table-reference" in the description of "Subselect".

include-columns
Specifies a set of columns that are included, along with the columns of *table-name* or *view-name*, in the intermediate result table of the UPDATE statement when it is nested in the FROM clause of a fullselect. The *include-columns* are appended at the end of the list of columns that are specified for *table-name* or *view-name*.

INCLUDE
Specifies a list of columns to be included in the intermediate result table of the UPDATE statement.

column-name
Specifies a column of the intermediate result table of the UPDATE statement. The name cannot be the same as the name of another include column or a column in *table-name* or *view-name* (SQLSTATE 42711).

data-type
Specifies the data type of the include column. The data type must be one that is supported by the CREATE TABLE statement.

SET
Introduces the assignment of values to column names.

assignment-clause

column-name
Identifies a column to be updated. The *column-name* must identify an updatable column of the specified table, view, or nickname, or identify an INCLUDE column. The object ID column of a typed table is not updatable (SQLSTATE 428DZ). A column must not be specified more than once, unless it is followed by *..attribute-name* (SQLSTATE 42701).

If it specifies an INCLUDE column, the column name cannot be qualified.

For a Positioned UPDATE:

- If the *update-clause* was specified in the *select-statement* of the cursor, each column name in the *assignment-clause* must also appear in the *update-clause*.
- If the *update-clause* was not specified in the *select-statement* of the cursor and LANGLEVEL MIA or SQL92E was specified when the application was precompiled, the name of any updatable column may be specified.
- If the *update-clause* was not specified in the *select-statement* of the cursor and LANGLEVEL SAA1 was specified either explicitly or by default when the application was precompiled, no columns may be updated.

..attribute-name
Specifies the attribute of a structured type that is set (referred to as an *attribute assignment*. The *column-name* specified must be defined with a user-defined structured type (SQLSTATE 428DP). The attribute-name must be an attribute of the structured type of *column-name* (SQLSTATE 42703). An assignment that does not involve the *..attribute-name* clause is referred to as a *conventional assignment*.

expression
Indicates the new value of the column. The expression is any expression of the type described in "Expressions". The expression cannot include a column function except when it occurs within a scalar fullselect (SQLSTATE 42903).

An *expression* may contain references to columns of the target table of the UPDATE statement. For each row that is updated, the value of such a column in an expression is the value of the column in the row before the row is updated.

An expression cannot contain references to an INCLUDE column.

NULL
Specifies the null value and can only be specified for nullable columns (SQLSTATE 23502). NULL cannot be the value in an attribute assignment (SQLSTATE 429B9) unless it is specifically cast to the data type of the attribute.

DEFAULT
Specifies that the default value should be used based on how the corresponding column is defined in the table. The value that is inserted depends on how the column was defined.

- If the column was defined as a generated column based on an expression, the column value will be generated by the system, based on the expression.
- If the column was defined using the IDENTITY clause, the value is generated by the database manager.
- If the column was defined using the WITH DEFAULT clause, the value is set to the default defined for the column (see *default-clause* in "ALTER TABLE").
- If the column was defined using the NOT NULL clause and the GENERATED clause was not used, or the WITH DEFAULT clause was not used, or DEFAULT NULL was used, the DEFAULT keyword cannot be specified for that column (SQLSTATE 23502).

The only value that a generated column defined with the GENERATED ALWAYS clause can be set to is DEFAULT (SQLSTATE 428C9).

The DEFAULT keyword cannot be used as the value in an attribute assignment (SQLSTATE 429B9).

The DEFAULT keyword cannot be used as the value in an assignment for update on a nickname where the data source does not support DEFAULT syntax.

row-fullselect
A fullselect that returns a single row with the number of columns corresponding to the number of *column-name*s specified for assignment. The values are assigned to each corresponding *column-name*. If the result of the *row-fullselect* is no rows, then null values are assigned.

A *row-fullselect* may contain references to columns of the target table of the UPDATE statement. For each row that is updated, the value of such a column in an expression is the value of the column in the row before the row is updated. An error is returned if there is more than one row in the result (SQLSTATE 21000).

WHERE
Introduces a condition that indicates what rows are updated. You can omit the clause, give a search condition, or name a cursor. If the clause is omitted, all rows of the table, view or nickname are updated.

search-condition
Each *column-name* in the search condition, other than in a subquery, must name a column of the table, view or nickname. When the search condition includes a subquery in which the same table is the base object of both the UPDATE and the subquery, the subquery is completely evaluated before any rows are updated.

The search-condition is applied to each row of the table, view or nickname and the updated rows are those for which the result of the search-condition is true.

If the search condition contains a subquery, the subquery can be thought of as being executed each time the search condition is applied to a row, and the results used in applying the search condition. In actuality, a subquery with no correlated references is executed only once, whereas a subquery with a correlated reference may have to be executed once for each row.

CURRENT OF *cursor-name*
Identifies the cursor to be used in the update operation. The *cursor-name* must identify a declared cursor, explained in "DECLARE CURSOR". The DECLARE CURSOR statement must precede the UPDATE statement in the program.

The table, view or nickname named must also be named in the FROM clause of the SELECT statement of the cursor, and the result table of the cursor must not be read-only. (For an explanation of read-only result tables, see "DECLARE CURSOR".)

When the UPDATE statement is executed, the cursor must be positioned on a row; that row is updated.

This form of UPDATE cannot be used (SQLSTATE 42828) if the cursor references:

- A view on which an INSTEAD OF UPDATE trigger is defined
- A view that includes an OLAP function in the select list of the fullselect that defines the view
- A view that is defined, either directly or indirectly, using the WITH ROW MOVEMENT clause

WITH
Specifies the isolation level at which the UPDATE statement is executed.

RR
Repeatable Read

RS
Read Stability

CS
Cursor Stability

UR
Uncommitted Read

The default isolation level of the statement is the isolation level of the package in which the statement is bound.

Rules:

- ***Triggers:*** UPDATE statements may cause triggers to be executed. A trigger may cause other statements to be executed, or may raise error conditions based on the update values. If an update operation on a view causes an INSTEAD OF trigger to fire, validity, referential integrity, and constraints will be checked against the updates that are performed in the trigger, and not against the view that caused the trigger to fire, or its underlying tables.
- ***Assignment***: Update values are assigned to columns according to specific assignment rules.
- ***Validity***: The updated row must conform to any constraints imposed on the table (or on the base table of the view) by any unique index on an updated column.

  If a view is used that is not defined using WITH CHECK OPTION, rows can be changed so that they no longer conform to the definition of the view. Such rows are updated in the base table of the view and no longer appear in the view.

  If a view is used that is defined using WITH CHECK OPTION, an updated row must conform to the definition of the view. For an explanation of the rules governing this situation, see "CREATE VIEW".
- ***Check Constraint***: Update value must satisfy the check-conditions of the check constraints defined on the table.

  An UPDATE to a table with check constraints defined has the constraint conditions for each column updated evaluated once for each row that is updated. When processing an UPDATE statement, only the check constraints referring to the updated columns are checked.
- ***Referential Integrity***: The value of the parent unique keys cannot be changed if the update rule is RESTRICT and there are one or more dependent rows. However, if the update rule is NO ACTION, parent unique keys can be updated as long as every child has a parent key by the time the update statement completes. A non-null update value of a foreign key must be equal to a value of the primary key of the parent table of the relationship.

Notes:

- If an update value violates any constraints, or if any other error occurs during the execution of the UPDATE statement, no rows are updated. The order in which multiple rows are updated is undefined.
- An update to a view defined using the WITH ROW MOVEMENT clause could cause a delete operation and an insert operation against the underlying tables of the view. For details, see the description of the CREATE VIEW statement.
- When an UPDATE statement completes execution, the value of SQLERRD(3) in the SQLCA is the number of rows that qualified for the update operation. In the context of an SQL procedure statement, the value can be retrieved using the ROW_COUNT variable of the GET DIAGNOSTICS statement. The SQLERRD(5) field contains the number of rows inserted, deleted, or updated by all activated triggers.
- Unless appropriate locks already exist, one or more exclusive locks are acquired by the execution of a successful UPDATE statement. Until the locks are released, the updated row can only be accessed by the application process that performed

the update (except for applications using the Uncommitted Read isolation level). For further information on locking, see the descriptions of the COMMIT, ROLLBACK, and LOCK TABLE statements.

- If the URL value of a DATALINK column is updated, this is the same as deleting the old DATALINK value then inserting the new one. First, if the old value was linked to a file, that file is unlinked. Then, unless the linkage attributes of the DATALINK value are empty, the specified file is linked to that column. The only exception to this is that if the URL of the new DATALINK value is *identical* to the URL of the existing DATALINK value, there is no need to communicate with the associated Data Links Manager to unlink and relink the same file. In this situation, the overhead is entirely eliminated.

  The comment value of a DATALINK column can be updated without relinking the file by specifying an empty string as the URL path (for example, as the *data-location* argument of the DLVALUE scalar function or by specifying the new value to be the same as the old value).

  If a DATALINK column is updated with a null, it is the same as deleting the existing DATALINK value.

  An error may occur when attempting to update a DATALINK value if the file server of either the existing value or the new value is no longer registered with the database server (SQLSTATE 55022).
- When updating the column distribution statistics for a typed table, the subtable that first introduced the column must be specified.
- Multiple attribute assignments on the same structured type column occur in the order specified in the SET clause and, within a parenthesized set clause, in left-to-right order.
- An attribute assignment invokes the mutator method for the attribute of the user-defined structured type. For example, the assignment `st..a1=x` has the same effect as using the mutator method in the assignment `st = st..a1(x)`.
- While a given column may be a target column in only one conventional assignment, a column may be a target column in multiple attribute assignments (but only if it is not also a target column in a conventional assignment).
- When an identity column defined as a distinct type is updated, the entire computation is done in the source type, and the result is cast to the distinct type before the value is actually assigned to the column. (There is no casting of the previous value to the source type prior to the computation.)
- To have DB2 generate a value on a SET statement for an identity column, use the DEFAULT keyword:

  ```
  SET NEW.EMPNO = DEFAULT
  ```

  In this example, NEW.EMPNO is defined as an identity column, and the value used to update this column is generated by DB2.
- For more information about consuming values of a generated sequence for an identity column, or about exceeding the maximum value for an identity column, see “INSERT”.

Examples:

- *Example 1:* Change the job (JOB) of employee number (EMPNO) ‘000290’ in the EMPLOYEE table to ‘LABORER’.

  ```
  UPDATE EMPLOYEE
     SET JOB = 'LABORER'
     WHERE EMPNO = '000290'
  ```
- *Example 2:* Increase the project staffing (PRSTAFF) by 1.5 for all projects that department (DEPTNO) ‘D21’ is responsible for in the PROJECT table.

```
UPDATE PROJECT
   SET PRSTAFF = PRSTAFF + 1.5
   WHERE DEPTNO = 'D21'
```

- *Example 3:* All the employees except the manager of department (WORKDEPT) 'E21' have been temporarily reassigned. Indicate this by changing their job (JOB) to NULL and their pay (SALARY, BONUS, COMM) values to zero in the EMPLOYEE table.

  ```
  UPDATE EMPLOYEE
    SET JOB=NULL, SALARY=0, BONUS=0, COMM=0
    WHERE WORKDEPT = 'E21' AND JOB <> 'MANAGER'
  ```

  This statement could also be written as follows.

  ```
  UPDATE EMPLOYEE
    SET (JOB, SALARY, BONUS, COMM) = (NULL, 0, 0, 0)
    WHERE WORKDEPT = 'E21' AND JOB <> 'MANAGER'
  ```

- *Example 4:* Update the salary and the commission column of the employee with employee number 000120 to the average of the salary and of the commission of the employees of the updated row's department, respectively.

  ```
  UPDATE (SELECT SALARY,
      COMM,
      AVG(SALARY) OVER (PARTITION BY WORKDEPT),
      AVG(COMM) OVER (PARTITION BY WORKDEPT)
    FROM EMPLOYEE)
    AS E(SALARY, COMM, AVGSAL, AVGCOMM)
    SET (SALARY, COMM)
      = (AVGSAL, AVGCOMM)
    WHERE EU.EMPNO = '000120'
  ```

  The previous statement is semantically equivalent to the following statement, but requires only one access to the EMPLOYEE table, whereas the following statement specifies the EMPLOYEE table twice.

  ```
  UPDATE EMPLOYEE EU
    SET (EU.SALARY, EU.COMM)
    =
  (SELECT AVG(ES.SALARY), AVG(ES.COMM)
    FROM EMPLOYEE ES
    WHERE ES.WORKDEPT = EU.WORKDEPT)
    WHERE EU.EMPNO = '000120'
  ```

- *Example 5:* In a C program display the rows from the EMPLOYEE table and then, if requested to do so, change the job (JOB) of certain employees to the new job keyed in.

  ```
  EXEC SQL  DECLARE C1 CURSOR FOR
                SELECT *
                  FROM EMPLOYEE
                  FOR UPDATE OF JOB;

  EXEC SQL  OPEN C1;

  EXEC SQL  FETCH C1 INTO ...     ;
  if ( strcmp (change, "YES") == 0 )
    EXEC SQL  UPDATE EMPLOYEE
                SET JOB = :newjob
                WHERE CURRENT OF C1;

  EXEC SQL  CLOSE C1;
  ```

- *Example 6:* These examples mutate attributes of column objects.

  Assume that the following types and tables exist:

  ```
  CREATE TYPE POINT AS (X INTEGER, Y INTEGER)
    NOT FINAL WITHOUT COMPARISONS
    MODE DB2SQL
  ```

```
CREATE TYPE CIRCLE AS (RADIUS INTEGER, CENTER POINT)
  NOT FINAL WITHOUT COMPARISONS
  MODE DB2SQL

CREATE TABLE CIRCLES (ID INTEGER, OWNER VARCHAR(50), C CIRCLE
```

The following example updates the CIRCLES table by changing the OWNER column and the RADIUS attribute of the CIRCLE column where the ID is 999:

```
UPDATE CIRCLES
  SET OWNER = 'Bruce'
    C..RADIUS = 5
  WHERE ID = 999
```

The following example transposes the X and Y coordinates of the center of the circle identified by 999:

```
UPDATE CIRCLES
  SET C..CENTER..X = C..CENTER..Y,
    C..CENTER..Y = C..CENTER..X
  WHERE ID = 999
```

The following example is another way of writing both of the above statements. This example combines the effects of both of the above examples:

```
UPDATE CIRCLES
  SET (OWNER,C..RADIUS,C..CENTER..X,C..CENTER..Y) =
    ('Bruce',5,C..CENTER..Y,C..CENTER..X)
  WHERE ID = 999
```

Related reference:

- "Expressions" in the *SQL Reference, Volume 1*
- "Search conditions" in the *SQL Reference, Volume 1*
- "Subselect" on page 904
- "ALTER TABLE" on page 525
- "CREATE VIEW" on page 656
- "DECLARE CURSOR statement" in the *SQL Reference, Volume 2*
- "INSERT" on page 724
- "SQLCA (SQL communications area)" on page 1004
- "Assignments and comparisons" in the *SQL Reference, Volume 1*

Related samples:

- "dbinline.sqc -- How to use inline SQL Procedure Language (C)"
- "spserver.sqc -- Definition of various types of stored procedures (C)"
- "tbmod.sqc -- How to modify table data (C)"
- "tut_mod.sqc -- How to modify table data (C)"
- "dtstruct.sqC -- Create, use, drop a hierarchy of structured types and typed tables (C++)"
- "spserver.sqC -- Definition of various types of stored procedures (C++)"
- "tbmod.sqC -- How to modify table data (C++)"
- "tut_mod.sqC -- How to modify table data (C++)"
- "SpServer.java -- Provide a variety of types of stored procedures to be called from (JDBC)"
- "TbMod.java -- How to modify table data (JDBC)"
- "TutMod.java -- Modify data in a table (JDBC)"
- "SpServer.sqlj -- Provide a variety of types of stored procedures to be called from (SQLj)"
- "TbMod.sqlj -- How to modify table data (SQLj)"

- “TutMod.sqlj -- Modify data in a table (SQLj)”
- “tbmod.c -- How to modify table data”
- “tut_mod.c -- How to modify table data”
- “updat.sqb -- How to update, delete and insert table data (MF COBOL)”
- “varinp.sqb -- How to update table data using parameter markers (MF COBOL)”

Chapter 16. Configuration Parameters

Configuration parameters

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

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

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

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

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

Database manager configuration parameters are stored in a file named `db2systm`. This file is created when the instance of the database manager is created. In UNIX-based environments, this file can be found in the `sqllib` subdirectory for the instance of the database manager. In Windows, the default location of this file is the instance subdirectory of the `sqllib` directory. If the DB2INSTPROF variable is set, the file is in the `instance` subdirectory of the directory specified by the DB2INSTPROF variable.

© Copyright IBM Corp. 1993-2004

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

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

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

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

In a partitioned database environment, a separate `SQLDBCON` file exists for each database partition. The values in the `SQLDBCON` file may be the same or different at each database partition, but the recommendation is that the database configuration parameter values be the same on all partitions.

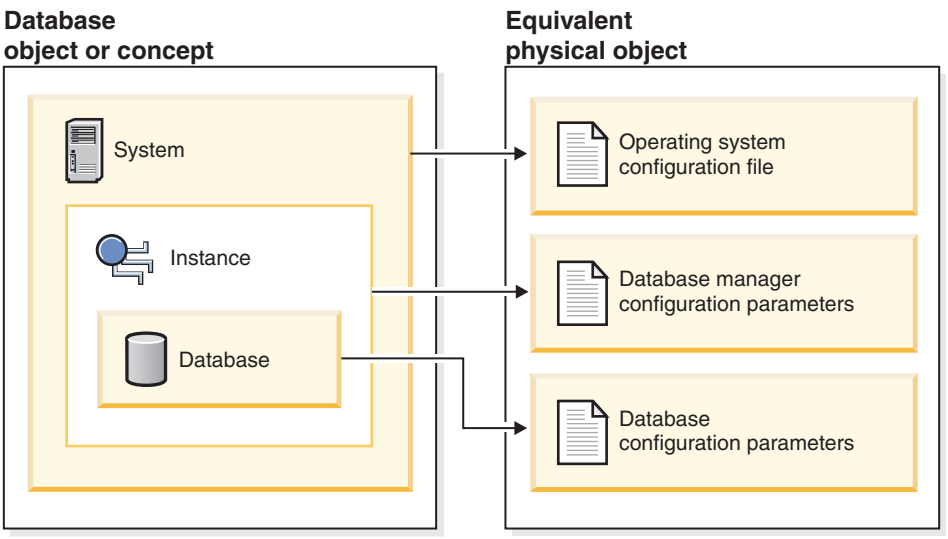


Figure 13. Relationship between database objects and configuration files

Related concepts:

- "Configuration parameter tuning" in the *Administration Guide: Performance*

Related tasks:

• "Configuring DB2 with configuration parameters" on page 779

Configuration parameters summary

Database Manager Configuration Parameter Summary

The following table lists the parameters in the database manager configuration file for database servers. When changing the database manager configuration parameters, consider the detailed information for each parameter. Specific operating environment information including defaults is part of each parameter description.

For some database manager configuration parameters, the database manager must be stopped (`db2stop`) and then restarted (`db2start`) for the new parameter values to take effect. Other parameters can be changed online; these are called *configurable online configuration parameters*. If you change the setting of a configurable online database manager configuration parameter while you are attached to an instance, the default behavior of the UPDATE DBM CFG command is to apply the change immediately. If you do not want the change applied immediately, use the DEFERRED option on the UPDATE DBM CFG command.

The column "Auto." in the following table indicates whether the parameter supports the AUTOMATIC keyword on the UPDATE DATABASE MANAGER CONFIGURATION command. If you set a parameter to automatic, DB2 will automatically adjust the parameter to reflect current resource requirements.

The column "Perf. Impact" provides an indication of the relative importance of each parameter as it relates to system performance. It is impossible for this column to apply accurately to all environments; you should view this information as a generalization.

- **High** — indicates the parameter can have a significant impact on performance. You should consciously decide the values of these parameters, which, in some cases, means that you will accept the default values provided.
- **Medium** — indicates the the parameter can have some impact on performance. Your specific environment and needs will determine how much tuning effort should be focused on these parameters.
- **Low** — indicates that the parameter has a less general or less significant impact on performance.
- **None** — indicates that the parameter does not directly impact performance. Although you do not have to tune these parameters for performance enhancement, they can be very important for other aspects of your system configuration, such as communication support, for example.

The columns "Token", "Token Value", and "Data Type" provide information that you will need when calling the `db2CfgGet` or the `db2CfgSet` API. This information includes configuration parameter identifiers, entries for the *token* element in the *db2CfgParam* data structure, and data types for values that are passed to the structure.

Table 64. Configurable Database Manager Configuration Parameters

Parameter	Cfg. Online	Auto.	Perf. Impact	Token	Token Value	Data Type	Additional Information
agent_stack_sz	No	No	Low	SQLF_KTN_AGENT_STACK_SZ	61	Uint16	
agentpri	No	No	High	SQLF_KTN_AGENTPRI	26	Sint16	
aslheapsz	No	No	High	SQLF_KTN_ASLHEAPSZ	15	Uint32	

Table 64. Configurable Database Manager Configuration Parameters (continued)

Parameter	Cfg. Online	Auto.	Perf. Impact	Token	Token Value	Data Type	Additional Information
audit_buf_sz	No	No	High	SQLF_KTN_AUDIT_BUF_SZ	312	Sint32	"audit_buf_sz - Audit buffer size" on page 782
authentication[1]	No	No	Low	SQLF_KTN_AUTHENTICATION	78	Uint16	"authentication - Authentication type" on page 783
catalog_noauth	Yes	No	None	SQLF_KTN_CATALOG_NOAUTH	314	Uint16	"catalog_noauth - Cataloging allowed without authority" on page 784
clnt_krb_plugin	No	No	None	SQLF_KTN_CLNT_KRB_PLUGIN	812	char(33)	"clnt_krb_plugin - Client Kerberos plug-in" on page 1085
clnt_pw_plugin	No	No	None	SQLF_KTN_CLNT_PW_PLUGIN	811	char(33)	"clnt_pw_plugin - Client userid-password plug-in" on page 1085
comm_bandwidth	Yes	No	Medium	SQLF_KTN_COMM_BANDWIDTH	307	float	
conn_elapse	Yes	No	Medium	SQLF_KTN_CONN_ELAPSE	508	Uint16	
cpuspeed	Yes	No	Low[2]	SQLF_KTN_CPUSPEED	42	float	
datalinks	No	No	Low	SQLF_KTN_DATALINKS	603	Sint16	
dft_account_str	Yes	No	None	SQLF_KTN_DFT_ACCOUNT_STR	28	char(25)	
dft_monswitches • *dft_mon_bufpool* • *dft_mon_lock* • *dft_mon_sort* • *dft_mon_stmt* • *dft_mon_table* • *dft_mon_timestamp* • *dft_mon_uow*	Yes	No	Medium	SQLF_KTN_DFT_MONSWITCHES[3] • SQLF_KTN_DFT_MON_BUFPOOL • SQLF_KTN_DFT_MON_LOCK • SQLF_KTN_DFT_MON_SORT • SQLF_KTN_DFT_MON_STMT • SQLF_KTN_DFT_MON_TABLE • SQLF_KTN_DFT_MON_ TIMESTAMP • SQLF_KTN_DFT_MON_UOW	29 • 33 • 34 • 35 • 31 • 32 • 36 • 30	Uint16 • Uint16 • Uint16 • Uint16 • Uint16 • Uint16 • Uint16 • Uint16	
dftdbpath	Yes	No	None	SQLF_KTN_DFTDBPATH	27	char(215)	"dftdbpath - Default database path" on page 785
diaglevel	Yes	No	Low	SQLF_KTN_DIAGLEVEL	64	Uint16	
diagpath	Yes	No	None	SQLF_KTN_DIAGPATH	65	char(215)	
dir_cache	No	No	Medium	SQLF_KTN_DIR_CACHE	40	Uint16	
discover[4]	No	No	Medium	SQLF_KTN_DISCOVER	304	Uint16	
discover_inst	Yes	No	Low	SQLF_KTN_DISCOVER_INST	308	Uint16	
fcm_num_anchors	No	Yes	Medium	SQLF_KTN_FCM_NUM_ANCHORS	506	Sint32	
fcm_num_buffers	Yes	No	Medium	SQLF_KTN_FCM_NUM_BUFFERS	503	Uint32	
fcm_num_connect	No	Yes	Medium	SQLF_KTN_FCM_NUM_CONNECT	505	Sint32	
fcm_num_rqb	No	Yes	Medium	SQLF_KTN_FCM_NUM_RQB	504	Uint32	
fed_noauth	Yes	No	None	SQLF_KTN_FED_NOAUTH	806	Uint16	
federated	No	No	Medium	SQLF_KTN_FEDERATED	604	Sint16	
fenced_pool	No	No	Medium	SQLF_KTN_FENCED_POOL	80	Sint32	
group_plugin	No	No	None	SQLF_KTN_GROUP_PLUGIN	810	char(33)	"group_plugin - Group plug-in" on page 1086
health_mon	Yes	No	Low	SQLF_KTN_HEALTH_MON	804	Uint16	
indexrec[5]	Yes	No	Medium	SQLF_KTN_INDEXREC	20	Uint16	
instance_memory	No	Yes	Medium	SQLF_KTN_INSTANCE_MEMORY	803	Uint64	
intra_parallel	No	No	High	SQLF_KTN_INTRA_PARALLEL	306	Sint16	
java_heap_sz	No	No	High	SQLF_KTN_JAVA_HEAP_SZ	310	Sint32	
jdk_path	No	No	None	SQLF_KTN_JDK_PATH	311	char(255)	
keepfenced	No	No	Medium	SQLF_KTN_KEEPFENCED	81	Uint16	
local_gssplugin	No	No	None	SQLF_KTN_LOCAL_GSSPLUGIN	816	char(33)	"local_gssplugin - GSS API plug-in used for local instance level authorization" on page 1086
max_connections	No	No	Medium	SQLF_DBTN_MAX_CONNECTIONS	802	Sint32	
max_connretries	Yes	No	Medium	SQLF_KTN_MAX_CONNRETRIES	509	Uint16	

Table 64. Configurable Database Manager Configuration Parameters (continued)

Parameter	Cfg. Online	Auto.	Perf. Impact	Token	Token Value	Data Type	Additional Information
max_coordagents	No	No	Medium	SQLF_KTN_MAX_COORDAGENTS	501	Sint32	
max_querydegree	Yes	No	High	SQLF_KTN_MAX_QUERYDEGREE	303	Sint32	
max_time_diff	No	No	Medium	SQLF_KTN_MAX_TIME_DIFF	510	Uint16	
maxagents	No	No	Medium	SQLF_KTN_MAXAGENTS	12	Uint32	
maxcagents	No	No	Medium	SQLF_KTN_MAXCAGENTS	13	Sint32	
maxtotfilop	No	No	Medium	SQLF_KTN_MAXTOTFILOP	45	Uint16	
min_priv_mem	No	No	Medium	SQLF_KTN_MIN_PRIV_MEM	43	Uint32	
mon_heap_sz	No	No	Low	SQLF_KTN_MON_HEAP_SZ	79	Uint16	
nname	No	No	None	SQLF_KTN_NNAME	7	char(8)	
notifylevel	Yes	No	Low	SQLF_KTN_NOTIFYLEVEL	605	Sint16	
num_initagents	No	No	Medium	SQLF_KTN_NUM_INITAGENTS	500	Uint32	
num_initfenced	No	No	Medium	SQLF_KTN_NUM_INITFENCED	601	Sint32	
num_poolagents	No	No	High	SQLF_KTN_NUM_POOLAGENTS	502	Sint32	
numdb	No	No	Low	SQLF_KTN_NUMDB	6	Uint16	
priv_mem_thresh	No	No	Medium	SQLF_KTN_PRIV_MEM_THRESH	44	Sint32	
query_heap_sz	No	No	Medium	SQLF_KTN_QUERY_HEAP_SZ	49	Sint32	
resync_interval	No	No	None	SQLF_KTN_RESYNC_INTERVAL	68	Uint16	
rqrioblk	No	No	High	SQLF_KTN_RQRIOBLK	1	Uint16	
sheapthres	No	No	High	SQLF_KTN_SHEAPTHRES	21	Uint32	
spm_log_file_sz	No	No	Low	SQLF_KTN_SPM_LOG_FILE_SZ	90	Sint32	
spm_log_path	No	No	Medium	SQLF_KTN_SPM_LOG_PATH	313	char(226)	
spm_max_resync	No	No	Low	SQLF_KTN_SPM_MAX_RESYNC	91	Sint32	
spm_name	No	No	None	SQLF_KTN_SPM_NAME	92	char(8)	
srvcon_auth	No	No	None	SQLF_KTN_SRVCON_AUTH	815	Uint16	"srvcon_auth - Authentication type for incoming connections at the server" on page 1087
srvcon_gssplugin_list	No	No	None	SQLF_KTN_SRVCON_GSSPLUGIN_LIST	814	char(256)	"srvcon_gssplugin_list - List of GSS API plug-ins for incoming connections at the server" on page 1087
srv_plugin_mode	No	No	None	SQLF_KTN_SRV_PLUGIN_MODE	809	Uint16	"srv_plugin_mode - Server plug-in mode" on page 1088
srvcon_pw_plugin	No	No	None	SQLF_KTN_SRVCON_PW_PLUGIN	813	char(33)	"srvcon_pw_plugin - Userid-password plug-in for incoming connections at the server" on page 1088
start_stop_time	Yes	No	Low	SQLF_KTN_START_STOP_TIME	511	Uint16	
svcename	No	No	None	SQLF_KTN_SVCENAME	24	char(14)	"svcename - TCP/IP service name" on page 786
sysadm_group	No	No	None	SQLF_KTN_SYSADM_GROUP	39	char(16)	"sysadm_group - System administration authority group name" on page 787
sysctrl_group	No	No	None	SQLF_KTN_SYSCTRL_GROUP	63	char(16)	"sysctrl_group - System control authority group name" on page 788
sysmaint_group	No	No	None	SQLF_KTN_SYSMAINT_GROUP	62	char(16)	"sysmaint_group - System maintenance authority group name" on page 789
sysmon_group	No	No	None	SQLF_KTN_SYSMON	808	char(9)	"sysmon_group - System monitor authority group name" on page 790
tm_database	No	No	None	SQLF_KTN_TM_DATABASE	67	char(8)	
tp_mon_name	No	No	None	SQLF_KTN_TP_MON_NAME	66	char(19)	
tpname	No	No	None	SQLF_KTN_TPNAME	25	char(64)	
trust_allclnts[6]	No	No	None	SQLF_KTN_TRUST_ALLCLNTS	301	Uint16	"trust_allclnts - Trust all clients" on page 790

Table 64. Configurable Database Manager Configuration Parameters (continued)

Parameter	Cfg. Online	Auto.	Perf. Impact	Token	Token Value	Data Type	Additional Information
trust_clntauth	No	No	None	SQLF_KTN_TRUST_CLNTAUTH	302	Uint16	"trust_clntauth - Trusted clients authentication" on page 791
use_sna_auth	Yes	No	None	SQLF_KTN_USE_SNA_AUTH	805	Uint16	
util_impact_lim	Yes	No	High	SQLF_KTN_UTIL_IMPACT_LIM	807	Uint32	

Notes:

1. Valid values (defined in `sqlenv.h`):

```
SQL_AUTHENTICATION_SERVER (0)
SQL_AUTHENTICATION_CLIENT (1)
SQL_AUTHENTICATION_DCS (2)
SQL_AUTHENTICATION_DCE (3)
SQL_AUTHENTICATION_SVR_ENCRYPT (4)
SQL_AUTHENTICATION_DCS_ENCRYPT (5)
SQL_AUTHENTICATION_DCE_SVR_ENC (6)
SQL_AUTHENTICATION_KERBEROS (7)
SQL_AUTHENTICATION_KRB_SVR_ENC (8)
SQL_AUTHENTICATION_GSSPLUGIN (9)
SQL_AUTHENTICATION_GSS_SVR_ENC (10)
SQL_AUTHENTICATION_DATAENC (11)
SQL_AUTHENTICATION_DATAENC_CMP (12)
SQL_AUTHENTICATION_NOT_SPEC (255)
```

2. The *cpuspeed* parameter can have a significant impact on performance, but you should use the default value, except in very specific circumstances, as documented in the parameter description.
3.
```
Bit 1 (xxxx xxx1): dft_mon_uow
Bit 2 (xxxx xx1x): dft_mon_stmt
Bit 3 (xxxx x1xx): dft_mon_table
Bit 4 (xxxx 1xxx): dft_mon_buffpool
Bit 5 (xxx1 xxxx): dft_mon_lock
Bit 6 (xx1x xxxx): dft_mon_sort
Bit 7 (x1xx xxxx): dft_mon_timestamp
```

4. Valid values (defined in `sqlutil.h`):

```
SQLF_DSCVR_KNOWN (1)
SQLF_DSCVR_SEARCH (2)
```

5. Valid values (defined in `sqlutil.h`):

```
SQLF_INX_REC_SYSTEM (0)
SQLF_INX_REC_REFERENCE (1)
```

6. Valid values (defined in `sqlutil.h`):

```
SQLF_TRUST_ALLCLNTS_NO (0)
SQLF_TRUST_ALLCLNTS_YES (1)
SQLF_TRUST_ALLCLNTS_DRDAONLY (2)
```

Table 65. Informational Database Manager Configuration Parameters

Parameter	Token	Token Value	Data Type	Additional Information
nodetype[1]	SQLF_KTN_NODETYPE	100	Uint16	
release	SQLF_KTN_RELEASE	101	Uint16	

Notes:

1. Valid values (defined in `sqlutil.h`):

```
SQLF_NT_STANDALONE (0)
SQLF_NT_SERVER (1)
SQLF_NT_REQUESTOR (2)
SQLF_NT_STAND_REQ (3)
SQLF_NT_MPP (4)
SQLF_NT_SATELLITE (5)
```

Database Configuration Parameter Summary

The following table lists the parameters in the database configuration file. When changing the database configuration parameters, consider the detailed information for the parameter.

For some database configuration parameters, changes will only take effect when the database is reactivated. In these cases, all applications must first disconnect from the database. (If the database was activated, then it must be deactivated and reactivated.) The changes take effect at the next connection to the database. Other parameters can be changed online; these are called *configurable online configuration parameters*.

The column "Auto." in the following table indicates whether the parameter supports the AUTOMATIC keyword on the UPDATE DATABASE MANAGER CONFIGURATION command. If you set a parameter to automatic, DB2 will automatically adjust the parameter to reflect current resource requirements.

The column "Perf. Impact" provides an indication of the relative importance of each parameter as it relates to system performance. It is impossible for this column to apply accurately to all environments; you should view this information as a generalization.

- **High** — indicates that the parameter can have a significant impact on performance. You should consciously decide the values of these parameters, which, in some cases, means that you will accept the default values provided.
- **Medium** — indicates that the parameter can have some impact on performance. Your specific environment and needs will determine how much tuning effort should be focused on these parameters.
- **Low** — indicates that the parameter has a less general or less significant impact on performance.
- **None** — indicates that the parameter does not directly impact performance. Although you do not have to tune these parameters for performance enhancement, they can be very important for other aspects of your system configuration, such as communication support, for example.

The columns "Token", "Token Value", and "Data Type" provide information that you will need when calling the `db2CfgGet` or the `db2CfgSet` API. This information includes configuration parameter identifiers, entries for the *token* element in the *db2CfgParam* data structure, and data types for values that are passed to the structure.

Table 66. Configurable Database Configuration Parameters

Parameter	Cfg. Online	Auto.	Perf. Impact	Token	Token Value	Data Type	Additional Information
alt_collate	No	No	None	SQLF_DBTN_ALT_COLLATE	809	Uint32	
app_ctl_heap_sz	No	No	Medium	SQLF_DBTN_APP_CTL_HEAP_SZ	500	Uint16	
appgroup_mem_sz	No	No	Medium	SQLF_DBTN_APPGROUP_MEM_SZ	800	Uint32	
applheapsz	No	No	Medium	SQLF_DBTN_APPLHEAPSZ	51	Uint16	
archretrydelay	Yes	No	None	SQLF_DBTN_ARCHRETRYDELAY	828	Uint16	

Table 66. Configurable Database Configuration Parameters (continued)

Parameter	Cfg. Online	Auto.	Perf. Impact	Token	Token Value	Data Type	Additional Information
autonomic_switches • *auto_maint* • *auto_db_backup* • *auto_tbl_maint* • *auto_runstats* • *auto_stats_prof* • *auto_prof_upd* • *auto_reorg*	Yes	No	Medium	SQLF_DBTN_AUTONOMIC_SWITCHES[1] • SQLF_ENABLE_AUTO_MAINT • SQLF_ENABLE_AUTO_DB_BACKUP • SQLF_ENABLE_AUTO_TBL_MAINT • SQLF_ENABLE_AUTO_RUNSTATS • SQLF_ENABLE_AUTO_STATS_PROF • SQLF_ENABLE_AUTO_PROF_UPD • SQLF_ENABLE_AUTO_REORG	830 • 831 • 833 • 835 • 837 • 839 • 844 • 841	Uint32	
autorestart	Yes	No	Low	SQLF_DBTN_AUTO_RESTART	25	Uint16	"autorestart - Auto restart enable" on page 795
avg_appls	Yes	No	High	SQLF_DBTN_AVG_APPLS	47	Uint16	
blk_log_dsk_ful	Yes	No	None	SQLF_DBTN_BLK_LOG_DSK_FUL	804	Uint16	
catalogcache_sz	Yes	No	High	SQLF_DBTN_CATALOGCACHE_SZ	56	Sint32	
chngpgs_thresh	No	No	High	SQLF_DBTN_CHNGPGS_THRESH	38	Uint16	
database_memory	No	Yes	Medium	SQLF_DBTN_DATABASE_MEMORY	803	Uint64	
dbheap	Yes	No	Medium	SQLF_DBTN_DB_HEAP	58	Uint64	
dft_degree	Yes	No	High	SQLF_DBTN_DFT_DEGREE	301	Sint32	
dft_extent_sz	Yes	No	Medium	SQLF_DBTN_DFT_EXTENT_SZ	54	Uint32	
dft_loadrec_ses	Yes	No	Medium	SQLF_DBTN_DFT_LOADREC_SES	42	Sint16	
dft_mttb_types	No	No	None	SQLF_DBTN_DFT_MTTB_TYPES	843	Uint32	
dft_prefetch_sz	Yes	Yes	Medium	SQLF_DBTN_DFT_PREFETCH_SZ	40	Sint16	
dft_queryopt	Yes	No	Medium	SQLF_DBTN_DFT_QUERYOPT	57	Sint32	
dft_refresh_age	No	No	Medium	SQLF_DBTN_DFT_REFRESH_AGE	702	char(22)	
dft_sqlmathwarn	No	No	None	SQLF_DBTN_DFT_SQLMATHWARN	309	Sint16	
discover_db	Yes	No	Medium	SQLF_DBTN_DISCOVER	308	Uint16	
dl_expint	Yes	No	None	SQLF_DBTN_DL_EXPINT	350	Sint32	
dl_num_copies	Yes	No	None	SQLF_DBTN_DL_NUM_COPIES	351	Uint16	
dl_time_drop	Yes	No	None	SQLF_DBTN_DL_TIME_DROP	353	Uint16	
dl_token	Yes	No	Low	SQLF_DBTN_DL_TOKEN	602	char(10)	
dl_upper	Yes	No	None	SQLF_DBTN_DL_UPPER	603	Sint16	
dl_wt_iexpint	Yes	No	None	SQLF_DBTN_DL_WT_IEXPINT	354	Sint32	
dlchktime	Yes	No	Medium	SQLF_DBTN_DLCHKTIME	9	Uint32	"dlchktime - Time interval for checking deadlock" on page 792
dyn_query_mgmt	No	No	Low	SQLF_DBTN_DYN_QUERY_MGMT	604	Uint16	
estore_seg_sz	No	No	Medium	SQLF_DBTN_ESTORE_SEG_SZ	303	Sint32	
failarchpath	Yes	No	None	SQLF_DBTN_FAILARCHPATH	826	char(243)	
groupheap_ratio	No	No	Medium	SQLF_DBTN_GROUPHEAP_RATIO	801	Uint16	
hadr_local_host	No	No	None	SQLF_DBTN_HADR_LOCAL_HOST	811	char(256)	
hadr_local_svc	No	No	None	SQLF_DBTN_HADR_LOCAL_SVC	812	char(41)	
hadr_remote_host	No	No	None	SQLF_DBTN_HADR_REMOTE_HOST	813	char(256)	
hadr_remote_inst	No	No	None	SQLF_DBTN_HADR_REMOTE_INST	815	char(9)	
hadr_remote_svc	No	No	None	SQLF_DBTN_HADR_REMOTE_SVC	814	char(41)	
hadr_syncmode	No	No	None	SQLF_DBTN_HADR_SYNCMODE	817	Uint32	
hadr_timeout	No	No	None	SQLF_DBTN_HADR_TIMEOUT	816	Sint32	
indexrec[2]	Yes	No	Medium	SQLF_DBTN_INDEXREC	30	Uint16	
locklist	Yes	No	High when it affects escalation	SQLF_DBTN_LOCK_LIST	704	Uint64	
locktimeout	No	No	Medium	SQLF_DBTN_LOCKTIMEOUT	34	Sint16	"locktimeout - Lock timeout" on page 793

Table 66. Configurable Database Configuration Parameters (continued)

Parameter	Cfg. Online	Auto.	Perf. Impact	Token	Token Value	Data Type	Additional Information
logarchmeth1	Yes	No	None	SQLF_DBTN_LOGARCHMETH1	822	Uint16	
logarchmeth2	Yes	No	None	SQLF_DBTN_LOGARCHMETH2	823	Uint16	
logarchopt1	Yes	No	None	SQLF_DBTN_LOGARCHOPT1	824	char(243)	
logarchopt2	Yes	No	None	SQLF_DBTN_LOGARCHOPT2	825	char(243)	
logbufsz	No	No	High	SQLF_DBTN_LOGBUFSZ	33	Uint16	
logfilsiz	No	No	Medium	SQLF_DBTN_LOGFIL_SIZ	92	Uint32	
logindexbuild	Yes	Yes	None	SQLF_DBTN_LOGINDEXBUILD	818	Uint32	
logprimary	No	No	Medium	SQLF_DBTN_LOGPRIMARY	16	Uint16	
logretain[3]	No	No	Low	SQLF_DBTN_LOG_RETAIN	23	Uint16	
logsecond	Yes	No	Medium	SQLF_DBTN_LOGSECOND	17	Uint16	
max_log	Yes	Yes		SQLF_DBTN_MAX_LOG	807	Uint16	
maxappls	Yes	Yes	Medium	SQLF_DBTN_MAXAPPLS	6	Uint16	
maxfilop	Yes	No	Medium	SQLF_DBTN_MAXFILOP	3	Uint16	
maxlocks	Yes	No	High when it affects escalation	SQLF_DBTN_MAXLOCKS	15	Uint16	"maxlocks - Maximum percent of lock list before escalation" on page 794
min_dec_div_3	No	No	High	SQLF_DBTN_MIN_DEC_DIV_3	605	Sint32	
mincommit	Yes	No	High	SQLF_DBTN_MINCOMMIT	32	Uint16	
mirrorlogpath	No	No	Low	SQLF_DBTN_MIRRORLOGPATH	806	char(242)	
newlogpath	No	No	Low	SQLF_DBTN_NEWLOGPATH	20	char(242)	
num_db_backups	Yes	No	None	SQLF_DBTN_NUM_DB_BACKUPS	601	Uint16	
num_estore_segs	No	No	Medium	SQLF_DBTN_NUM_ESTORE_SEGS	304	Sint32	
num_freqvalues	Yes	No	Low	SQLF_DBTN_NUM_FREQVALUES	36	Uint16	
num_iocleaners	No	No	High	SQLF_DBTN_NUM_IOCLEANERS	37	Uint16	
num_ioservers	No	No	High	SQLF_DBTN_NUM_IOSERVERS	39	Uint16	
num_log_span	Yes	Yes		SQLF_DBTN_NUM_LOG_SPAN	808	Uint16	
num_quantiles	Yes	No	Low	SQLF_DBTN_NUM_QUANTILES	48	Uint16	
numarchretry	Yes	No	None	SQLF_DBTN_NUMARCHRETRY	827	Uint16	
overflowlogpath	No	No	Medium	SQLF_DBTN_OVERFLOWLOGPATH	805	char(242)	
pckcachesz	Yes	No	High	SQLF_DBTN_PCKCACHE_SZ	505	Uint32	
rec_his_retentn	No	No	None	SQLF_DBTN_REC_HIS_RETENTN	43	Sint16	
seqdetect	Yes	No	High	SQLF_DBTN_SEQDETECT	41	Uint16	
sheapthres_shr	No	No	High	SQLF_DBTN_SHEAPTHRES_SHR	802	Uint32	
softmax	No	No	Medium	SQLF_DBTN_SOFTMAX	5	Uint16	
sortheap	Yes	No	High	SQLF_DBTN_SORT_HEAP	52	Uint32	
stat_heap_sz	No	No	Low	SQLF_DBTN_STAT_HEAP_SZ	45	Uint32	
stmtheap	Yes	No	Medium	SQLF_DBTN_STMT_HEAP	821	Uint32	
trackmod	No	No	Low	SQLF_DBTN_TRACKMOD	703	Uint16	
tsm_mgmtclass	Yes	No	None	SQLF_DBTN_TSM_MGMTCLASS	307	char(30)	
tsm_nodename	Yes	No	None	SQLF_DBTN_TSM_NODENAME	306	char(64)	
tsm_owner	Yes	No	None	SQLF_DBTN_TSM_OWNER	305	char(64)	
tsm_password	Yes	No	None	SQLF_DBTN_TSM_PASSWORD	501	char(64)	
userexit	No	No	Low	SQLF_DBTN_USER_EXIT	24	Uint16	
util_heap_sz	Yes	No	Low	SQLF_DBTN_UTIL_HEAP_SZ	55	Uint32	
vendoropt	Yes	No	None	SQLF_DBTN_VENDOROPT	829	char(242)	

Table 66. Configurable Database Configuration Parameters (continued)

Parameter	Cfg. Online	Auto.	Perf. Impact	Token	Token Value	Data Type	Additional Information

Notes:

1. Default => Bit 1 on (xxxx xxxx xxxx xxx1): auto_maint
   Bit 2 off (xxxx xxxx xxxx xx0x): auto_db_backup
   Bit 3 on (xxxx xxxx xxxx x0xx): auto_tbl_maint
   Bit 4 on (xxxx xxxx xxxx 1xxx): auto_runstats
   Bit 5 off (xxxx xxxx xxx1 xxxx): auto_stats_prof
   Bit 6 off (xxxx xxxx xx0x xxxx): auto_prof_upd
   Bit 7 off (xxxx xxxx x0xx xxxx): auto_reorg
   0 0 1 9

   Maximum => Bit 1 on (xxxx xxxx xxxx xxx1): auto_maint
   Bit 2 off (xxxx xxxx xxxx xx1x): auto_db_backup
   Bit 3 on (xxxx xxxx xxxx x1xx): auto_tbl_maint
   Bit 4 on (xxxx xxxx xxxx 1xxx): auto_runstats
   Bit 5 off (xxxx xxxx xxx1 xxxx): auto_stats_prof
   Bit 6 off (xxxx xxxx xx1x xxxx): auto_prof_upd
   Bit 7 off (xxxx xxxx x1xx xxxx): auto_reorg
   0 0 7 F
2. Valid values (defined in sqlutil.h):

   ```
   SQLF_INX_REC_SYSTEM (0)
   SQLF_INX_REC_REFERENCE (1)
   SQLF_INX_REC_RESTART (2)
   ```

3. Valid values (defined in sqlutil.h):

   ```
   SQLF_LOGRETAIN_NO (0)
   SQLF_LOGRETAIN_RECOVERY (1)
   SQLF_LOGRETAIN_CAPTURE (2)
   ```

Table 67. Informational Database Configuration Parameters

Parameter	Token	Token Value	Data Type	Additional Information
backup_pending	SQLF_DBTN_BACKUP_PENDING	112	Uint16	
codepage	SQLF_DBTN_CODEPAGE	101	Uint16	
codeset	SQLF_DBTN_CODESET	120	char(9)[1]	
collate_info	SQLF_DBTN_COLLATE_INFO	44	char(260)	
country	SQLF_DBTN_COUNTRY	100	Uint16	
database_consistent	SQLF_DBTN_CONSISTENT	111	Uint16	"database_consistent - Database is consistent" on page 796
database_level	SQLF_DBTN_DATABASE_LEVEL	124	Uint16	
hadr_db_role	SQLF_DBTN_HADR_DB_ROLE	810	Uint32	
log_retain_status	SQLF_DBTN_LOG_RETAIN_STATUS	114	Uint16	
loghead	SQLF_DBTN_LOGHEAD	105	char(12)	
logpath	SQLF_DBTN_LOGPATH	103	char(242)	
multipage_alloc	SQLF_DBTN_MULTIPAGE_ALLOC	506	Uint16	
numsegs	SQLF_DBTN_NUMSEGS	122	Uint16	
release	SQLF_DBTN_RELEASE	102	Uint16	
restore_pending	SQLF_DBTN_RESTORE_PENDING	503	Uint16	
rollfwd_pending	SQLF_DBTN_ROLLFWD_PENDING	113	Uint16	
territory	SQLF_DBTN_TERRITORY	121	char(5)[2]	
user_exit_status	SQLF_DBTN_USER_EXIT_STATUS	115	Uint16	

Table 67. Informational Database Configuration Parameters (continued)

Parameter	Token	Token Value	Data Type	Additional Information
Notes: 1. char(17) on HP-UX and Solaris Operating Environment. 2. char(33) on HP-UX and Solaris Operating Environment.				

DB2 Administration Server (DAS) Configuration Parameter Summary

Table 68. DAS Configuration Parameters

Parameter	Parameter Type	Additional Information
authentication	Configurable	"authentication - Authentication type DAS" on page 784
contact_host	Configurable Online	
das_codepage	Configurable Online	
das_territory	Configurable Online	
dasadm_group	Configurable	"dasadm_group - DAS administration authority group name" on page 785
db2system	Configurable Online	
discover	Configurable Online	
exec_exp_task	Configurable	
jdk_64_path	Configurable Online	
jdk_path	Configurable Online	
sched_enable	Configurable	
sched_userid	Informational	
smtp_server	Configurable Online	
toolscat_db	Configurable	
toolscat_inst	Configurable	
toolscat_schema	Configurable	

Related concepts:

- "Indirect privileges through a package containing nicknames" on page 48
- "Effect of DYNAMICRULES bind option on dynamic SQL" on page 952

Related reference:

- "BIND" on page 232

Configuring DB2 with configuration parameters

Database manager configuration parameters are stored in a file named `db2systm`. Database configuration parameters are stored in a file named `SQLDBCON`. These files cannot be directly edited, and can only be changed or viewed via a supplied API or by a tool which calls that API.

Attention: If you edit `db2systm` or `SQLDBCON` using a method other than those provided by DB2, you may make the database unusable. **We strongly recommend** that you do not change these files using methods other than those documented and supported by DB2.

You may use one of the following methods to reset, update, and view configuration parameters:

- Using the Control Center. The Configure Instance notebook can be used to set the database manager configuration parameters on either a client or a server. The Configure Database notebook can be used to alter the value of database configuration parameters. The DB2 Control Center also provides the Configuration Advisor to alter the value of configuration parameters. This advisor generates values for parameters based on the responses you provide to a set of questions, such as the workload and the type of transactions that run against the database.

  In a partitioned database environment, the `SQLDBCON` file exists for each database partition. The Configure Database notebook will change the value on all partitions if you launch the notebook from the database object in the tree view of the Control Center. If you launch the notebook from a database partition object, then it will only change the values for that partition. (We recommend, however, that the configuration parameter values be the same on all partitions.)

  Note: The Configuration Advisor is not available in the partitioned database environment.
- Using the command line processor. Commands to change the settings can be quickly and conveniently entered:

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

  For database configuration parameters:
  - GET DATABASE CONFIGURATION (or GET DB CFG)
  - UPDATE DATABASE CONFIGURATION (or UPDATE DB CFG)
  - RESET DATABASE CONFIGURATION (or RESET DB CFG) to reset *all* database parameters to their default values
  - AUTOCONFIGURE.
- Using the application programming interfaces (APIs). The APIs can easily be called from an application or a host-language program.
- Using the Configuration Assistant (for database manager configuration parameters). You can only use the Configuration Assistant to set the database manager configuration parameters on a client.

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

For some database parameters, changes will only take effect when the database is reactivated. In these cases, all applications must first disconnect from the database.

(If the database was activated, then it must be deactivated and reactivated.) Then, at the first new connect to the database, the changes will take effect.

Other parameters can be changed online; these are called *configurable online configuration parameters*.

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

To change a database manager configuration parameter online:

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

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

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

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

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

- **Immediate**: Parameters that change immediately upon command or API invocation. For example, *diaglevel* has a propagation class of immediate.
- **Statement boundary**: Parameters that change on statement and statement-like boundaries. For example, if you change the value of *sortheap*, all new SQL requests will start using the new value.
- **Transaction boundary**: Parameters that change on transaction boundaries. For example, a new value for *dl_expint* is updated after a COMMIT statement.

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

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

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

Related concepts:

- "Configuration parameters" on page 769
- "Configuration parameter tuning" in the *Administration Guide: Performance*

Related reference:

- "GET DATABASE CONFIGURATION" on page 275
- "GET DATABASE MANAGER CONFIGURATION" on page 281
- "RESET DATABASE CONFIGURATION Command" in the *Command Reference*
- "RESET DATABASE MANAGER CONFIGURATION Command" in the *Command Reference*
- "UPDATE DATABASE CONFIGURATION" on page 381
- "UPDATE DATABASE MANAGER CONFIGURATION" on page 384
- "Configuration parameters summary" in the *Administration Guide: Performance*
- "FLUSH PACKAGE CACHE statement" in the *SQL Reference, Volume 2*

Security-Related Configuration Parameters

audit_buf_sz - Audit buffer size

Configuration Type	Database manager
Applies To	• Database server with local and remote clients • Database server with local clients • Partitioned database server with local and remote clients
Parameter Type	Configurable
Default [Range]	0 [0 – 65 000]
Unit of Measure	Pages (4 KB)
When Allocated	When DB2 is started
When Freed	When DB2 is stopped

This parameter specifies the size of the buffer used when auditing the database.

The default value for this parameter is zero (0). If the value is zero (0), the audit buffer is not used. If the value is greater than zero (0), space is allocated for the audit buffer where the audit records will be placed when they are generated by the audit facility. The value times 4 KB pages is the amount of space allocated for the audit buffer. The audit buffer cannot be allocated dynamically; DB2 must be stopped and then restarted before the new value for this parameter takes effect.

By changing this parameter from the default to some value larger than zero (0), the audit facility writes records to disk asynchronously compared to the execution of the statements generating the audit records. This improves DB2 performance over

leaving the parameter value at zero (0). The value of zero (0) means the audit facility writes records to disk synchronously with (at the same time as) the execution of the statements generating the audit records. The synchronous operation during auditing decreases the performance of applications running in DB2.

Related reference:

- "GET DATABASE MANAGER CONFIGURATION" on page 281
- "RESET DATABASE MANAGER CONFIGURATION Command" in the *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION" on page 384

authentication - Authentication type

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; KERBEROS; KRB_SERVER_ENCRYPT; GSSPLUGIN; GSS_SERVER_ENCRYPT]

This parameter specifies and determines how and where authentication of a user takes place.

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 passwords sent over the network are encrypted.

Note: For a Common Criteria compliant 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.

Related reference:

- "GET DATABASE MANAGER CONFIGURATION" on page 281
- "RESET DATABASE MANAGER CONFIGURATION Command" in the *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION" on page 384

authentication - Authentication type DAS

Configuration Type	DB2 Administration Server
Applies to	DB2 Administration Server
Parameter Type	Configurable
Default [Range]	SERVER_ENCRYPT [SERVER_ENCRYPT; KERBEROS_ENCRYPT]

This parameter determines how and where authentication of a user takes place.

If authentication is SERVER_ENCRYPT, then the user ID and password are sent from the client to the server so authentication can take place on the server. Passwords sent over the network are encrypted.

A value of KERBEROS_ENCRYPT means that authentication is performed at a Kerberos server using the Kerberos security protocol for authentication.

Note: The KERBEROS_ENCRYPT authentication type is only supported on servers running Windows 2000.

This parameter can only be updated from a Version 8 command line processor (CLP).

Related reference:

- "GET ADMIN CONFIGURATION Command" in the *Command Reference*
- "RESET ADMIN CONFIGURATION Command" in the *Command Reference*
- "UPDATE ADMIN CONFIGURATION Command" in the *Command Reference*

catalog_noauth - Cataloging allowed without authority

Configuration Type	Database manager
Applies to	• Database server with local and remote clients • Client • Database server with local clients • Partitioned database server with local and remote clients
Parameter Type	Configurable Online
Propagation Class	Immediate

Default [Range]

Database server with local and remote clients
NO [NO (0) — YES (1)]

Client; Database server with local clients
YES [NO (0) — YES (1)]

This parameter specifies whether users are able to catalog and uncatalog databases and nodes, or DCS and ODBC directories, without SYSADM authority. The default value (0) for this parameter indicates that SYSADM authority is required. When this parameter is set to 1 (yes), SYSADM authority is not required.

Related reference:

- "GET DATABASE MANAGER CONFIGURATION" on page 281
- "RESET DATABASE MANAGER CONFIGURATION Command" in the *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION" on page 384

dasadm_group - DAS administration authority group name

Configuration Type	DB2 Administration Server
Applies to	DB2 Administration Server
Parameter Type	Configurable
Default [Range]	Null [any valid group name]

DAS Administration (DASADM) authority is the highest level of authority within the DAS. This parameter defines the group name with DASADM authority for the DAS.

DASADM authority is determined by the security facilities used in a specific operating environment.

- For the Windows NT and Windows 2000 operating systems, this parameter can be set to any local group that has a name of 8 characters or fewer, and is defined in the Windows NT and Windows 2000 security database. If "NULL" is specified for this parameter, all members of the Administrators group have DASADM authority.
- For UNIX-based systems, if "NULL" is specified as the value of this parameter, the DASADM group defaults to the primary group of the instance owner.

  If the value is not "NULL", the DASADM group can be any valid UNIX group name.

This parameter can only be updated from a Version 8 command line processor (CLP).

Related reference:

- "GET ADMIN CONFIGURATION Command" in the *Command Reference*
- "RESET ADMIN CONFIGURATION Command" in the *Command Reference*
- "UPDATE ADMIN CONFIGURATION Command" in the *Command Reference*

dftdbpath - Default database path

Configuration Type	Database manager

Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type Configurable Online

Propagation Class Immediate

Default [Range]

UNIX	Home directory of instance owner [any existing path]
Windows	Drive on which DB2 is installed [any existing path]

This parameter contains the default file path used to create databases under the database manager. If no path is specified when a database is created, the database is created under the path specified by the *dftdbpath* parameter.

In a partitioned database environment, you should ensure that the path on which the database is being created is not an NFS-mounted path (on UNIX-based platforms), or a network drive (in a Windows environment). The specified path must physically exist on each database partition server. To avoid confusion, it is best to specify a path that is locally mounted on each database partition server. The maximum length of the path is 205 characters. The system appends the node name to the end of the path.

Given that databases can grow to a large size and that many users could be creating databases (depending on your environment and intentions), it is often convenient to be able to have all databases created and stored in a specified location. It is also good to be able to isolate databases from other applications and data both for integrity reasons and for ease of backup and recovery.

For UNIX-based environments, the length of the *dftdbpath* name cannot exceed 215 characters and must be a valid, absolute, path name. For Windows, the *dftdbpath* can be a drive letter, optionally followed by a colon.

Recommendation: If possible, put high volume databases on a different disk than other frequently accessed data, such as the operating system files and the database logs.

Related reference:

- "GET DATABASE MANAGER CONFIGURATION" on page 281
- "RESET DATABASE MANAGER CONFIGURATION Command" in the *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION" on page 384

svcename - TCP/IP service name

Configuration Type Database manager

Applies to

- Database server with local and remote clients
- Database server with local clients

- Partitioned database server with local and remote clients

Parameter Type Configurable

Default Null

This 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 first of two consecutive ports reserved for use by the database manager; the second port is used to handle interrupt requests from down-level clients.

In order to accept connection requests from a database 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. If the database server needs to support requests from down-level clients, a second port (number *n+1*, for interrupt requests) needs to be defined 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. Down-level clients also require the interrupt port (number *n+1*) to be defined in the client's services file.

On UNIX-based 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. If you are supporting or using a down-level client, the service name for the interrupt port is not saved in the configuration file. The interrupt port number can be derived based on the main connection port number (*interrupt port number = main connection port* + 1).

Related reference:

- "GET DATABASE MANAGER CONFIGURATION" on page 281
- "RESET DATABASE MANAGER CONFIGURATION Command" in the *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION" on page 384

sysadm_group - System administration authority group name

Configuration Type Database manager

Applies to

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type Configurable

Default Null

System administration (SYSADM) authority is the highest level of authority within the database manager and controls all database objects. This parameter defines the group name with SYSADM authority for the database manager instance.

SYSADM authority is determined by the security facilities used in a specific operating environment.

- In the Windows 98 operating system the SYSADM group must be NULL.

  This parameter must be "NULL" for Windows 98 clients when system security is used because the Windows 98 operating system does not store group information, thereby providing no way of determining if a user is a member of a designated SYSADM group. When a group name is specified, no user can be a member of it.
- For the Windows NT and Windows 2000 operating system, this parameter can be set to any local group that has a name of 8 characters or fewer, and is defined in the Windows NT and Windows 2000 security database. If "NULL" is specified for this parameter, all members of the Administrators group have SYSADM authority.
- For UNIX-based systems, if "NULL" is specified as the value of this parameter, the SYSADM group defaults to the primary group of the instance owner.

  If the value is not "NULL", the SYSADM group can be any valid UNIX group name.

To restore the parameter to its default (NULL) value, use UPDATE DBM CFG USING SYSADM_GROUP NULL. You must specify the keyword "NULL" in uppercase. You can also use the Configure Instance notebook in the DB2 Control Center.

Related reference:

- "sysctrl_group - System control authority group name" on page 788
- "sysmaint_group - System maintenance authority group name" on page 789
- "GET DATABASE MANAGER CONFIGURATION" on page 281
- "RESET DATABASE MANAGER CONFIGURATION Command" in the *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION" on page 384

sysctrl_group - System control authority group name

Configuration Type	Database manager
Applies to	• Database server with local and remote clients • Client • Database server with local clients • Partitioned database server with local and remote clients
Parameter Type	Configurable
Default	Null

This parameter defines the group name with system control (SYSCTRL) authority. SYSCTRL has privileges allowing operations affecting system resources, but does not allow direct access to data.

Attention: This parameter must be NULL for Windows 98 clients when system security is used (that is, authentication is CLIENT, SERVER, DCS, or any other valid authentication). This is because the Windows 98 operating systems do not

store group information, thereby providing no way of determining if a user is a member of a designated SYSCTRL group. When a group name is specified, no user can be a member of it.

To restore the parameter to its default (NULL) value, use UPDATE DBM CFG USING SYSCTRL_GROUP NULL. You must specify the keyword "NULL" in uppercase. You can also use the Configure Instance notebook in the DB2 Control Center.

Related reference:

- "sysadm_group - System administration authority group name" on page 787
- "sysmaint_group - System maintenance authority group name" on page 789
- "GET DATABASE MANAGER CONFIGURATION" on page 281
- "RESET DATABASE MANAGER CONFIGURATION Command" in the *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION" on page 384

sysmaint_group - System maintenance authority group name

Configuration Type Database manager

Applies to

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type Configurable

Default Null

This parameter defines the group name with system maintenance (SYSMAINT) authority. SYSMAINT has privileges to perform maintenance operations on all databases associated with an instance without having direct access to data.

Attention: This parameter must be NULL for Windows 98 clients when system security is used (that is, authentication is CLIENT, SERVER, DCS, or any other valid authentication). This is because the Windows 98 operating systems do not store group information, thereby providing no way of determining if a user is a member of a designated SYSMAINT group. When a group name is specified, no user can be a member of it.

To restore the parameter to its default (NULL) value, use UPDATE DBM CFG USING SYSMAINT_GROUP NULL. You must specify the keyword "NULL" in uppercase. You can also use the Configure Instance notebook in the DB2 Control Center.

Related reference:

- "sysadm_group - System administration authority group name" on page 787
- "sysctrl_group - System control authority group name" on page 788
- "GET DATABASE MANAGER CONFIGURATION" on page 281
- "RESET DATABASE MANAGER CONFIGURATION Command" in the *Command Reference*

- "UPDATE DATABASE MANAGER CONFIGURATION" on page 384

sysmon_group - System monitor authority group name

Configuration Type Database manager

Applies to

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type Configurable

Default Null

This parameter defines the group name with system monitor (SYSMON) authority. Users having SYSMON authority at the instance level have the ability to take database system monitor snapshots of a database manager instance or its databases. SYSMON authority includes the ability to use the following commands:

- GET DATABASE MANAGER MONITOR SWITCHES
- GET MONITOR SWITCHES
- GET SNAPSHOT
- LIST ACTIVE DATABASES
- LIST APPLICATIONS
- LIST DCS APPLICATIONS
- RESET MONITOR
- UPDATE MONITOR SWITCHES

Users with SYSADM, SYSCTRL, or SYSMAINT authority automatically have the ability to take database system monitor snapshots and to use these commands.

Attention: This parameter must be NULL for Windows 98 clients when system security is used (that is, authentication is CLIENT, SERVER, DCS, or any other valid authentication). This is because the Windows 98 operating systems do not store group information, thereby providing no way of determining if a user is a member of a designated SYSMON group. When a group name is specified, no user can be a member of it.

To restore the parameter to its default (NULL) value, use UPDATE DBM CFG USING SYSMON_GROUP NULL. You must specify the keyword "NULL" in uppercase. You can also use the Configure Instance notebook in the DB2 Control Center.

Related reference:

- "GET DATABASE MANAGER CONFIGURATION" on page 281
- "RESET DATABASE MANAGER CONFIGURATION Command" in the *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION" on page 384

trust_allclnts - Trust all clients

Configuration Type Database manager

Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type Configurable

Default [Range] YES [NO, YES, DRDAONLY]

This parameter is only active when the *authentication* parameter is set to CLIENT.

This parameter and *trust_clntauth* are used to determine where users are validated to the database environment.

By accepting the default of "YES" for this parameter, all clients are treated as trusted clients. This means that the server assumes that a level of security is available at the client and the possibility that users can be validated at the client.

This parameter can only be changed to "NO" if the *authentication* parameter is set to CLIENT. If this parameter is set to "NO", the untrusted clients must provide a userid and password combination when they connect to the server. Untrusted clients are operating system platforms that do not have a security subsystem for authenticating users.

Setting this parameter to "DRDAONLY" protects against all clients except clients from DB2 for OS/390 and z/OS, DB2 for VM and VSE, and DB2 for OS/400. Only these clients can be trusted to perform client-side authentication. All other clients must provide a user ID and password to be authenticated by the server.

When *trust_allclnts* is set to "DRDAONLY", the *trust_clntauth* parameter is used to determine where the clients are authenticated. If *trust_clntauth* is set to "CLIENT", authentication occurs at the client. If *trust_clntauth* is set to "SERVER", authentication occurs at the client if no password is provided, and at the server if a password is provided.

Related reference:

- "authentication - Authentication type" on page 783
- "trust_clntauth - Trusted clients authentication" on page 791
- "GET DATABASE MANAGER CONFIGURATION" on page 281
- "RESET DATABASE MANAGER CONFIGURATION Command" in the *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION" on page 384

trust_clntauth - Trusted clients authentication

Configuration Type Database manager

Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type Configurable

Default [Range] CLIENT [CLIENT, SERVER]

This parameter specifies whether a trusted client is authenticated at the server or the client when the client provides a userid and password combination for a connection. This parameter (and *trust_allclnts*) is only active if the *authentication* parameter is set to CLIENT. If a user ID and password are not provided, the client is assumed to have validated the user, and no further validation is performed at the server.

If this parameter is set to CLIENT (the default), the trusted client can connect without providing a user ID and password combination, and the assumption is that the operating system has already authenticated the user. If it is set to SERVER, the user ID and password will be validated at the server.

The numeric value for CLIENT is 0. The numeric value for SERVER is 1.

Related reference:

- "authentication - Authentication type" on page 783
- "trust_allclnts - Trust all clients" on page 790
- "GET DATABASE MANAGER CONFIGURATION" on page 281
- "RESET DATABASE MANAGER CONFIGURATION Command" in the *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION" on page 384

Locking Configuration Parameters

dlchktime - Time interval for checking deadlock

Configuration Type	Database
Parameter Type	Configurable online
Propagation Class	Immediate
Default [Range]	10 000 (10 seconds) [1 000 – 600 000]
Unit of Measure	Milliseconds

A deadlock occurs when two or more applications connected to the same database wait indefinitely for a resource. The waiting is never resolved because each application is holding a resource that the other needs to continue.

The deadlock check interval defines the frequency at which the database manager checks for deadlocks among all the applications connected to a database.

Notes:

1. In a partitioned database environment, this parameter applies to the catalog node only.
2. In a partitioned database environment, a deadlock is not flagged until after the second iteration.

Recommendation: Increasing this parameter decreases the frequency of checking for deadlocks, thereby increasing the time that application programs must wait for the deadlock to be resolved.

Decreasing this parameter increases the frequency of checking for deadlocks, thereby decreasing the time that application programs must wait for the deadlock to be resolved but increasing the time that the database manager takes to check for deadlocks. If the deadlock interval is too small, it can decrease run-time performance, because the database manager is frequently performing deadlock detection. If this parameter is set lower to improve concurrency, you should ensure that *maxlocks* and *locklist* are set appropriately to avoid unnecessary lock escalation, which can result in more lock contention and as a result, more deadlock situations.

Related reference:

- "locklist - Maximum storage for lock list configuration parameter" in the *Administration Guide: Performance*
- "maxlocks - Maximum percent of lock list before escalation" on page 794
- "GET DATABASE CONFIGURATION" on page 275
- "RESET DATABASE CONFIGURATION Command" in the *Command Reference*
- "UPDATE DATABASE CONFIGURATION" on page 381

locktimeout - Lock timeout

Configuration Type	Database
Parameter Type	Configurable
Default [Range]	-1 [-1; 0 – 32 767]
Unit of Measure	Seconds

This parameter specifies the number of seconds that an application will wait to obtain a lock. This helps avoid global deadlocks for applications.

If you set this parameter to `0`, locks are not waited for. In this situation, if no lock is available at the time of the request, the application immediately receives a -911.

If you set this parameter to `-1`, lock timeout detection is turned off. In this situation a lock will be waited for (if one is not available at the time of the request) until either of the following:

- The lock is granted
- A deadlock occurs.

Recommendation: In a transaction processing (OLTP) environment, you can use an initial starting value of 30 seconds. In a query-only environment you could start with a higher value. In both cases, you should use benchmarking techniques to tune this parameter.

When working with Data Links Manager, if you see lock timeouts in the administration notification log of the Data Links Manager (dlfm) instance, then you should increase the value of *locktimeout*. You should also consider increasing the value of *locklist*.

The value should be set to quickly detect waits that are occurring because of an abnormal situation, such as a transaction that is stalled (possibly as a result of a user leaving their workstation). You should set it high enough so valid lock requests do not time-out because of peak workloads, during which time, there is more waiting for locks.

You can use the database system monitor to help you track the number of times an application (connection) experienced a lock timeout or that a database detected a timeout situation for all applications that were connected.

High values of the *lock_timeout* (number of lock timeouts) monitor element can be caused by:

- Too low a value for this configuration parameter.
- An application (transaction) that is holding locks for an extended period. You can use the database system monitor to further investigate these applications.
- A concurrency problem, that could be caused by lock escalations (from row-level to a table-level lock).

Related reference:

- "locklist - Maximum storage for lock list configuration parameter" in the *Administration Guide: Performance*
- "maxlocks - Maximum percent of lock list before escalation" on page 794
- "lock_timeouts - Number of Lock Timeouts monitor element" in the *System Monitor Guide and Reference*
- "GET DATABASE CONFIGURATION" on page 275
- "RESET DATABASE CONFIGURATION Command" in the *Command Reference*
- "UPDATE DATABASE CONFIGURATION" on page 381

maxlocks - Maximum percent of lock list before escalation

Configuration Type	Database	
Parameter Type	Configurable online	
Propagation Class	Immediate	
Default [Range]		
	UNIX	10 [1 – 100]
	Windows	22 [1 – 100]
Unit of Measure	Percentage	

Lock escalation is the process of replacing row locks with table locks, reducing the number of locks in the list. This parameter defines a percentage of the lock list held by an application that must be filled before the database manager performs escalation. When the number of locks held by any one application reaches this percentage of the total lock list size, lock escalation will occur for the locks held by that application. Lock escalation also occurs if the lock list runs out of space.

The database manager determines which locks to escalate by looking through the lock list for the application and finding the table with the most row locks. If after replacing these with a single table lock, the *maxlocks* value is no longer exceeded, lock escalation will stop. If not, it will continue until the percentage of the lock list held is below the value of *maxlocks*. The *maxlocks* parameter multiplied by the *maxappls* parameter cannot be less than 100.

Recommendation: The following formula allows you to set *maxlocks* to allow an application to hold twice the average number of locks:

```
maxlocks = 2 * 100 / maxappls
```

Where 2 is used to achieve twice the average and 100 represents the largest percentage value allowed. If you have only a few applications that run concurrently, you could use the following formula as an alternative to the first formula:

```
   maxlocks = 2 * 100 / (average number of applications running
concurrently)
```

One of the considerations when setting *maxlocks* is to use it in conjunction with the size of the lock list (*locklist*). The actual limit of the number of locks held by an application before lock escalation occurs is:

```
maxlocks * locklist * 4 096 / (100 * 36) on a 32-bit system

maxlocks * locklist * 4 096 / (100 * 56) on a 64-bit system
```

Where 4 096 is the number of bytes in a page, 100 is the largest percentage value allowed for *maxlocks*, and 36 is the number of bytes per lock on a 32-bit system, and 56 is the number of bytes per lock on a 64-bit system. If you know that one of your applications requires 1 000 locks, and you do not want lock escalation to occur, then you should choose values for *maxlocks* and *locklist* in this formula so that the result is greater than 1 000. (Using 10 for *maxlocks* and 100 for *locklist*, this formula results in greater than the 1 000 locks needed.)

If *maxlocks* is set too low, lock escalation happens when there is still enough lock space for other concurrent applications. If *maxlocks* is set too high, a few applications can consume most of the lock space, and other applications will have to perform lock escalation. The need for lock escalation in this case results in poor concurrency.

You can use the database system monitor to help you track and tune this configuration parameter.

Related reference:

- "locklist - Maximum storage for lock list configuration parameter" in the *Administration Guide: Performance*
- "maxappls - Maximum number of active applications configuration parameter" in the *Administration Guide: Performance*
- "GET DATABASE CONFIGURATION" on page 275
- "RESET DATABASE CONFIGURATION Command" in the *Command Reference*
- "UPDATE DATABASE CONFIGURATION" on page 381

autorestart - Auto restart enable

Configuration Type	Database
Parameter Type	Configurable Online
Propagation Class	Immediate
Default [Range]	On [On; Off]

When this parameter is set on, the database manager automatically calls the restart database utility, if needed, when an application connects to a database. *Crash recovery* is the operation performed by the restart database utility. It is performed if the database terminated abnormally while applications were connected to it. An abnormal termination of the database could be caused by a power failure or a

system software failure. It applies any committed transactions that were in the database buffer pool but were not written to disk at the time of the failure. It also backs out any uncommitted transactions that might have been written to disk.

If *autorestart* is not enabled, then an application that attempts to connect to a database which needs to have crash recovery performed (needs to be restarted) will receive a SQL1015N error. In this case, the application can call the restart database utility, or you can restart the database by selecting the restart operation of the recovery tool.

Related concepts:

- "Crash recovery" on page 803

Related reference:

- "GET DATABASE CONFIGURATION" on page 275
- "RESET DATABASE CONFIGURATION Command" in the *Command Reference*
- "RESTART DATABASE" on page 352
- "UPDATE DATABASE CONFIGURATION" on page 381

database_consistent - Database is consistent

Configuration Type	Database
Parameter Type	Informational

This parameter indicates whether the database is in a consistent state.

YES indicates that all transactions have been committed or rolled back so that the data is consistent. If the system "crashes" while the database is consistent, you do not need to take any special action to make the database usable.

NO indicates that a transaction is pending or some other task is pending on the database and the data is not consistent at this point. If the system "crashes" while the database is not consistent, you will need to restart the database using the RESTART DATABASE command to make the database usable.

Related reference:

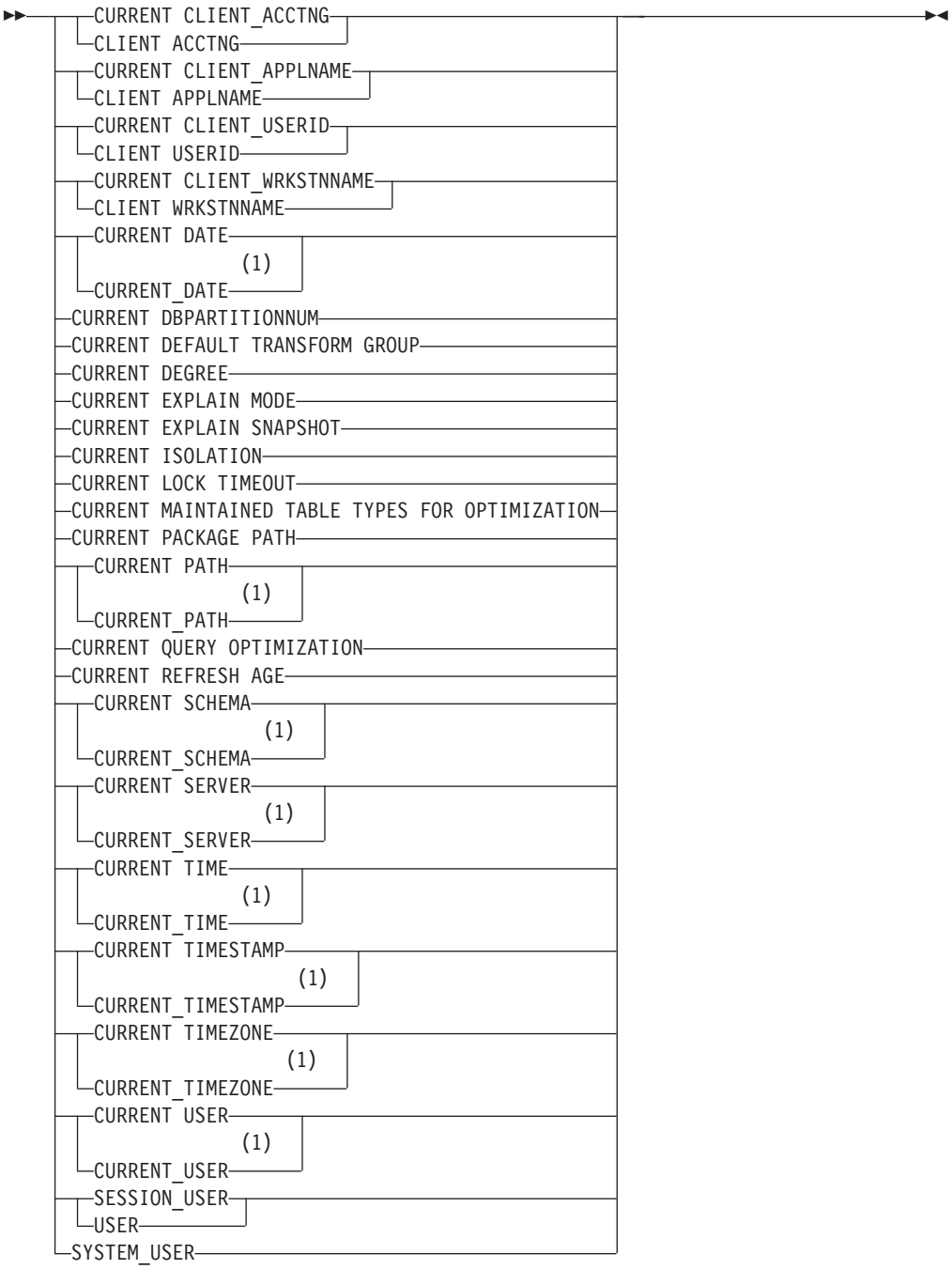
- "GET DATABASE CONFIGURATION" on page 275

Chapter 17. Security-Related Special Registers

Special registers

A *special register* is a storage area that is defined for an application process by the database manager. It is used to store information that can be referenced in SQL statements. A reference to a special register is a reference to a value provided by the current server. If the value is a string, its CCSID is a default CCSID of the current server. The special registers can be referenced as follows:

© Copyright IBM Corp. 1993-2004



Notes:

1 The SQL 1999 Core standard uses the form with the underscore.

Some special registers can be updated using the SET statement. The following table shows which of the special registers can be updated.

Table 69. Special Registers

Special Register	Updatable
CURRENT CLIENT_ACCTNG	No
CURRENT CLIENT_APPLNAME	No

Table 69. Special Registers (continued)

Special Register	Updatable
CURRENT CLIENT_USERID	No
CURRENT CLIENT_WRKSTNNAME	No
CURRENT DATE	No
CURRENT DBPARTITIONNUM	No
CURRENT DEFAULT TRANSFORM GROUP	Yes
CURRENT DEGREE	Yes
CURRENT EXPLAIN MODE	Yes
CURRENT EXPLAIN SNAPSHOT	Yes
CURRENT ISOLATION	Yes
CURRENT LOCK TIMEOUT	Yes
CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION	Yes
CURRENT PACKAGE PATH	Yes
CURRENT PATH	Yes
CURRENT QUERY OPTIMIZATION	Yes
CURRENT REFRESH AGE	Yes
CURRENT SCHEMA	Yes
CURRENT SERVER	No
CURRENT TIME	No
CURRENT TIMESTAMP	No
CURRENT TIMEZONE	No
CURRENT USER	No
SESSION_USER	Yes
SYSTEM_USER	No
USER	Yes

When a special register is referenced in a routine, the value of the special register in the routine depends on whether the special register is updatable or not. For non-updatable special registers, the value is set to the default value for the special register. For updatable special registers, the initial value is inherited from the invoker of the routine and can be changed with a subsequent SET statement inside the routine.

CURRENT CLIENT_APPLNAME

The CURRENT CLIENT_APPLNAME (or CLIENT APPLNAME) special register contains the value of the application name from the client information specified for this connection. The data type of the register is VARCHAR(255). The default value of this register is an empty string.

The value of the application name can be changed by using the Set Client Information (`sqleseti`) API.

Note that the value provided via the sqleseti API is in the application code page, and the special register value is stored in the database code page. Depending on

the data values used when setting the client information, truncation of the data value stored in the special register may occur during code page conversion.

Example: Select which departments are allowed to use the application being used in this connection.

```
SELECT DEPT
  FROM DEPT_APPL_MAP
  WHERE APPL_NAME = CURRENT CLIENT_APPLNAME
```

CURRENT CLIENT_USERID

The CURRENT CLIENT_USERID (or CLIENT USERID) special register contains the value of the client user ID from the client information specified for this connection. The data type of the register is VARCHAR(255). The default value of this register is an empty string.

The value of the client user ID can be changed by using the Set Client Information (`sqleseti`) API.

Note that the value provided via the sqleseti API is in the application code page, and the special register value is stored in the database code page. Depending on the data values used when setting the client information, truncation of the data value stored in the special register may occur during code page conversion.

Example: Find out in which department the current client user ID works.

```
SELECT DEPT
  FROM DEPT_USERID_MAP
  WHERE USER_ID = CURRENT CLIENT_USERID
```

CURRENT CLIENT_WRKSTNNAME

The CURRENT CLIENT_WRKSTNNAME (or CLIENT WRKSTNNAME) special register contains the value of the workstation name from the client information specified for this connection. The data type of the register is VARCHAR(255). The default value of this register is an empty string.

The value of the workstation name can be changed by using the Set Client Information (`sqleseti`) API.

Note that the value provided via the sqleseti API is in the application code page, and the special register value is stored in the database code page. Depending on the data values used when setting the client information, truncation of the data value stored in the special register may occur during code page conversion.

Example: Get the workstation name being used for this connection.

```
VALUES (CURRENT CLIENT_WRKSTNNAME)
  INTO :WS_NAME
```

CURRENT SERVER

The CURRENT SERVER (or CURRENT_SERVER) special register specifies a VARCHAR(18) value that identifies the current application server. The register contains the actual name of the application server, not an alias.

CURRENT SERVER can be changed through the CONNECT statement, but only under certain conditions.

When used in an SQL statement inside a routine, CURRENT SERVER is not inherited from the invoking statement.

Example: Set the host variable APPL_SERVE (VARCHAR(18)) to the name of the application server to which the application is connected.

```
VALUES CURRENT SERVER INTO :APPL_SERVE
```

Related reference:

- "CONNECT (Type 1)" on page 887

CURRENT SCHEMA

The CURRENT SCHEMA (or CURRENT_SCHEMA) special register specifies a VARCHAR(128) value that identifies the schema name used to qualify database object references, where applicable, in dynamically prepared SQL statements. For compatibility with DB2 for OS/390, CURRENT SQLID (or CURRENT_SQLID) is a synonym for CURRENT SCHEMA.

The initial value of CURRENT SCHEMA is the authorization ID of the current session user. The value can be changed by invoking the SET SCHEMA statement.

The QUALIFIER bind option controls the schema name used to qualify database object references, where applicable, for static SQL statements.

Example: Set the schema for object qualification to 'D123'.

```
SET CURRENT SCHEMA =  'D123'
```

USER

The USER special register specifies the run-time authorization ID passed to the database manager when an application starts on a database. The data type of the register is VARCHAR(128).

When used in an SQL statement inside a routine, USER is not inherited from the invoking statement.

Example: Select all notes from the IN_TRAY table that were placed there by the user.

```
SELECT * FROM IN_TRAY
  WHERE SOURCE = USER
```

Chapter 18. Crash Recovery and Database Logs

Crash recovery

Transactions (or units of work) against a database can be interrupted unexpectedly. If a failure occurs before all of the changes that are part of the unit of work are completed and committed, the database is left in an inconsistent and unusable state. *Crash recovery* is the process by which the database is moved back to a consistent and usable state. This is done by rolling back incomplete transactions and completing committed transactions that were still in memory when the crash occurred (Figure 14). When a database is in a consistent and usable state, it has attained what is known as a "point of consistency".

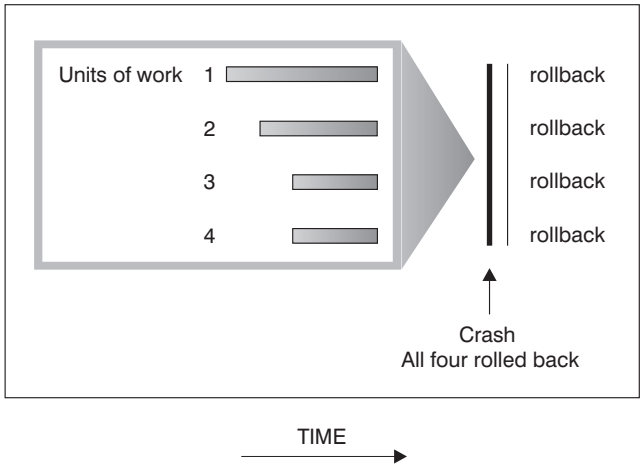


Figure 14. Rolling Back Units of Work (Crash Recovery)

A *transaction failure* results from a severe error or condition that causes the database or the database manager to end abnormally. Partially completed units of work, or UOW that have not been flushed to disk at the time of failure, leave the database in an inconsistent state. Following a transaction failure, the database must be recovered. Conditions that can result in transaction failure include:

- A power failure on the machine, causing the database manager and the database partitions on it to go down
- A hardware failure such as memory corruption, or disk, CPU, or network failure.
- A serious operating system error that causes DB2® to go down

If you want the rollback of incomplete units of work to be done automatically by the database manager, enable the automatic restart (*autorestart*) database configuration parameter by setting it to `ON`. (This is the default value.) If you do not want automatic restart behavior, set the *autorestart* database configuration parameter to `OFF`. As a result, you will need to issue the RESTART DATABASE command when a database failure occurs. If the database I/O was suspended before the crash occurred, you must specify the WRITE RESUME option of the

© Copyright IBM Corp. 1993-2004

RESTART DATABASE command in order for the crash recovery to continue. The administration notification log records when the database restart operation begins.

If crash recovery is applied to a database that is enabled for forward recovery (that is, the *logarchmeth1* configuration parameter is not set to OFF), and an error occurs during crash recovery that is attributable to an individual table space, that table space will be taken offline, and cannot be accessed until it is repaired. Crash recovery continues. At the completion of crash recovery, the other table spaces in the database will be accessible, and connections to the database can be established. However, if the table space that is taken offline is the table space that contains the system catalogs, it must be repaired before any connections will be permitted.

Related reference:

- "autorestart - Auto restart enable" on page 795
- "logarchmeth1 - Primary log archive method configuration parameter" in the *Administration Guide: Performance*

Understanding recovery logs

All databases have logs associated with them. These logs keep records of database changes. If a database needs to be restored to a point beyond the last full, offline backup, logs are required to roll the data forward to the point of failure.

There are two types of DB2® logging: *circular*, and *archive*, each provides a different level of recovery capability:

- *Circular* logging is the default behavior when a new database is created. (The *logarchmeth1* and *logarchmeth2* database configuration parameters are set to OFF.) With this type of logging, only full, offline backups of the database are allowed. The database must be offline (inaccessible to users) when a full backup is taken. As the name suggests, circular logging uses a "ring" of online logs to provide recovery from transaction failures and system crashes. The logs are used and retained only to the point of ensuring the integrity of current transactions. Circular logging does not allow you to roll a database forward through transactions performed after the last full backup operation. All changes occurring since the last backup operation are lost. Since this type of restore operation recovers your data to the specific point in time at which a full backup was taken, it is called *version recovery*.

  Figure 15 on page 805 shows that the active log uses a ring of log files when circular logging is active.

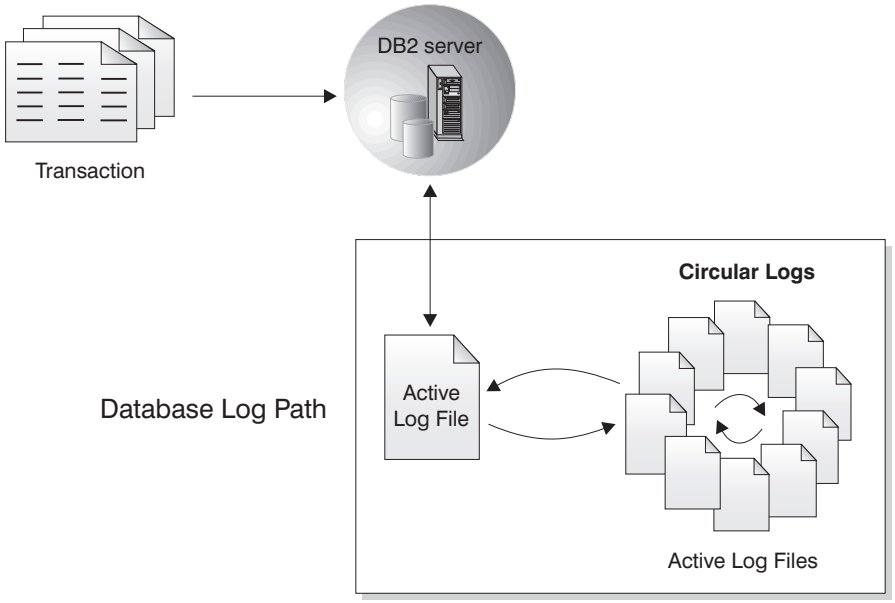


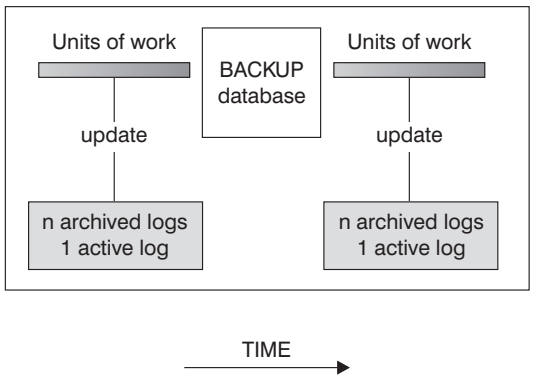
Figure 15. Circular Logging

Active logs are used during crash recovery to prevent a failure (system power or application error) from leaving a database in an inconsistent state. Active logs are located in the database log path directory.

- *Archive* logging is used specifically for rollforward recovery. Archived logs are logs that were active but are no longer required for crash recovery. Use the *logarchmeth1* database configuration parameter to enable archive logging.

The advantage of choosing archive logging is that rollforward recovery can use both archived logs and active logs to rebuild a database either to the end of the logs, or to a specific point in time. The archived log files can be used to recover changes made after the backup was taken. This is different from circular logging where you can only recover to the time of the backup, and all changes made after that are lost.

Taking online backups is only supported if the database is configured for archive logging. During an online backup operation, all activities against the database are logged. When an online backup image is restored, the logs must be rolled forward at least to the point in time at which the backup operation completed. For this to happen, the logs must have been archived and made available when the database is restored. After an online backup is complete, DB2 forces the currently active log to be closed, and as a result, it will be archived. This ensures that your online backup has a complete set of archived logs available for recovery.



Logs are used between backups to track the changes to the databases.

Figure 16. Active and Archived Database Logs in Rollforward Recovery. There can be more than one active log in the case of a long-running transaction.

The following database configuration parameters allow you to change where archived logs are stored: The *newlogpath* parameter, and the *logarchmeth1* and *logarchmeth2* parameters. Changing the *newlogpath* parameter also affects where active logs are stored.

To determine which log *extents* in the database log path directory are archived logs, check the value of the *loghead* database configuration parameter. This parameter indicates the lowest numbered log that is active. Those logs with sequence numbers less than *loghead* are archived logs and can be moved. You can check the value of this parameter by using the Control Center; or, by using the command line processor and the GET DATABASE CONFIGURATION command to view the "First active log file". For more information about this configuration parameter, see the *Administration Guide: Performance* book.

Related concepts:

- "Log mirroring" in the *Data Recovery and High Availability Guide and Reference*

Related reference:

- "User exit for database recovery" in the *Data Recovery and High Availability Guide and Reference*
- "loghead - First active log file configuration parameter" in the *Administration Guide: Performance*
- "Configuration parameters for database logging" in the *Data Recovery and High Availability Guide and Reference*
- "logarchmeth1 - Primary log archive method configuration parameter" in the *Administration Guide: Performance*

Chapter 19. Application processes, concurrency, and recovery

All SQL programs execute as part of an *application process* or agent. An application process involves the execution of one or more programs, and is the unit to which the database manager allocates resources and locks. Different application processes may involve the execution of different programs, or different executions of the same program.

More than one application process may request access to the same data at the same time. *Locking* is the mechanism used to maintain data integrity under such conditions, preventing, for example, two application processes from updating the same row of data simultaneously.

The database manager acquires locks to prevent uncommitted changes made by one application process from being accidentally perceived by any other process. The database manager releases all locks it has acquired and retained on behalf of an application process when that process ends. However, an application process can explicitly request that locks be released sooner. This is done using a *commit* operation, which releases locks acquired during the unit of work and also commits database changes made during the unit of work.

The database manager provides a means of backing out uncommitted changes made by an application process. This might be necessary in the event of a failure on the part of an application process, or in the case of a deadlock, or a lock time-out situation. An application process can explicitly request that its database changes be backed out. This is done using a *rollback* operation.

A *unit of work* is a recoverable sequence of operations within an application process. A unit of work is initiated when an application process is started, or when the previous unit of work is ended by something other than the termination of the application process. A unit of work is ended by a commit operation, a rollback operation, or the end of an application process. A commit or rollback operation affects only the database changes made within the unit of work it is ending.

As long as these changes remain uncommitted, other application processes are unable to perceive them, and they can be backed out. This is not true, however, when the isolation level is uncommitted read (UR). Once committed, these database changes are accessible by other application processes and can no longer be backed out through a rollback.

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

Locks acquired by the database manager on behalf of an application process are held until the end of a unit of work. This is not true, however, when the isolation level is cursor stability (CS, in which the lock is released as the cursor moves from row to row) or uncommitted read (UR, in which locks are not obtained).

An application process is never prevented from performing operations because of its own locks. However, if an application uses concurrent transactions, the locks from one transaction may affect the operation of a concurrent transaction.

© Copyright IBM Corp. 1993-2004

The initiation and the termination of a unit of work define points of consistency within an application process. For example, a banking transaction may involve the transfer of funds from one account to another. Such a transaction would require that these funds be subtracted from the first account, and then added to the second account. Following the subtraction step, the data is inconsistent. Only after the funds have been added to the second account is consistency reestablished. When both steps are complete, the commit operation can be used to end the unit of work, thereby making the changes available to other application processes. If a failure occurs before the unit of work ends, the database manager will roll back uncommitted changes to restore the data consistency that it assumes existed when the unit of work was initiated.

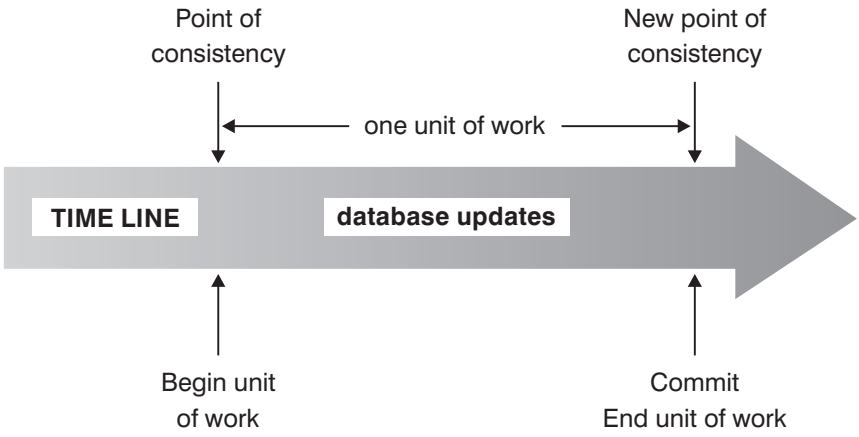


Figure 17. Unit of Work with a COMMIT Statement

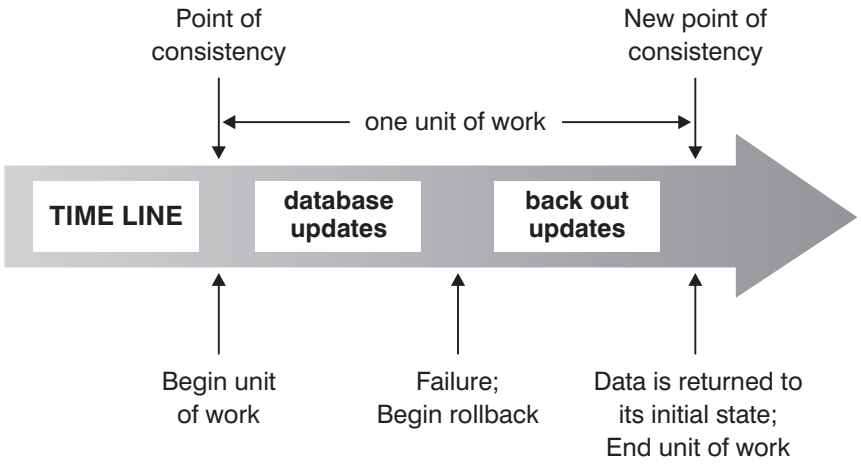


Figure 18. Unit of Work with a ROLLBACK Statement

Related concepts:

- "Isolation levels" in the *SQL Reference, Volume 1*

Chapter 20. Identifiers

An *identifier* is a token that is used to form a name. An identifier in an SQL statement is either an SQL identifier or a host identifier.

- SQL identifiers

  There are two types of *SQL identifiers*: ordinary and delimited.

  – An *ordinary identifier* is a letter followed by zero or more characters, each of which is an uppercase letter, a digit, or the underscore character. An ordinary identifier should not be identical to a reserved word.

    Examples

    ```
    WKLYSAL     WKLY_SAL
    ```

  – A *delimited identifier* is a sequence of one or more characters enclosed by double quotation marks. Two consecutive quotation marks are used to represent one quotation mark within the delimited identifier. In this way an identifier can include lowercase letters.

    Examples

    ```
    "WKLY_SAL"     "WKLY SAL"     "UNION"     "wkly_sal"
    ```

  Character conversion of identifiers created on a double-byte code page, but used by an application or database on a multi-byte code page, may require special consideration: After conversion, such identifiers may exceed the length limit for an identifier.

- Host identifiers

  A *host identifier* is a name declared in the host program. The rules for forming a host identifier are the rules of the host language. A host identifier should not be greater than 255 characters in length and should not begin with SQL or DB2 (in uppercase or lowercase characters).

Naming conventions and implicit object name qualifications

The rules for forming the name of an object depend on the object type. Database object names may be made up of a single identifier, or they may be schema-qualified objects made up of two identifiers. Schema-qualified object names may be specified without the schema name; in such cases, the schema name is implicit.

In dynamic SQL statements, a schema-qualified object name implicitly uses the CURRENT SCHEMA special register value as the qualifier for unqualified object name references. By default it is set to the current authorization ID. If the dynamic SQL statement is contained in a package that exhibits bind, define, or invoke behaviour, the CURRENT SCHEMA special register is not used for qualification. In a bind behaviour package, the package default qualifier is used as the value for implicit qualification of unqualified object references. In a define behaviour package, the authorization ID of the routine definer is used as the value for implicit qualification of unqualified object references within that routine. In an invoke behaviour package, the statement authorization ID in effect when the routine is invoked is used as the value for implicit qualification of unqualified object references within dynamic SQL statements within that routine. For more information, see “Dynamic SQL characteristics at run time” on page 815.

© Copyright IBM Corp. 1993-2004

Naming conventions and implicit object name qualifications

In static SQL statements, the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified database object names. By default, this value is set to the package authorization ID.

The following object names, when used in the context of an SQL procedure, are permitted to use only the characters allowed in an ordinary identifier, even if the names are delimited:

- condition-name
- label
- parameter-name
- procedure-name
- SQL-variable-name
- statement-name

The syntax diagrams use different terms for different types of names. The following list defines these terms.

alias-name A schema-qualified name that designates an alias.

attribute-name An identifier that designates an attribute of a structured data type.

authorization-name An identifier that designates a user or a group:

- Valid characters are A through Z, a through z, 0 through 9, #, @, $, and _.
- The name must not begin with the characters 'SYS', 'IBM', or 'SQL'.
- The name must not be: ADMINS, GUESTS, LOCAL, PUBLIC, or USERS.
- A delimited authorization ID must not contain lowercase letters.

bufferpool-name An identifier that designates a bufferpool.

column-name A qualified or unqualified name that designates a column of a table or view. The qualifier is a table name, a view name, a nickname, or a correlation name.

condition-name An identifier that designates a condition in an SQL procedure.

constraint-name An identifier that designates a referential constraint, primary key constraint, unique constraint, or a table check constraint.

correlation-name An identifier that designates a result table.

cursor-name An identifier that designates an SQL cursor. For host compatibility, a hyphen character may be used in the name.

data-source-name An identifier that designates a data source. This identifier is the first part of a three-part remote object name.

descriptor-name A colon followed by a host identifier that designates an SQL descriptor area (SQLDA). For the description of a host identifier, see “References

	to host variables" on page 823. Note that a descriptor name never includes an indicator variable.
distinct-type-name	A qualified or unqualified name that designates a distinct type. An unqualified distinct type name in an SQL statement is implicitly qualified by the database manager, depending on context.
event-monitor-name	An identifier that designates an event monitor.
function-mapping-name	An identifier that designates a function mapping.
function-name	A qualified or unqualified name that designates a function. An unqualified function name in an SQL statement is implicitly qualified by the database manager, depending on context.
group-name	An unqualified identifier that designates a transform group defined for a structured type.
host-variable	A sequence of tokens that designates a host variable. A host variable includes at least one host identifier, explained in "References to host variables" on page 823.
index-name	A schema-qualified name that designates an index or an index specification.
label	An identifier that designates a label in an SQL procedure.
method-name	An identifier that designates a method. The schema context for a method is determined by the schema of the subject type (or a supertype of the subject type) of the method.
nickname	A schema-qualified name that designates a federated server reference to a table or a view.
db-partition-group-name	An identifier that designates a database partition group.
package-name	A schema-qualified name that designates a package. If a package has a version ID that is not the empty string, the package name also includes the version ID at the end of the name, in the form: `schema-id.package-id.version-id`.
parameter-name	An identifier that designates a parameter that can be referenced in a procedure, user-defined function, method, or index extension.
procedure-name	A qualified or unqualified name that designates a procedure. An unqualified procedure name in an SQL statement is implicitly qualified by the database manager, depending on context.
remote-authorization-name	An identifier that designates a data source user. The rules for authorization names vary from data source to data source.
remote-function-name	A name that designates a function registered to a data source database.

remote-object-name
A three-part name that designates a data source table or view, and that identifies the data source in which the table or view resides. The parts of this name are data-source-name, remote-schema-name, and remote-table-name.

remote-schema-name
A name that designates the schema to which a data source table or view belongs. This name is the second part of a three-part remote object name.

remote-table-name
A name that designates a table or view at a data source. This name is the third part of a three-part remote object name.

remote-type-name
A data type supported by a data source database. Do not use the long form for built-in types (use CHAR instead of CHARACTER, for example).

savepoint-name
An identifier that designates a savepoint.

schema-name
An identifier that provides a logical grouping for SQL objects. A schema name used as a qualifier for the name of an object may be implicitly determined:

- from the value of the CURRENT SCHEMA special register
- from the value of the QUALIFIER precompile/bind option
- on the basis of a resolution algorithm that uses the CURRENT PATH special register
- on the basis of the schema name for another object in the same SQL statement.

To avoid complications, it is recommended that the name SESSION not be used as a schema, except as the schema for declared global temporary tables (which *must* use the schema name SESSION).

server-name
An identifier that designates an application server. In a federated system, the server name also designates the local name of a data source.

specific-name
A qualified or unqualified name that designates a specific name. An unqualified specific name in an SQL statement is implicitly qualified by the database manager, depending on context.

SQL-variable-name
The name of a local variable in an SQL procedure statement. SQL variable names can be used in other SQL statements where a host variable name is allowed. The name can be qualified by the label of the compound statement that declared the SQL variable.

statement-name
An identifier that designates a prepared SQL statement.

supertype-name
A qualified or unqualified name that designates the supertype of a type. An unqualified supertype

	name in an SQL statement is implicitly qualified by the database manager, depending on context.
table-name	A schema-qualified name that designates a table.
tablespace-name	An identifier that designates a table space.
trigger-name	A schema-qualified name that designates a trigger.
type-mapping-name	An identifier that designates a data type mapping.
type-name	A qualified or unqualified name that designates a type. An unqualified type name in an SQL statement is implicitly qualified by the database manager, depending on context.
typed-table-name	A schema-qualified name that designates a typed table.
typed-view-name	A schema-qualified name that designates a typed view.
view-name	A schema-qualified name that designates a view.
wrapper-name	An identifier that designates a wrapper.

Aliases

A table alias can be thought of as an alternative name for a table or a view. A table or view, therefore, can be referred to in an SQL statement by its name or by a table alias.

An alias can be used wherever a table or a view name can be used. An alias can be created even if the object does not exist (although it must exist by the time a statement referring to it is compiled). It can refer to another alias if no circular or repetitive references are made along the chain of aliases. An alias can only refer to a table, view, or alias within the same database. An alias name cannot be used where a new table or view name is expected, such as in the CREATE TABLE or CREATE VIEW statements; for example, if the alias name PERSONNEL has been created, subsequent statements such as CREATE TABLE PERSONNEL... will return an error.

The option of referring to a table or a view by an alias is not explicitly shown in the syntax diagrams, or mentioned in the descriptions of SQL statements.

A new unqualified alias cannot have the same fully-qualified name as an existing table, view, or alias.

The effect of using an alias in an SQL statement is similar to that of text substitution. The alias, which must be defined by the time that the SQL statement is compiled, is replaced at statement compilation time by the qualified base table or view name. For example, if PBIRD.SALES is an alias for DSPN014.DIST4_SALES_148, then at compilation time:

```
SELECT * FROM PBIRD.SALES
```

effectively becomes

```
SELECT * FROM DSPN014.DIST4_SALES_148
```

In a federated system, the aforementioned uses and restrictions apply, not only to table aliases, but also to aliases for nicknames. Thus, a nickname's alias can be

used instead of the nickname in an SQL statement; an alias can be created for a nickname that does not yet exist, provided that the nickname is created before statements that reference the alias are compiled; an alias for a nickname can refer to another alias for that nickname; and so on.

For syntax toleration of applications running under other relational database management systems, SYNONYM can be used in place of ALIAS in the CREATE ALIAS and DROP ALIAS statements.

Authorization IDs and authorization names

An *authorization ID* is a character string that is obtained by the database manager when a connection is established between the database manager and either an application process or a program preparation process. It designates a set of privileges. It may also designate a user or a group of users, but this property is not controlled by the database manager.

Authorization IDs are used by the database manager to provide:

- Authorization checking of SQL statements
- A default value for the QUALIFIER precompile/bind option and the CURRENT SCHEMA special register. The authorization ID is also included in the default CURRENT PATH special register and the FUNCPATH precompile/bind option.

An authorization ID applies to every SQL statement. The authorization ID that applies to a static SQL statement is the authorization ID that is used during program binding. The authorization ID that applies to a dynamic SQL statement is based on the DYNAMICRULES option supplied at bind time, and on the current runtime environment for the package issuing the dynamic SQL statement:

- In a package that has bind behavior, the authorization ID used is the authorization ID of the package owner.
- In a package that has define behavior, the authorization ID used is the authorization ID of the corresponding routine's definer.
- In a package that has run behavior, the authorization ID used is the current authorization ID of the user executing the package.
- In a package that has invoke behavior, the authorization ID used is the authorization ID currently in effect when the routine is invoked. This is called the runtime authorization ID.

For more information, see "Dynamic SQL characteristics at run time" on page 815.

An *authorization name* specified in an SQL statement should not be confused with the authorization ID of the statement. An authorization name is an identifier that is used within various SQL statements. An authorization name is used in the CREATE SCHEMA statement to designate the owner of the schema. An authorization name is used in the GRANT and REVOKE statements to designate a target of the grant or revoke operation. Granting privileges to *X* means that *X* (or a member of the group *X*) will subsequently be the authorization ID of statements that require those privileges.

Examples:

- Assume that SMITH is the user ID and the authorization ID that the database manager obtained when a connection was established with the application process. The following statement is executed interactively:

```
GRANT SELECT ON TDEPT TO KEENE
```

SMITH is the authorization ID of the statement. Therefore, in a dynamic SQL statement, the default value of the CURRENT SCHEMA special register is SMITH, and in static SQL, the default value of the QUALIFIER precompile/bind option is SMITH. The authority to execute the statement is checked against SMITH, and SMITH is the *table-name* implicit qualifier based on qualification rules described in "Naming conventions and implicit object name qualifications" on page 809.

KEENE is an authorization name specified in the statement. KEENE is given the SELECT privilege on SMITH.TDEPT.

- Assume that SMITH has administrative authority and is the authorization ID of the following dynamic SQL statements, with no SET SCHEMA statement issued during the session:

```
DROP TABLE TDEPT
```

Removes the SMITH.TDEPT table.

```
DROP TABLE SMITH.TDEPT
```

Removes the SMITH.TDEPT table.

```
DROP TABLE KEENE.TDEPT
```

Removes the KEENE.TDEPT table. Note that KEENE.TDEPT and SMITH.TDEPT are different tables.

```
CREATE SCHEMA PAYROLL AUTHORIZATION KEENE
```

KEENE is the authorization name specified in the statement that creates a schema called PAYROLL. KEENE is the owner of the schema PAYROLL and is given CREATEIN, ALTERIN, and DROPIN privileges, with the ability to grant them to others.

Dynamic SQL characteristics at run time

The BIND option DYNAMICRULES determines the authorization ID that is used for checking authorization when dynamic SQL statements are processed. In addition, the option also controls other dynamic SQL attributes, such as the implicit qualifier that is used for unqualified object references, and whether certain SQL statements can be invoked dynamically.

The set of values for the authorization ID and other dynamic SQL attributes is called the dynamic SQL statement behavior. The four possible behaviors are run, bind, define, and invoke. As the following table shows, the combination of the value of the DYNAMICRULES BIND option and the runtime environment determines which of the behaviors is used. DYNAMICRULES RUN, which implies run behavior, is the default.

Table 70. How DYNAMICRULES and the runtime environment determine dynamic SQL statement behavior

	Behavior of dynamic SQL statements	
DYNAMICRULES value	**Standalone program environment**	**Routine environment**
BIND	Bind behavior	Bind behavior
RUN	Run behavior	Run behavior
DEFINEBIND	Bind behavior	Define behavior
DEFINERUN	Run behavior	Define behavior
INVOKEBIND	Bind behavior	Invoke behavior

Table 70. How DYNAMICRULES and the runtime environment determine dynamic SQL statement behavior (continued)

DYNAMICRULES value	Behavior of dynamic SQL statements	
	Standalone program environment	**Routine environment**
INVOKERUN	Run behavior	Invoke behavior

Run behavior

DB2 uses the authorization ID of the user (the ID that initially connected to DB2) executing the package as the value to be used for authorization checking of dynamic SQL statements and for the initial value used for implicit qualification of unqualified object references within dynamic SQL statements.

Bind behavior

At run time, DB2 uses all the rules that apply to static SQL for authorization and qualification. It takes the authorization ID of the package owner as the value to be used for authorization checking of dynamic SQL statements, and the package default qualifier for implicit qualification of unqualified object references within dynamic SQL statements.

Define behavior

Define behavior applies only if the dynamic SQL statement is in a package that is run within a routine context, and the package was bound with DYNAMICRULES DEFINEBIND or DYNAMICRULES DEFINERUN. DB2 uses the authorization ID of the routine definer (not the routine's package binder) as the value to be used for authorization checking of dynamic SQL statements, and for implicit qualification of unqualified object references within dynamic SQL statements within that routine.

Invoke behavior

Invoke behavior applies only if the dynamic SQL statement is in a package that is run within a routine context, and the package was bound with DYNAMICRULES INVOKEBIND or DYNAMICRULES INVOKERUN. DB2 uses the statement authorization ID in effect when the routine is invoked as the value to be used for authorization checking of dynamic SQL, and for implicit qualification of unqualified object references within dynamic SQL statements within that routine. This is summarized by the following table.

Invoking Environment	ID Used
any static SQL	implicit or explicit value of the OWNER of the package the SQL invoking the routine came from
used in definition of view or trigger	definer of the view or trigger
dynamic SQL from a bind behavior package	implicit or explicit value of the OWNER of the package the SQL invoking the routine came from
dynamic SQL from a run behavior package	ID used to make the initial connection to DB2

Invoking Environment	ID Used
dynamic SQL from a define behavior package	definer of the routine that uses the package that the SQL invoking the routine came from
dynamic SQL from an invoke behavior package	the current authorization ID invoking the routine

Restricted statements when run behavior does not apply

When bind, define, or invoke behavior is in effect, you cannot use the following dynamic SQL statements: GRANT, REVOKE, ALTER, CREATE, DROP, COMMENT, RENAME, SET INTEGRITY, SET EVENT MONITOR STATE; or queries that reference a nickname.

Considerations regarding the DYNAMICRULES option

The CURRENT SCHEMA special register cannot be used to qualify unqualified object references within dynamic SQL statements executed from bind, define or invoke behavior packages. This is true even after you issue the SET CURRENT SCHEMA statement to change the CURRENT SCHEMA special register; the register value is changed but not used.

In the event that multiple packages are referenced during a single connection, all dynamic SQL statements prepared by those packages will exhibit the behavior specified by the DYNAMICRULES option for that specific package and the environment in which they are used.

It is important to keep in mind that when a package exhibits bind behavior, the binder of the package should not have any authorities granted that the user of the package should not receive, because a dynamic statement will be using the authorization ID of the package owner. Similarly, when a package exhibits define behavior, the definer of the routine should not have any authorities granted that the user of the package should not receive.

Authorization IDs and statement preparation

If the VALIDATE BIND option is specified at bind time, the privileges required to manipulate tables and views must also exist at bind time. If these privileges or the referenced objects do not exist, and the SQLERROR NOPACKAGE option is in effect, the bind operation will be unsuccessful. If the SQLERROR CONTINUE option is specified, the bind operation will be successful, and any statements in error will be flagged. Any attempt to execute such a statement will result in an error.

If a package is bound with the VALIDATE RUN option, all normal bind processing is completed, but the privileges required to use the tables and views that are referenced in the application need not exist yet. If a required privilege does not exist at bind time, an incremental bind operation is performed whenever the statement is first executed in an application, and all privileges required for the statement must exist. If a required privilege does not exist, execution of the statement is unsuccessful.

Authorization checking at run time is performed using the authorization ID of the package owner.

Column names

The meaning of a *column name* depends on its context. A column name can be used to:

- Declare the name of a column, as in a CREATE TABLE statement.
- Identify a column, as in a CREATE INDEX statement.
- Specify values of the column, as in the following contexts:
  - In a column function, a column name specifies all values of the column in the group or intermediate result table to which the function is applied. For example, MAX(SALARY) applies the function MAX to all values of the column SALARY in a group.
  - In a GROUP BY or ORDER BY clause, a column name specifies all values in the intermediate result table to which the clause is applied. For example, ORDER BY DEPT orders an intermediate result table by the values of the column DEPT.
  - In an expression, a search condition, or a scalar function, a column name specifies a value for each row or group to which the construct is applied. For example, when the search condition CODE = 20 is applied to some row, the value specified by the column name CODE is the value of the column CODE in that row.
- Temporarily rename a column, as in the *correlation-clause* of a *table-reference* in a FROM clause.

Qualified column names

A qualifier for a column name may be a table, view, nickname, alias, or correlation name.

Whether a column name may be qualified depends on its context:

- Depending on the form of the COMMENT ON statement, a single column name may need to be qualified. Multiple column names must be unqualified.
- Where the column name specifies values of the column, it may be qualified at the user's option.
- In the assignment-clause of an UPDATE statement, it may be qualified at the user's option.
- In all other contexts, a column name must not be qualified.

Where a qualifier is optional, it can serve two purposes. They are described under "Column name qualifiers to avoid ambiguity" on page 820 and "Column name qualifiers in correlated references" on page 822.

Correlation names

A *correlation name* can be defined in the FROM clause of a query and in the first clause of an UPDATE or DELETE statement. For example, the clause FROM X.MYTABLE Z establishes Z as a correlation name for X.MYTABLE.

```
FROM X.MYTABLE Z
```

With Z defined as a correlation name for X.MYTABLE, only Z can be used to qualify a reference to a column of that instance of X.MYTABLE in that SELECT statement.

A correlation name is associated with a table, view, nickname, alias, nested table expression or table function only within the context in which it is defined. Hence,

the same correlation name can be defined for different purposes in different statements, or in different clauses of the same statement.

As a qualifier, a correlation name can be used to avoid ambiguity or to establish a correlated reference. It can also be used merely as a shorter name for a table, view, nickname, or alias. In the case of a nested table expression or table function, a correlation name is required to identify the result table. In the example, Z might have been used merely to avoid having to enter X.MYTABLE more than once.

If a correlation name is specified for a table, view, nickname, or alias name, any qualified reference to a column of that instance of the table, view, nickname, or alias must use the correlation name, rather than the table, view, nickname, or alias name. For example, the reference to EMPLOYEE.PROJECT in the following example is incorrect, because a correlation name has been specified for EMPLOYEE:

Example

```
FROM EMPLOYEE E
  WHERE EMPLOYEE.PROJECT='ABC'      * incorrect*
```

The qualified reference to PROJECT should instead use the correlation name, "E", as shown below:

```
FROM EMPLOYEE E
  WHERE E.PROJECT='ABC'
```

Names specified in a FROM clause are either *exposed* or *non-exposed*. A table, view, nickname, or alias name is said to be exposed in the FROM clause if a correlation name is not specified. A correlation name is always an exposed name. For example, in the following FROM clause, a correlation name is specified for EMPLOYEE but not for DEPARTMENT, so DEPARTMENT is an exposed name, and EMPLOYEE is not:

```
FROM EMPLOYEE E, DEPARTMENT
```

A table, view, nickname, or alias name that is exposed in a FROM clause may be the same as any other table name, view name or nickname exposed in that FROM clause or any correlation name in the FROM clause. This may result in ambiguous column name references which returns an error (SQLSTATE 42702).

The first two FROM clauses shown below are correct, because each one contains no more than one reference to EMPLOYEE that is exposed:

1. Given the FROM clause:

   ```
   FROM EMPLOYEE E1, EMPLOYEE
   ```

   a qualified reference such as EMPLOYEE.PROJECT denotes a column of the second instance of EMPLOYEE in the FROM clause. A qualified reference to the first instance of EMPLOYEE must use the correlation name "E1" (E1.PROJECT).
2. Given the FROM clause:

   ```
   FROM EMPLOYEE, EMPLOYEE E2
   ```

   a qualified reference such as EMPLOYEE.PROJECT denotes a column of the first instance of EMPLOYEE in the FROM clause. A qualified reference to the second instance of EMPLOYEE must use the correlation name "E2" (E2.PROJECT).
3. Given the FROM clause:

   ```
   FROM EMPLOYEE, EMPLOYEE
   ```

the two exposed table names included in this clause (EMPLOYEE and EMPLOYEE) are the same. This is allowed, but references to specific column names would be ambiguous (SQLSTATE 42702).

4. Given the following statement:

```
SELECT *
  FROM EMPLOYEE E1, EMPLOYEE E2            * incorrect *
  WHERE EMPLOYEE.PROJECT = 'ABC'
```

   the qualified reference EMPLOYEE.PROJECT is incorrect, because both instances of EMPLOYEE in the FROM clause have correlation names. Instead, references to PROJECT must be qualified with either correlation name (E1.PROJECT or E2.PROJECT).

5. Given the FROM clause:

```
FROM EMPLOYEE, X.EMPLOYEE
```

   a reference to a column in the second instance of EMPLOYEE must use X.EMPLOYEE (X.EMPLOYEE.PROJECT). If X is the CURRENT SCHEMA special register value in dynamic SQL or the QUALIFIER precompile/bind option in static SQL, then the columns cannot be referenced since any such reference would be ambiguous.

The use of a correlation name in the FROM clause also allows the option of specifying a list of column names to be associated with the columns of the result table. As with a correlation name, these listed column names become the *exposed* names of the columns that must be used for references to the columns throughout the query. If a column name list is specified, then the column names of the underlying table become *non-exposed*.

Given the FROM clause:

```
FROM DEPARTMENT D (NUM,NAME,MGR,ANUM,LOC)
```

a qualified reference such as D.NUM denotes the first column of the DEPARTMENT table that is defined in the table as DEPTNO. A reference to D.DEPTNO using this FROM clause is incorrect since the column name DEPTNO is a non-exposed column name.

Column name qualifiers to avoid ambiguity

In the context of a function, a GROUP BY clause, ORDER BY clause, an expression, or a search condition, a column name refers to values of a column in some table, view, nickname, nested table expression or table function. The tables, views, nicknames, nested table expressions and table functions that might contain the column are called the *object tables* of the context. Two or more object tables might contain columns with the same name; one reason for qualifying a column name is to designate the table from which the column comes. Qualifiers for column names are also useful in SQL procedures to distinguish column names from SQL variable names used in SQL statements.

A nested table expression or table function will consider *table-references* that precede it in the FROM clause as object tables. The *table-references* that follow are not considered as object tables.

Table designators

A qualifier that designates a specific object table is called a *table designator*. The clause that identifies the object tables also establishes the table designators for them. For example, the object tables of an expression in a SELECT clause are named in the FROM clause that follows it:

```
SELECT CORZ.COLA, OWNY.MYTABLE.COLA
  FROM OWNX.MYTABLE CORZ, OWNY.MYTABLE
```

Table designators in the FROM clause are established as follows:

- A name that follows a table, view, nickname, alias, nested table expression or table function is both a correlation name and a table designator. Thus, CORZ is a table designator. CORZ is used to qualify the first column name in the select list.
- An exposed table, view name, nickname or alias is a table designator. Thus, OWNY.MYTABLE is a table designator. OWNY.MYTABLE is used to qualify the second column name in the select list.

Each table designator should be unique within a particular FROM clause to avoid the possibility of ambiguous references to columns.

Avoiding undefined or ambiguous references

When a column name refers to values of a column, exactly one object table must include a column with that name. The following situations are considered errors:

- No object table contains a column with the specified name. The reference is undefined.
- The column name is qualified by a table designator, but the table designated does not include a column with the specified name. Again the reference is undefined.
- The name is unqualified, and more than one object table includes a column with that name. The reference is ambiguous.
- The column name is qualified by a table designator, but the table designated is not unique in the FROM clause and both occurrences of the designated table include the column. The reference is ambiguous.
- The column name is in a nested table expression which is not preceded by the TABLE keyword or in a table function or nested table expression that is the right operand of a right outer join or a full outer join and the column name does not refer to a column of a *table-reference* within the nested table expression's fullselect. The reference is undefined.

Avoid ambiguous references by qualifying a column name with a uniquely defined table designator. If the column is contained in several object tables with different names, the table names can be used as designators. Ambiguous references can also be avoided without the use of the table designator by giving unique names to the columns of one of the object tables using the column name list following the correlation name.

When qualifying a column with the exposed table name form of a table designator, either the qualified or unqualified form of the exposed table name may be used. However, the qualifier used and the table used must be the same after fully qualifying the table name, view name or nickname and the table designator.

1. If the authorization ID of the statement is CORPDATA:

   ```
   SELECT CORPDATA.EMPLOYEE.WORKDEPT
     FROM EMPLOYEE
   ```

   is a valid statement.

2. If the authorization ID of the statement is REGION:

   ```
   SELECT CORPDATA.EMPLOYEE.WORKDEPT
     FROM EMPLOYEE                          * incorrect *
   ```

   is invalid, because EMPLOYEE represents the table REGION.EMPLOYEE, but the qualifier for WORKDEPT represents a different table, CORPDATA.EMPLOYEE.

Column name qualifiers in correlated references

A *fullselect* is a form of a query that may be used as a component of various SQL statements. A fullselect used within a search condition of any statement is called a *subquery*. A fullselect used to retrieve a single value as an expression within a statement is called a *scalar fullselect* or *scalar subquery*. A fullselect used in the FROM clause of a query is called a *nested table expression*. Subqueries in search conditions, scalar subqueries and nested table expressions are referred to as subqueries through the remainder of this topic.

A subquery may include subqueries of its own, and these may, in turn, include subqueries. Thus an SQL statement may contain a hierarchy of subqueries. Those elements of the hierarchy that contain subqueries are said to be at a higher level than the subqueries they contain.

Every element of the hierarchy contains one or more table designators. A subquery can reference not only the columns of the tables identified at its own level in the hierarchy, but also the columns of the tables identified previously in the hierarchy, back to the highest level of the hierarchy. A reference to a column of a table identified at a higher level is called a *correlated reference*.

For compatibility with existing standards for SQL, both qualified and unqualified column names are allowed as correlated references. However, it is good practice to qualify all column references used in subqueries; otherwise, identical column names may lead to unintended results. For example, if a table in a hierarchy is altered to contain the same column name as the correlated reference and the statement is prepared again, the reference will apply to the altered table.

When a column name in a subquery is qualified, each level of the hierarchy is searched, starting at the same subquery as the qualified column name appears and continuing to the higher levels of the hierarchy until a table designator that matches the qualifier is found. Once found, it is verified that the table contains the given column. If the table is found at a higher level than the level containing column name, then it is a correlated reference to the level where the table designator was found. A nested table expression must be preceded with the optional TABLE keyword in order to search the hierarchy above the fullselect of the nested table expression.

When the column name in a subquery is not qualified, the tables referenced at each level of the hierarchy are searched, starting at the same subquery where the column name appears and continuing to higher levels of the hierarchy, until a match for the column name is found. If the column is found in a table at a higher level than the level containing column name, then it is a correlated reference to the level where the table containing the column was found. If the column name is found in more than one table at a particular level, the reference is ambiguous and considered an error.

In either case, T, used in the following example, refers to the table designator that contains column C. A column name, T.C (where T represents either an implicit or an explicit qualifier), is a correlated reference if, and only if, these conditions are met:

- T.C is used in an expression of a subquery.
- T does not designate a table used in the from clause of the subquery.
- T designates a table used at a higher level of the hierarchy that contains the subquery.

Since the same table, view or nickname can be identified at many levels, unique correlation names are recommended as table designators. If T is used to designate a table at more than one level (T is the table name itself or is a duplicate correlation name), T.C refers to the level where T is used that most directly contains the subquery that includes T.C. If a correlation to a higher level is needed, a unique correlation name must be used.

The correlated reference T.C identifies a value of C in a row or group of T to which two search conditions are being applied: condition 1 in the subquery, and condition 2 at some higher level. If condition 2 is used in a WHERE clause, the subquery is evaluated for each row to which condition 2 is applied. If condition 2 is used in a HAVING clause, the subquery is evaluated for each group to which condition 2 is applied.

For example, in the following statement, the correlated reference X.WORKDEPT (in the last line) refers to the value of WORKDEPT in table EMPLOYEE at the level of the first FROM clause. (That clause establishes X as a correlation name for EMPLOYEE.) The statement lists employees who make less than the average salary for their department.

```
SELECT EMPNO, LASTNAME, WORKDEPT
  FROM EMPLOYEE X
  WHERE SALARY < (SELECT AVG(SALARY)
                  FROM EMPLOYEE
                  WHERE WORKDEPT = X.WORKDEPT)
```

The next example uses THIS as a correlation name. The statement deletes rows for departments that have no employees.

```
DELETE FROM DEPARTMENT THIS
   WHERE NOT EXISTS(SELECT *
                    FROM EMPLOYEE
                    WHERE WORKDEPT = THIS.DEPTNO)
```

References to host variables

A *host variable* is either:

- A variable in a host language such as a C variable, a C++ variable, a COBOL data item, a FORTRAN variable, or a Java variable

or:

- A host language construct that was generated by an SQL precompiler from a variable declared using SQL extensions

that is referenced in an SQL statement. Host variables are either directly defined by statements in the host language or are indirectly defined using SQL extensions.

A host variable in an SQL statement must identify a host variable described in the program according to the rules for declaring host variables.

All host variables used in an SQL statement must be declared in an SQL DECLARE section in all host languages except REXX. No variables may be declared outside an SQL DECLARE section with names identical to variables declared inside an SQL DECLARE section. An SQL DECLARE section begins with BEGIN DECLARE SECTION and ends with END DECLARE SECTION.

The meta-variable *host-variable*, as used in the syntax diagrams, shows a reference to a host variable. A host-variable in the VALUES INTO clause or the INTO clause of a FETCH or a SELECT INTO statement, identifies a host variable to which a

value from a column of a row or an expression is assigned. In all other contexts a host-variable specifies a value to be passed to the database manager from the application program.

Host variables in dynamic SQL

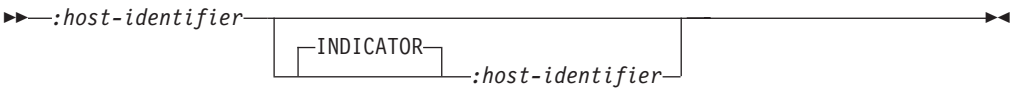
In dynamic SQL statements, parameter markers are used instead of host variables. A parameter marker is a question mark (?) representing a position in a dynamic SQL statement where the application will provide a value; that is, where a host variable would be found if the statement string were a static SQL statement. The following example shows a static SQL statement using host variables:

```
INSERT INTO DEPARTMENT
  VALUES (:hv_deptno, :hv_deptname, :hv_mgrno, :hv_admrdept)
```

This example shows a dynamic SQL statement using parameter markers:

```
INSERT INTO DEPARTMENT VALUES (?, ?, ?, ?)
```

The meta-variable *host-variable* in syntax diagrams can generally be expanded to:



Each *host-identifier* must be declared in the source program. The variable designated by the second host-identifier must have a data type of small integer.

The first host-identifier designates the *main variable*. Depending on the operation, it either provides a value to the database manager or is provided a value from the database manager. An input host variable provides a value in the runtime application code page. An output host variable is provided a value that, if necessary, is converted to the runtime application code page when the data is copied to the output application variable. A given host variable can serve as both an input and an output variable in the same program.

The second host-identifier designates its *indicator variable*. The purposes of the indicator variable are to:

- Specify the null value. A negative value of the indicator variable specifies the null value. A value of -2 indicates a numeric conversion or arithmetic expression error occurred in deriving the result
- Record the original length of a truncated string (if the source of the value is not a large object type)
- Record the seconds portion of a time if the time is truncated on assignment to a host variable.

For example, if :HV1:HV2 is used to specify an insert or update value, and if HV2 is negative, the value specified is the null value. If HV2 is not negative the value specified is the value of HV1.

Similarly, if :HV1:HV2 is specified in a VALUES INTO clause or in a FETCH or SELECT INTO statement, and if the value returned is null, HV1 is not changed, and HV2 is set to a negative value. If the database is configured with DFT_SQLMATHWARN yes (or was during binding of a static SQL statement), HV2 could be -2. If HV2 is -2, a value for HV1 could not be returned because of an error converting to the numeric type of HV1, or an error evaluating an arithmetic expression that is used to determine the value for HV1. When accessing a database

with a client version earlier than DB2 Universal Database Version 5, HV2 will be -1 for arithmetic exceptions. If the value returned is not null, that value is assigned to HV1 and HV2 is set to zero (unless the assignment to HV1 requires string truncation of a non-LOB string; in which case HV2 is set to the original length of the string). If an assignment requires truncation of the seconds part of a time, HV2 is set to the number of seconds.

If the second host identifier is omitted, the host-variable does not have an indicator variable. The value specified by the host-variable reference :HV1 is always the value of HV1, and null values cannot be assigned to the variable. Thus, this form should not be used in an INTO clause unless the corresponding column cannot contain null values. If this form is used and the column contains nulls, the database manager will generate an error at run time.

An SQL statement that references host variables must be within the scope of the declaration of those host variables. For host variables referenced in the SELECT statement of a cursor, that rule applies to the OPEN statement rather than to the DECLARE CURSOR statement.

Example

Using the PROJECT table, set the host variable PNAME (VARCHAR(26)) to the project name (PROJNAME), the host variable STAFF (dec(5,2)) to the mean staffing level (PRSTAFF), and the host variable MAJPROJ (char(6)) to the major project (MAJPROJ) for project (PROJNO) 'IF1000'. Columns PRSTAFF and MAJPROJ may contain null values, so provide indicator variables STAFF_IND (smallint) and MAJPROJ_IND (smallint).

```
SELECT PROJNAME, PRSTAFF, MAJPROJ
  INTO :PNAME, :STAFF :STAFF_IND, :MAJPROJ :MAJPROJ_IND
  FROM PROJECT
  WHERE PROJNO = 'IF1000'
```

MBCS Considerations: Whether multi-byte characters can be used in a host variable name depends on the host language.

References to BLOB, CLOB, and DBCLOB host variables

Regular BLOB, CLOB, and DBCLOB variables, LOB locator variables (see "References to locator variables" on page 826), and LOB file reference variables (see "References to BLOB, CLOB, and DBCLOB file reference variables" on page 826) can be defined in all host languages. Where LOBs are allowed, the term *host-variable* in a syntax diagram can refer to a regular host variable, a locator variable, or a file reference variable. Since these are not native data types, SQL extensions are used and the precompilers generate the host language constructs necessary to represent each variable. In the case of REXX, LOBs are mapped to strings.

It is sometimes possible to define a large enough variable to hold an entire large object value. If this is true and if there is no performance benefit to be gained by deferred transfer of data from the server, a locator is not needed. However, since host language or space restrictions will often dictate against storing an entire large object in temporary storage at one time and/or because of performance benefit, a large object may be referenced via a locator and portions of that object may be selected into or updated from host variables that contain only a portion of the large object at one time.

As with all other host variables, a large object locator variable may have an associated indicator variable. Indicator variables for large object locator host

variables behave in the same way as indicator variables for other data types. When a null value is returned from the database, the indicator variable is set and the locator host variable is unchanged. This means a locator can never point to a null value.

References to locator variables

A *locator variable* is a host variable that contains the locator representing a LOB value on the application server.

A locator variable in an SQL statement must identify a locator variable described in the program according to the rules for declaring locator variables. This is always indirectly through an SQL statement.

The term locator variable, as used in the syntax diagrams, shows a reference to a locator variable. The meta-variable *locator-variable* can be expanded to include a *host-identifier* the same as that for *host-variable*.

When the indicator variable associated with a locator is null, the value of the referenced LOB is null.

If a locator-variable that does not currently represent any value is referenced, an error is raised (SQLSTATE 0F001).

At transaction commit, or any transaction termination, all locators acquired by that transaction are released.

References to BLOB, CLOB, and DBCLOB file reference variables

BLOB, CLOB, and DBCLOB file reference variables are used for direct file input and output for LOBs, and can be defined in all host languages. Since these are not native data types, SQL extensions are used and the precompilers generate the host language constructs necessary to represent each variable. In the case of REXX, LOBs are mapped to strings.

A file reference variable represents (rather than contains) the file, just as a LOB locator represents, rather than contains, the LOB bytes. Database queries, updates and inserts may use file reference variables to store or to retrieve single column values.

A file reference variable has the following properties:

Data Type
BLOB, CLOB, or DBCLOB. This property is specified when the variable is declared.

Direction
This must be specified by the application program at run time (as part of the File Options value). The direction is one of:

- Input (used as a source of data on an EXECUTE statement, an OPEN statement, an UPDATE statement, an INSERT statement, or a DELETE statement).
- Output (used as the target of data on a FETCH statement or a SELECT INTO statement).

File name
This must be specified by the application program at run time. It is one of:

- The complete path name of the file (which is advised).
- A relative file name. If a relative file name is provided, it is appended to the current path of the client process.

Within an application, a file should only be referenced in one file reference variable.

File Name Length

This must be specified by the application program at run time. It is the length of the file name (in bytes).

File Options

An application must assign one of a number of options to a file reference variable before it makes use of that variable. Options are set by an INTEGER value in a field in the file reference variable structure. One of the following values must be specified for each file reference variable:

- Input (from client to server)

  SQL_FILE_READ
  This is a regular file that can be opened, read and closed. (The option is SQL-FILE-READ in COBOL, sql_file_read in FORTRAN, and READ in REXX.)

- Output (from server to client)

  SQL_FILE_CREATE
  Create a new file. If the file already exists, an error is returned. (The option is SQL-FILE-CREATE in COBOL, sql_file_create in FORTRAN, and CREATE in REXX.)

  SQL_FILE_OVERWRITE (Overwrite)
  If an existing file with the specified name exists, it is overwritten; otherwise a new file is created. (The option is SQL-FILE-OVERWRITE in COBOL, sql_file_overwrite in FORTRAN, and OVERWRITE in REXX.)

  SQL_FILE_APPEND
  If an existing file with the specified name exists, the output is appended to it; otherwise a new file is created. (The option is SQL-FILE-APPEND in COBOL, sql_file_append in FORTRAN, and APPEND in REXX.)

Data Length
This is unused on input. On output, the implementation sets the data length to the

length of the new data written to the file. The length is in bytes.

As with all other host variables, a file reference variable may have an associated indicator variable.

Example of an output file reference variable (in C)

Given a declare section coded as:

```
EXEC SQL BEGIN DECLARE SECTION
   SQL TYPE IS CLOB_FILE  hv_text_file;
   char  hv_patent_title[64];
EXEC SQL END DECLARE SECTION
```

Following preprocessing this would be:

```
EXEC SQL BEGIN DECLARE SECTION
   /* SQL TYPE IS CLOB_FILE  hv_text_file; */
   struct {
       unsigned long  name_length; //  File Name Length
       unsigned long  data_length; //  Data Length
       unsigned long  file_options; // File Options
       char           name[255];   // File Name
   } hv_text_file;
   char  hv_patent_title[64];
EXEC SQL END DECLARE SECTION
```

Then, the following code can be used to select from a CLOB column in the database into a new file referenced by :hv_text_file.

```
strcpy(hv_text_file.name, "/u/gainer/papers/sigmod.94");
hv_text_file.name_length = strlen("/u/gainer/papers/sigmod.94");
hv_text_file.file_options = SQL_FILE_CREATE;

EXEC SQL SELECT content INTO :hv_text_file from papers
     WHERE TITLE = 'The Relational Theory behind Juggling';
```

Example of an input file reference variable (in C)

Given the same declare section as above, the following code can be used to insert the data from a regular file referenced by :hv_text_file into a CLOB column.

```
strcpy(hv_text_file.name, "/u/gainer/patents/chips.13");
hv_text_file.name_length = strlen("/u/gainer/patents/chips.13");
hv_text_file.file_options = SQL_FILE_READ:
strcpy(:hv_patent_title, "A Method for Pipelining Chip Consumption");

EXEC SQL INSERT INTO patents( title, text )
         VALUES(:hv_patent_title, :hv_text_file);
```

References to structured type host variables

Structured type variables can be defined in all host languages except FORTRAN, REXX, and Java. Since these are not native data types, SQL extensions are used and the precompilers generate the host language constructs necessary to represent each variable.

As with all other host variables, a structured type variable may have an associated indicator variable. Indicator variables for structured type host variables behave in the same way as indicator variables for other data types. When a null value is returned from the database, the indicator variable is set and the structured type host variable is unchanged.

The actual host variable for a structured type is defined as a built-in data type. The built-in data type associated with the structured type must be assignable:

- from the result of the FROM SQL transform function for the structured type as defined by the specified TRANSFORM GROUP option of the precompile command; and
- to the parameter of the TO SQL transform function for the structured type as defined by the specified TRANSFORM GROUP option of the precompile command.

If using a parameter marker instead of a host variable, the appropriate parameter type characteristics must be specified in the SQLDA. This requires a "doubled" set of SQLVAR structures in the SQLDA, and the SQLDATATYPE_NAME field of the secondary SQLVAR must be filled with the schema and type name of the structured type. If the schema is omitted in the SQLDA structure, an error results (SQLSTATE 07002).

Example

Define the host variables *hv_poly* and *hv_point* (of type POLYGON, using built-in type BLOB(1048576)) in a C program.

```
EXEC SQL BEGIN DECLARE SECTION;
      static SQL
         TYPE IS POLYGON AS BLOB(1M)
         hv_poly, hv_point;
EXEC SQL END DECLARE SECTION;
```

Related reference:

- "CREATE ALIAS statement" in the *SQL Reference, Volume 2*
- "PREPARE statement" in the *SQL Reference, Volume 2*
- "SET SCHEMA" on page 902
- "SQL limits" in the *SQL Reference, Volume 1*
- "SQLDA (SQL descriptor area)" on page 1008
- "Reserved schema names and reserved words" in the *SQL Reference, Volume 1*
- "Japanese and traditional-Chinese extended UNIX code (EUC) considerations" in the *SQL Reference, Volume 1*
- "SQL queries" in the *SQL Reference, Volume 1*
- "Large objects (LOBs)" in the *SQL Reference, Volume 1*

Example

Chapter 21. Naming Conventions

This section provides information about the conventions that apply when naming database manager objects, such as databases and tables, 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.
- 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 is character strings that represent names under the systems network architecture (SNA). Many values, such as logical unit names (partner_lu and local_lu), are case sensitive. 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 characters in length. Column names can be 1 to 30 characters 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.
- Authentication IDs cannot exceed 30 characters on Windows 32-bit operating systems and 8 characters on all other operating systems.
- Group IDs cannot exceed 30 characters in length.
- Local aliases for remote nodes that are to be cataloged in the node directory cannot exceed eight characters in length.

© Copyright IBM Corp. 1993-2004

Part 2. User Information

© Copyright IBM Corp. 1993-2004

Chapter 22. User responsibilities for security

In DB2 Universal Database, 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 DB2 Universal Database.

  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 DB2 Universal Database, contact your manager or database administrator immediately.
- When using the CONNECT statement to connect to the database from the command line processor (CLP), allow DB2 Universal Database 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.

© Copyright IBM Corp. 1993-2004

Chapter 23. Utility Considerations

Privileges, authorities and authorization required to use export

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 or DBADM authority, or CONTROL or SELECT privilege for each table participating in the export operation.

Related reference:

- "db2Export - Export" on page 405
- "EXPORT" on page 269

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 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.

© Copyright IBM Corp. 1993-2004

Privileges, authorities, and authorization required to use 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, authorities, and authorizations required to use Load

To load data into a table, you must have one of the following:

- SYSADM authority
- DBADM authority
- 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.

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.

On Windows® NT, Windows 2000 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.

Related reference:

- "LOAD" on page 304
- "db2Load - Load" on page 437

Chapter 24. 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.

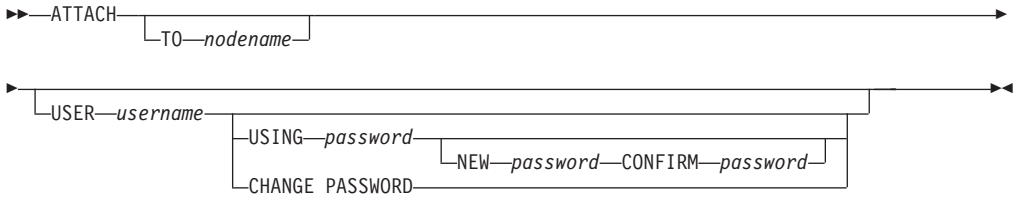
Authorization:

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 Universal Database (UDB) instance on a Windows operating system, the user name can be specified in a format compatible with Microsoft Windows NT Security Account Manager (SAM), 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. Passwords can be up to 18 characters in length. The system on which the password will be changed depends on how user authentication has been set up.

© Copyright IBM Corp. 1993-2004

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.

Related reference:

- "DETACH" on page 840

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
Related reference:
- "ATTACH" on page 839

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.

Related reference:

- "SET CLIENT Command" in the *Command Reference*
- "UPDATE ALTERNATE SERVER FOR DATABASE Command" in the *Command Reference*

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:

- *sysadm* or *dbadm* authority
- BINDADD privilege if a package does not exist, and one of:
  - IMPLICIT_SCHEMA authority on the database if the schema name of the package does not exist
  - CREATEIN privilege on the schema if the schema name of the package exists
- ALTERIN privilege on the schema if the package exists
- BIND privilege on the package if it exists.

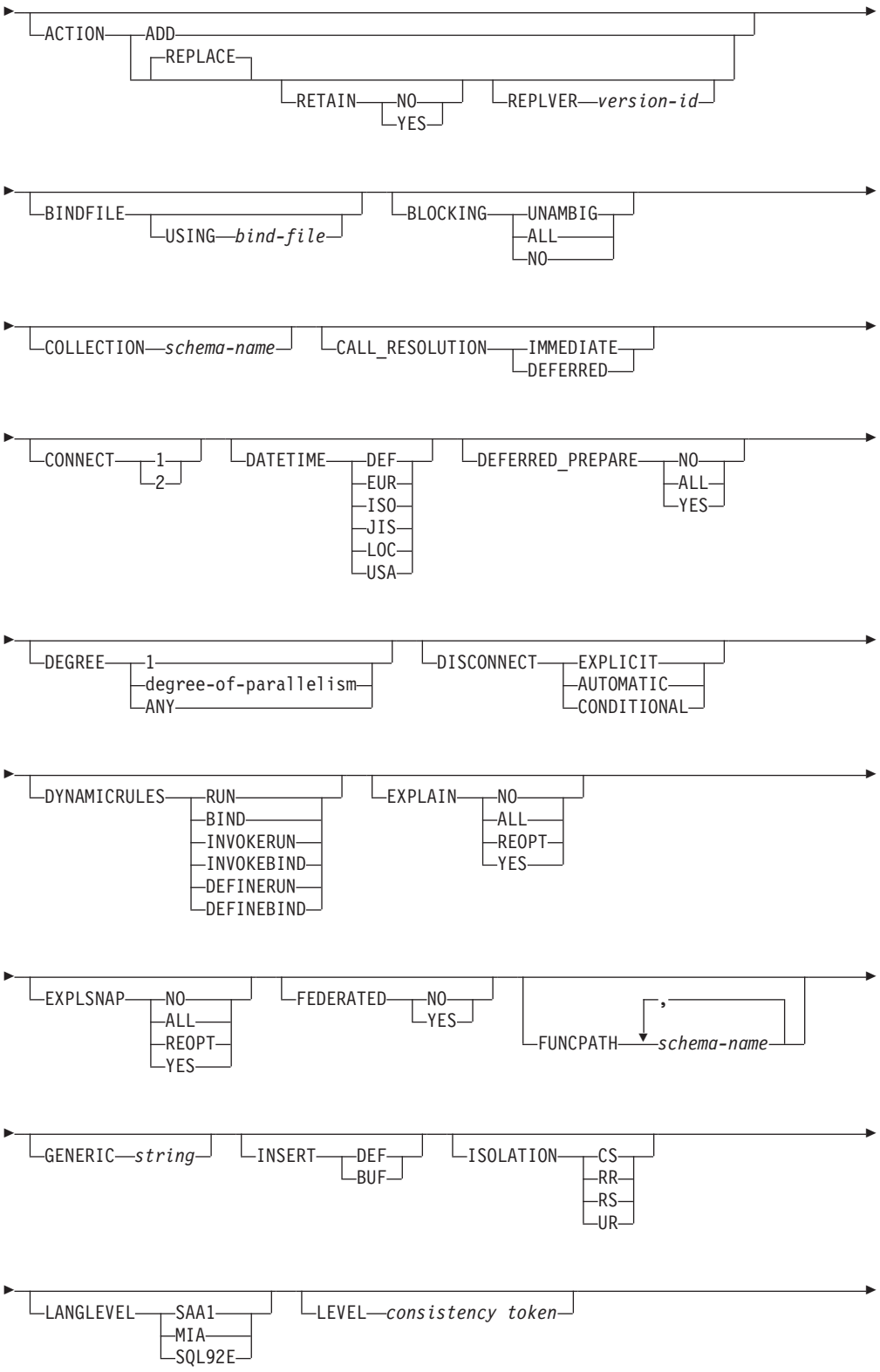
The user also needs all privileges required to compile any static SQL statements in the application. Privileges granted to groups are not used for authorization checking of static statements. If the user has *sysadm* authority, but not explicit privileges to complete the bind, the database manager grants explicit *dbadm* authority automatically.

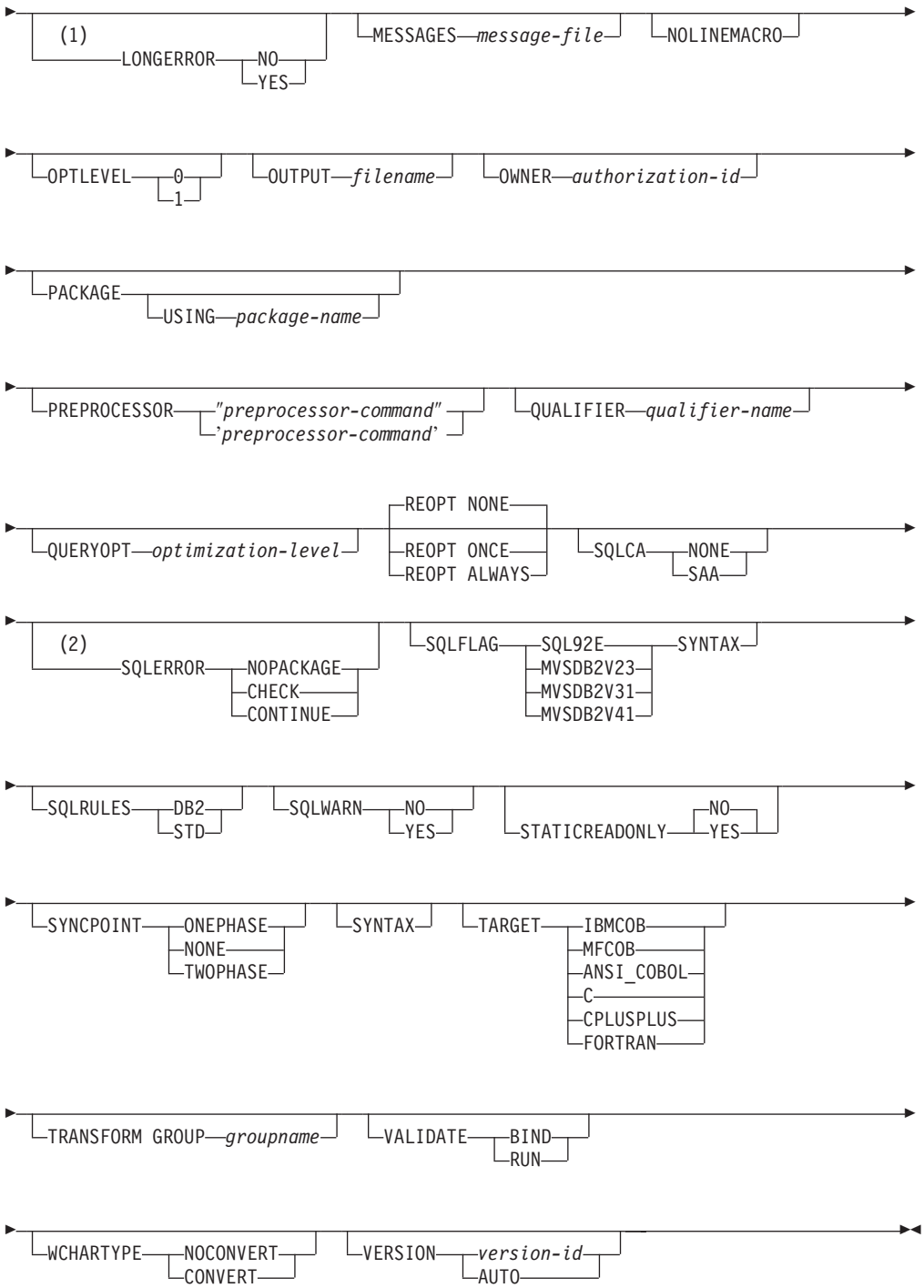
Required connection:

Database. If implicit connect is enabled, a connection to the default database is established.

Command syntax:

For DB2 for Windows and UNIX


ACTION
ADD
REPLACE
RETAIN
NO
YES
REPLVER
version-id
BINDFILE
USING
bind-file
BLOCKING
UNAMBIG
ALL
NO
COLLECTION
schema-name
CALL_RESOLUTION
IMMEDIATE
DEFERRED
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
REOPT
YES
EXPLSNAP
NO
ALL
REOPT
YES
FEDERATED
NO
YES
FUNCPATH
,
schema-name
GENERIC
string
INSERT
DEF
BUF
ISOLATION
CS
RR
RS
UR
LANGLEVEL
SAA1
MIA
SQL92E
LEVEL
consistency token

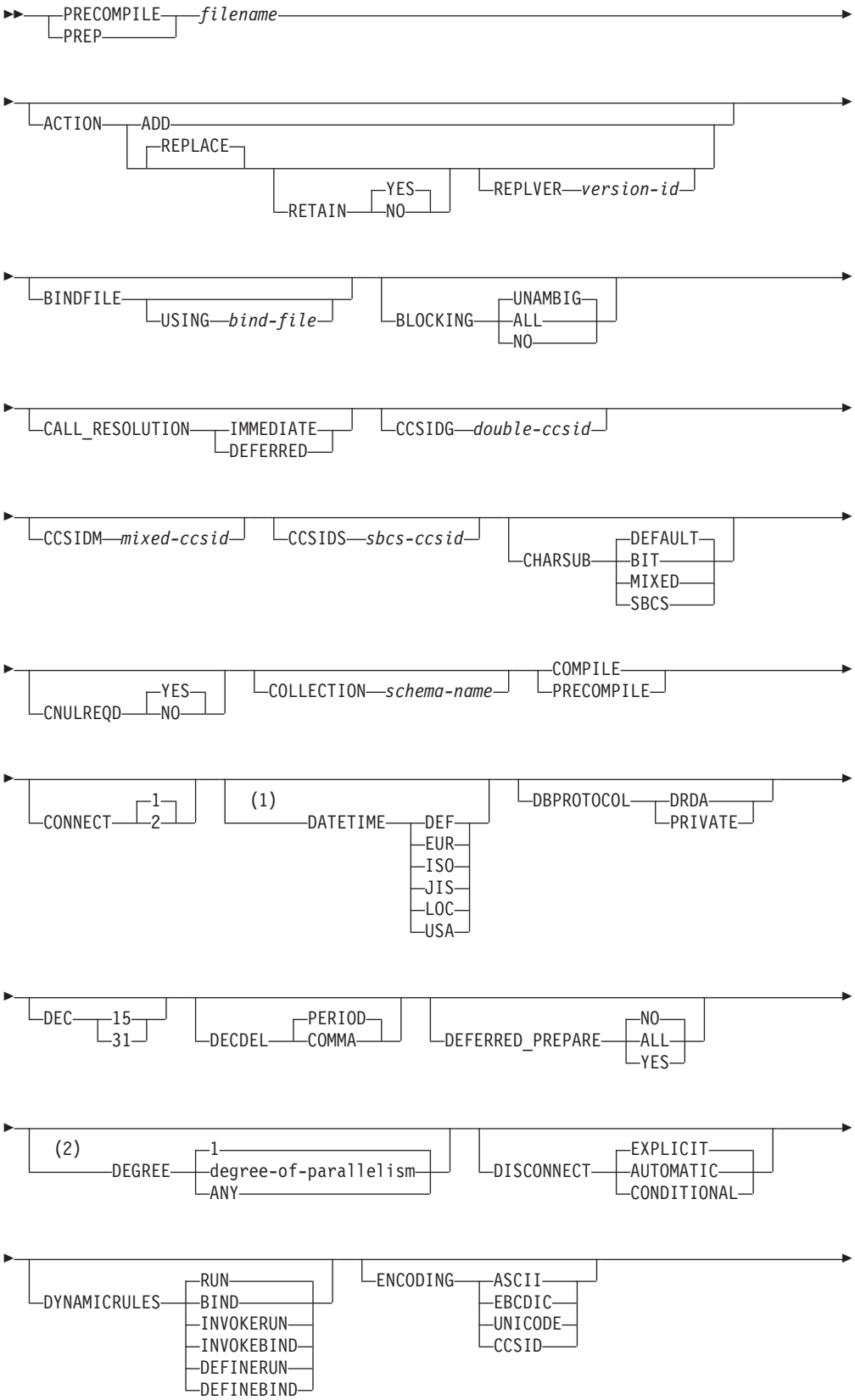


Notes:

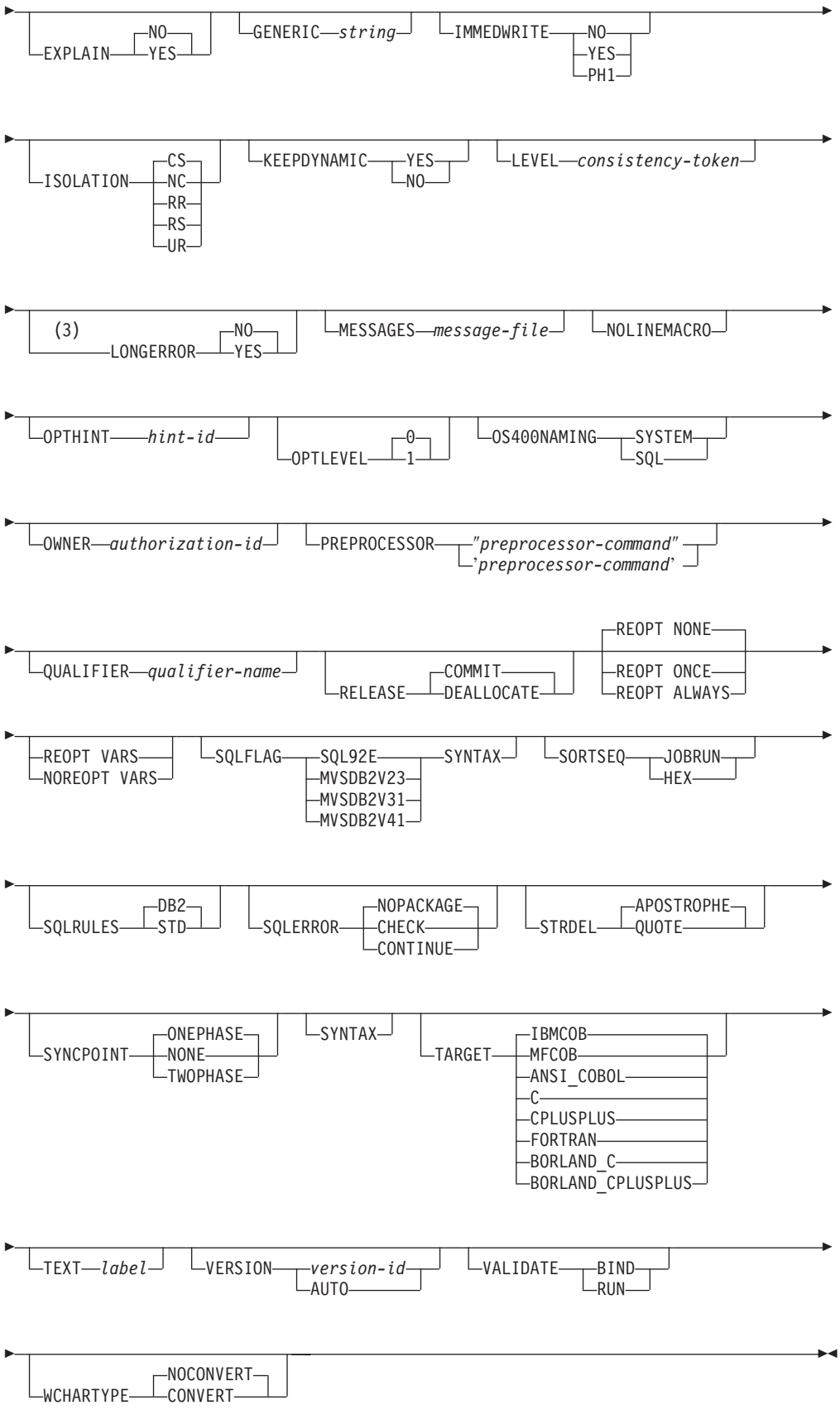
1 NO is the default for 32 bit systems and for 64 bit NT systems where long host variables can be used as declarations for INTEGER columns. YES is the default for 64 bit UNIX systems.

2 SYNTAX is a synonym for SQLERROR(CHECK).

For DB2 on servers other than Windows and UNIX


PRECOMPILE
PREP
filename
ACTION
ADD
REPLACE
RETAIN
YES
NO
REPLVER
version-id
BINDFILE
USING
bind-file
BLOCKING
UNAMBIG
ALL
NO
CALL_RESOLUTION
IMMEDIATE
DEFERRED
CCSIDG
double-ccsid
CCSIDM
mixed-ccsid
CCSIDS
sbcs-ccsid
CHARSUB
DEFAULT
BIT
MIXED
SBCS
CNULREQD
YES
NO
COLLECTION
schema-name
COMPILE
PRECOMPILE
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

PRECOMPILE


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'
QUALIFIER qualifier-name
RELEASE
COMMIT
DEALLOCATE
REOPT NONE
REOPT ONCE
REOPT ALWAYS
REOPT VARS
NOREOPT VARS
SQLFLAG
SQL92E
MVSDB2V23
MVSDB2V31
MVSDB2V41
SYNTAX
SORTSEQ
JOBRUN
HEX
SQLRULES
DB2
STD
SQLERROR
NOPACKAGE
CHECK
CONTINUE
STRDEL
APOSTROPHE
QUOTE
SYNCPOINT
ONEPHASE
NONE
TWOPHASE
SYNTAX
TARGET
IBMCOB
MFCOB
ANSI_COBOL
C
CPLUSPLUS
FORTRAN
BORLAND_C
BORLAND_CPLUSPLUS
TEXT label
VERSION
version-id
AUTO
VALIDATE
BIND
RUN
WCHARTYPE
NOCONVERT
CONVERT

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 based systems) must be specified for C++ applications (generates a `.cxx` file on Windows operating systems, or a `.C` file on UNIX based 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 based systems).

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.

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.

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 authentication 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.

ALL Specifies to block for:
- Read-only cursors
- Cursors not specified as FOR UPDATE OF

Ambiguous cursors are treated as read-only.

NO Specifies not to block any cursors. Ambiguous cursors are treated as updatable.

UNAMBIG
Specifies to block for:
- Read-only cursors
- Cursors not specified as FOR UPDATE OF

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. Note that 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 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.

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 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.

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 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.

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 30-character collection identifier for the package. If not specified, the authorization identifier for the user processing the package is used.

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.

Note: 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).

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).

Note: 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 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.

Note: 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.

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 number of schemas that can be specified is limited by the length of the resulting function path, which cannot exceed 254 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 1023 bytes.

IMMEDWRITE
Indicates whether immediate writes will be done for updates made to group buffer pool dependent pagesets or 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.

Note: 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.

Note: 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.

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 UDB for iSeries data. Supported by DB2 UDB for iSeries only. For a list of supported option values, refer to the documentation for DB2 for iSeries.

Please note that 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 iSeries 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 iSeries system naming convention is used, whether or not it has been precompiled or bound using the OS400NAMING SYSTEM option.

OWNER authorization-id
Designates a 30-character 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 SYSADM or 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.

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 8 characters.

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 30-character 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, and special registers. Valid values are:

NONE
The access path for a given SQL statement containing host variables, parameter markers 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 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 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 back-level compatibility. Specifies whether to have DB2 determine an access path at run time using values for host 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 iSeries system. Supported by DB2 UDB for iSeries only. For a list of supported option values, refer to the documentation for DB2 for iSeries.

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 a user or application can specify the format of LOB answer set columns.

DB2

- Permits the SQL CONNECT statement to switch the current connection to another established (*dormant*) connection.
- The user or application can specify the format of a LOB column only during the first fetch request.

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 user or application can change the format of a LOB column with each fetch request.

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). This DB2 precompile/bind option is not supported by DRDA.

NO Warnings will not be returned from the SQL compiler.

YES Warnings will be returned from the SQL compiler.

Note: 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.

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.

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 UNIX platforms and Windows NT.

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 18 characters 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.

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.

Related concepts:

- "Authorization Considerations for Dynamic SQL" on page 951
- "WCHARTYPE Precompiler Option in C and C++" in the *Application Development Guide: Programming Client Applications*
- "Effect of DYNAMICRULES bind option on dynamic SQL" on page 952
- "Effects of REOPT on static SQL" in the *Application Development Guide: Programming Client Applications*
- "Effects of REOPT on dynamic SQL" in the *Application Development Guide: Programming Client Applications*

Related tasks:

- "Specifying row blocking to reduce overhead" in the *Administration Guide: Performance*

Related reference:

- "SET CURRENT QUERY OPTIMIZATION statement" in the *SQL Reference, Volume 2*
- "BIND" on page 232
- "Datetime values" in the *SQL Reference, Volume 1*

REBIND

Allows the user to recreate a package stored in the database without the need for a bind file.

Authorization:

One of the following:

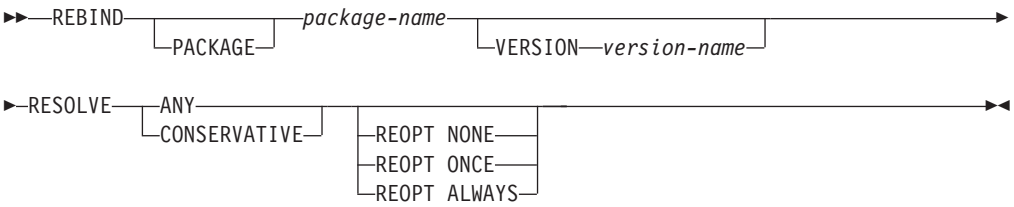
- *sysadm* or *dbadm* authority
- ALTERIN privilege on the schema
- BIND privilege on the package.

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 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 functions and data types are considered during function resolution and type resolution on static DML statements in the package. This option is not supported by DRDA. Valid values are:

ANY Any of the functions and types in the SQL path are considered for function and type resolution. Conservative binding semantics are not used. This is the default.

CONSERVATIVE
Only functions and types in the SQL path that were defined before

the last explicit bind time stamp are considered for function and type resolution. Conservative binding semantics are used. This option is not supported for an inoperative package.

REOPT
Specifies whether to have DB2 optimize an access path using values for host variables, parameter markers, and special registers.

NONE
The access path for a given SQL statement containing host variables, parameter markers 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 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 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.

Note: 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 migration, all packages stored in the database will be invalidated by the DB2 Version 8 migration process. 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.

Related reference:

- "BIND" on page 232
- "RUNSTATS Command" in the *Command Reference*
- "db2rbind - Rebind all Packages" on page 263

Chapter 25. DB2 UDB APIs for Users

Following are the application programming interfaces (APIs) that correspond to the DB2 UDB commands that are used for the Common Criteria evaluation. Note that:

- APIs are not used for the Common Criteria certification. The APIs are included in this document for reasons of completeness only.
- Not every API has a corresponding command, and vice versa.

sqlaprep - Precompile 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:

- *sysadm* or *dbadm* authority
- BINDADD privilege if a package does not exist and one of:
  - IMPLICIT_SCHEMA authority on the database if the schema name of the package does not exist
  - CREATEIN privilege on the schema if the schema name of the package exists
- ALTERIN privilege on the schema if the package exists
- BIND privilege on the package if it exists.

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. If the user has *sysadm* authority, but not explicit privileges to complete the bind, the database manager grants explicit *dbadm* authority automatically.

Required connection:

Database

API include file:

sql.h

C API syntax:

© Copyright IBM Corp. 1993-2004

```
/* File: sql.h */
/* API: sqlaprep */
/* ... */
SQL_API_RC SQL_API_FN
  sqlaprep (
    _SQLOLDCHAR *pProgramName,
    _SQLOLDCHAR *pMsgFileName,
    struct sqlopt *pPrepOptions,
    struct sqlca *pSqlca);
/* ... */
```

Generic API syntax:

```
/* File: sql.h */
/* API: sqlgprep */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgprep (
    unsigned short MsgFileNameLen,
    unsigned short ProgramNameLen,
    struct sqlca *pSqlca,
    struct sqlopt *pPrepOptions,
    _SQLOLDCHAR *pMsgFileName,
    _SQLOLDCHAR *pProgramName);
/* ... */
```

API parameters:

MsgFileNameLen
Input. A 2-byte unsigned integer representing the length of the message file name in bytes.

ProgramNameLen
Input. A 2-byte unsigned integer representing the length of the program name in bytes.

pSqlca
Output. A pointer to the *sqlca* structure.

pPrepOptions
Input. A structure used to pass precompile options to the API. For more information about this structure, see SQLOPT.

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.

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.

REXX API syntax:

This API can be called from REXX through the SQLDB2 interface.

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`.

Related reference:

- "sqlabndx - Bind" on page 484
- "SQLCA" in the *Administrative API Reference*
- "SQLOPT" in the *Administrative API Reference*

Related samples:

- "dbpkg.sqc -- How to work with packages (C)"
- "dbpkg.sqC -- How to work with packages (C++)"

sqlarbnd - Rebind

Allows the user to recreate a package stored in the database without the need for a bind file.

Authorization:

One of the following:

- *sysadm* or *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

C API syntax:

```
/* File: sql.h */
/* API: sqlarbnd */
/* ... */
SQL_API_RC SQL_API_FN
sqlarbnd (
  char *pPackageName,
  struct sqlca *pSqlca,
  struct sqlopt *pRebindOptions);
/* ... */
```

Generic API syntax:

```
/* File: sql.h */
/* API: sqlgrbnd */
/* ... */
SQL_API_RC SQL_API_FN
sqlgrbnd (
  unsigned short PackageNameLen,
  char *pPackageName,
  struct sqlca *pSqlca,
  struct sqlopt *pRebindOptions);
/* ... */
```

API parameters:

PackageNameLen
Input. A 2-byte unsigned integer representing the length of the package name in bytes.

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_OPT_VERSION 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.

REXX API syntax:

This API can be called from REXX through the SQLDB2 interface.

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. 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 migration, all packages stored in the database will be invalidated by the DB2 Version 5 migration process. 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_OPT_VERSION 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`.

Related tasks:

- "Registering SQLEXEC, SQLDBS and SQLDB2 in REXX" in the *Application Development Guide: Programming Client Applications*

Related reference:

- "sqlabndx - Bind" on page 484
- "SQLCA" in the *Administrative API Reference*
- "SQLOPT" in the *Administrative API Reference*
- "REBIND" on page 866
- "db2rbind - Rebind all Packages" on page 263
- "db2Runstats - Runstats" in the *Administrative API Reference*

Related samples:

- "dbpkg.sqc -- How to work with packages (C)"
- "dbsample.sqc -- Creates a sample database (C)"
- "dbpkg.sqC -- How to work with packages (C++)"
- "rebind.sqb -- How to rebind a package (IBM COBOL)"

sqleatcp - Attach 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.

Authorization:

None

Required connection:

This API establishes an instance attachment.

API include file:

sqlenv.h

C API syntax:

```
/* File: sqlenv.h */
/* API: sqleatcp */
/* ... */
SQL_API_RC SQL_API_FN
  sqleatcp (
    char *pNodeName,
    char *pUserName,
    char *pPassword,
    char *pNewPassword,
    struct sqlca *pSqlca);
/* ... */
```

Generic API syntax:

```
/* File: sqlenv.h */
/* API: sqlgatcp */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgatcp (
    unsigned short NewPasswordLen,
    unsigned short PasswordLen,
    unsigned short UserNameLen,
    unsigned short NodeNameLen,
    struct sqlca *pSqlca,
    char *pNewPassword,
    char *pPassword,
    char *pUserName,
    char *pNodeName);
/* ... */
```

API parameters:

NewPasswordLen
Input. A 2-byte unsigned integer representing the length of the new password in bytes. Set to zero if no new password 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.

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.

pSqlca
Output. A pointer to the *sqlca* structure.

pNewPassword
Input. A string containing the new password for the specified user name. Set to NULL if a password change is not required.

pPassword
Input. A string containing the password for the specified user name. May be NULL.

pUserName
Input. A string containing the user name under which the attachment is to be authenticated. May be NULL.

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.

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.

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 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.

Related reference:

- "sqlesetc - Set Client" in the *Administrative API Reference*
- "sqleatin - Attach" on page 879
- "sqledtin - Detach" on page 882
- "SQLCA" in the *Administrative API Reference*
- "SQLE-CONN-SETTING" in the *Administrative API Reference*

Related samples:

- "dbinst.cbl -- Attach to and detach from an instance (IBM COBOL)"
- "inattach.c -- Attach to and detach from an instance (C)"
- "inattach.C -- Attach to and detach from an instance (C++)"

sqleatin - Attach

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

C API syntax:

```
/* File: sqlenv.h */
/* API: sqleatin */
/* ... */
SQL_API_RC SQL_API_FN
  sqleatin (
    char *pNodeName,
    char *pUserName,
    char *pPassword,
    struct sqlca *pSqlca);
/* ... */
```

Generic API syntax:

```
/* File: sqlenv.h */
/* API: sqlgatin */
/* ... */
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);
/* ... */
```

API 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.

pSqlca
Output. A pointer to the *sqlca* structure.

pPassword
Input. A string containing the password for the specified user name. May be NULL.

pUserName
Input. A string containing the user name under which the attachment is to be authenticated. May be NULL.

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.

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.

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 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.

Related reference:

- "sqlesetc - Set Client" in the *Administrative API Reference*
- "sqledtin - Detach" on page 882
- "sqleatcp - Attach and Change Password" on page 876
- "SQLCA" in the *Administrative API Reference*
- "SQLE-CONN-SETTING" in the *Administrative API Reference*

Related samples:

- "dbinst.cbl -- Attach to and detach from an instance (IBM COBOL)"
- "inattach.c -- Attach to and detach from an instance (C)"

- "utilapi.c -- Error-checking utility for non-embedded SQL samples in C (C)"
- "inattach.C -- Attach to and detach from an instance (C++)"
- "utilapi.C -- Checks for and prints to the screen SQL warnings and errors (C++)"

sqledtin - Detach

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

C API syntax:

```
/* File: sqlenv.h */
/* API: sqledtin */
/* ... */
SQL_API_RC SQL_API_FN
  sqledtin (
    struct sqlca *pSqlca);
/* ... */
```

Generic API syntax:

```
/* File: sqlenv.h */
/* API: sqlgdtin */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgdtin (
    struct sqlca *pSqlca);
/* ... */
```

API parameters:

pSqlca
Output. A pointer to the *sqlca* structure.

REXX API syntax:

```
DETACH
```

Related reference:

- "sqleatin - Attach" on page 879
- "SQLCA" in the *Administrative API Reference*

Related samples:

- "dbinst.cbl -- Attach to and detach from an instance (IBM COBOL)"
- "inattach.c -- Attach to and detach from an instance (C)"
- "utilapi.c -- Error-checking utility for non-embedded SQL samples in C (C)"

- "inattach.C -- Attach to and detach from an instance (C++)"
- "utilapi.C -- Checks for and prints to the screen SQL warnings and errors (C++)"

Chapter 26. SQL Statements for Users

COMMIT

The COMMIT statement terminates a unit of work and commits the database changes that were made by that unit of work.

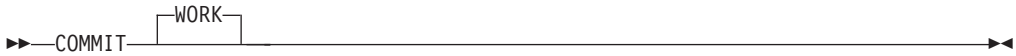
Invocation:

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared.

Authorization:

None required.

Syntax:



Description:
The unit of work in which the COMMIT statement is executed is terminated and a new unit of work is initiated. All changes made by the following statements executed during the unit of work are committed: ALTER, COMMENT, CREATE, DROP, GRANT, LOCK TABLE, REVOKE, SET INTEGRITY, SET Variable, and the data change statements (INSERT, DELETE, MERGE, UPDATE), including those nested in a query.

The following statements, however, are not under transaction control and changes made by them are independent of the COMMIT statement:

- SET CONNECTION
- SET CURRENT DEFAULT TRANSFORM GROUP
- SET CURRENT DEGREE
- SET CURRENT EXPLAIN MODE
- SET CURRENT EXPLAIN SNAPSHOT

© Copyright IBM Corp. 1993-2004

- SET CURRENT LOCK TIMEOUT
- SET CURRENT PACKAGESET
- SET CURRENT QUERY OPTIMIZATION
- SET CURRENT REFRESH AGE
- SET EVENT MONITOR STATE
- SET PASSTHRU

  Note: Although the SET PASSTHRU statement is not under transaction control, the passthru session initiated by the statement is under transaction control.
- SET PATH
- SET SCHEMA
- SET SERVER OPTION

All locks acquired by the unit of work subsequent to its initiation are released, except necessary locks for open cursors that are declared WITH HOLD. All open cursors not defined WITH HOLD are closed. Open cursors defined WITH HOLD remain open, and the cursor is positioned before the next logical row of the result table. (A FETCH must be performed before a positioned UPDATE or DELETE statement is issued.) All LOB locators are freed. Note that this is true even when the locators are associated with LOB values retrieved via a cursor that has the WITH HOLD property.

All savepoints set within the transaction are released.

Notes:

- It is strongly recommended that each application process explicitly ends its unit of work before terminating. If the application program ends normally without a COMMIT or ROLLBACK statement then the database manager attempts a commit or rollback depending on the application environment.
- For information on the impact of COMMIT on cached dynamic SQL statements, see "EXECUTE".
- For information on potential impacts of COMMIT on declared temporary tables, see "DECLARE GLOBAL TEMPORARY TABLE".

Example:

Commit alterations to the database made since the last commit point.

```
COMMIT WORK
```

Related reference:

- "EXECUTE statement" in the *SQL Reference, Volume 2*
- "DECLARE GLOBAL TEMPORARY TABLE statement" in the *SQL Reference, Volume 2*

Related samples:

- "dynamic.sqb -- How to update table data with cursor dynamically (MF COBOL)"
- "tbconstr.sqc -- How to create, use, and drop constraints (C)"
- "tbsavept.sqc -- How to use external savepoints (C)"
- "tut_mod.sqc -- How to modify table data (C)"

- "tut_use.sqc -- How to modify a database (C)"
- "tbconstr.sqC -- How to create, use, and drop constraints (C++)"
- "tut_mod.sqC -- How to modify table data (C++)"
- "tut_use.sqC -- How to modify a database (C++)"

CONNECT (Type 1)

The CONNECT (Type 1) statement connects an application process to the identified application server according to the rules for remote unit of work.

An application process can only be connected to one application server at a time. This is called the *current server*. A default application server may be established when the application requester is initialized. If implicit connect is available and an application process is started, it is implicitly connected to the default application server. The application process can explicitly connect to a different application server by issuing a CONNECT TO statement. A connection lasts until a CONNECT RESET statement or a DISCONNECT statement is issued or until another CONNECT TO statement changes the application server.

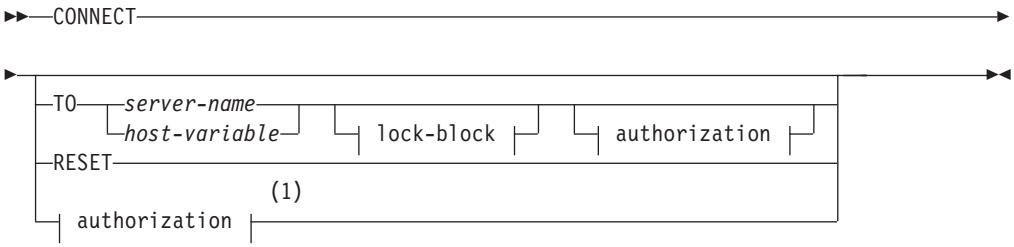
Invocation:

Although an interactive SQL facility might provide an interface that gives the appearance of interactive execution, this statement can only be embedded within an application program. It is an executable statement that cannot be dynamically prepared.

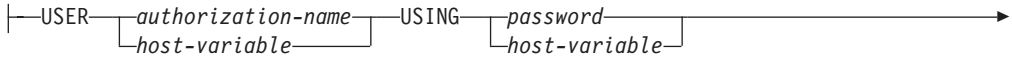
Authorization:

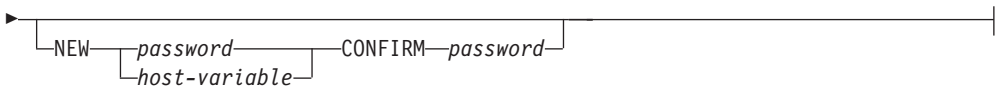
The authorization ID of the statement must be authorized to connect to the identified application server. Depending on the authentication setting for the database, the authorization check may be performed by either the client or the server. For a partitioned database, the user and group definitions must be identical across partitions.

Syntax:

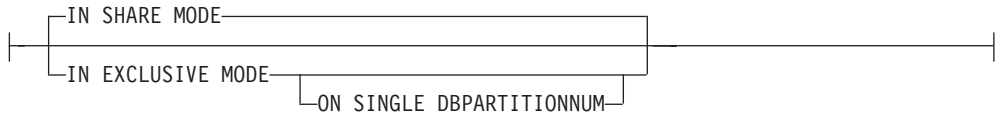


authorization:





lock-block:



Notes:

1 This form is only valid if implicit connect is enabled.

Description:

CONNECT (with no operand)
Returns information about the current server. The information is returned in the SQLERRP field of the SQLCA as described in "Successful Connection".

If a connection state exists, the authorization ID and database alias are placed in the SQLERRMC field of the SQLCA. If the authorization ID is longer than 8 bytes, it will be truncated to 8 bytes, and the truncation will be flagged in the SQLWARN0 and SQLWARN1 fields of the SQLCA, with 'W' and 'A', respectively. If the database configuration parameter DYN_QUERY_MGMT is enabled, then the SQLWARN0 and SQLWARN7 fields of the SQLCA will be flagged with 'W' and 'E', respectively.

If no connection exists and implicit connect is possible, then an attempt to make an implicit connection is made. If implicit connect is not available, this attempt results in an error (no existing connection). If no connection, then the SQLERRMC field is blank.

The territory code and code page of the application server are placed in the SQLERRMC field (as they are with a successful CONNECT TO statement).

This form of CONNECT:

- Does not require the application process to be in the connectable state.
- If connected, does not change the connection state.
- If unconnected and implicit connect is available, a connection to the default application server is made. In this case, the country or region code and code page of the application server are placed in the SQLERRMC field, like a successful CONNECT TO statement.
- If unconnected and implicit connect is not available, the application process remains unconnected.
- Does not close cursors.

TO *server-name* or *host-variable*
Identifies the application server by the specified *server-name* or a *host-variable* which contains the server-name.

If a *host-variable* is specified, it must be a character string variable with a length attribute that is not greater than 8, and it must not include an indicator variable. The *server-name* that is contained within the *host-variable* must be left-justified and must not be delimited by quotation marks.

Note that the *server-name* is a database alias identifying the application server. It must be listed in the application requester's local directory.

Note: DB2 UDB for OS/390 and z/OS supports a 16-byte location name, and DB2 UDB for iSeries supports an 18-byte target database name. DB2 Version 8 only supports the use of an 8-byte database alias name on the SQL CONNECT statement. However, the database alias name can be mapped to an 18-byte database name through the Database Connection Service Directory.

When the CONNECT TO statement is executed, the application process must be in the connectable state.

Successful Connection:

If the CONNECT TO statement is successful:

- All open cursors are closed, all prepared statements are destroyed, and all locks are released from the previous application server.
- The application process is disconnected from its previous application server, if any, and connected to the identified application server.
- The actual name of the application server (not an alias) is placed in the CURRENT SERVER special register.
- Information about the application server is placed in the SQLERRP field of the SQLCA. If the application server is an IBM product, the information has the form *pppvvrrm*, where:
  - *ppp* identifies the product as follows:
    - DSN for DB2 UDB for OS/390 and z/OS
    - ARI for DB2 Server for VSE & VM
    - QSQ for DB2 UDB for iSeries
    - SQL for DB2 UDB for UNIX and Windows
  - *vv* is a two-digit version identifier, such as '08'
  - *rr* is a two-digit release identifier, such as '01'
  - *m* is a one-digit modification level identifier, such as '0'.

  This release (Version 8) of DB2 UDB for UNIX and Windows is identified as 'SQL08010'.
- The SQLERRMC field of the SQLCA is set to contain the following values (separated by X'FF')
  1. the country or region code of the application server (or blanks if using DB2 Connect),
  2. the code page of the application server (or CCSID if using DB2 Connect),
  3. the authorization ID (up to first 8 bytes only),
  4. the database alias,
  5. the platform type of the application server. Currently identified values are:

Token	Server
QAS	DB2 Universal Database for iSeries
QDB2	DB2 Universal Database for OS/390 and z/OS
QDB2/2	DB2 Universal Database for OS/2
QDB2/6000	DB2 Universal Database for AIX

QDB2/HPUX	DB2 Universal Database for HP-UX
QDB2/LINUX	DB2 Universal Database for Linux
QDB2/NT	DB2 Universal Database for Windows NT, 2000, and XP
QDB2/SUN	DB2 Universal Database for Solaris Operating System
QSQLDS/VM	DB2 Server for VM
QSQLDS/VSE	DB2 Server for VSE

6. The agent ID. It identifies the agent executing within the database manager on behalf of the application. This field is the same as the agent_id element returned by the database monitor.
7. The agent index. It identifies the index of the agent and is used for service.
8. Partition number. For a non-partitioned database, this is always 0, if present.
9. The code page of the application client.
10. Number of partitions in a partitioned database. If the database cannot be partitioned, the value is 0 (zero). Token is present only with Version 5 or later.

- The SQLERRD(1) field of the SQLCA indicates the maximum expected difference in length of mixed character data (CHAR data types) when converted to the database code page from the application code page. A value of 0 or 1 indicates no expansion; a value greater than 1 indicates a possible expansion in length; a negative value indicates a possible contraction.
- The SQLERRD(2) field of the SQLCA indicates the maximum expected difference in length of mixed character data (CHAR data types) when converted to the application code page from the database code page. A value of 0 or 1 indicates no expansion; a value greater than 1 indicates a possible expansion in length; a negative value indicates a possible contraction.
- The SQLERRD(3) field of the SQLCA indicates whether or not the database on the connection is updatable. A database is initially updatable, but is changed to read-only if a unit of work determines the authorization ID cannot perform updates. The value is one of:
  - 1 - updatable
  - 2 - read-only
- The SQLERRD(4) field of the SQLCA returns certain characteristics of the connection. The value is one of:

  0 N/A (only possible if running from a down-level client that is one-phase commit and is an updater).

  1 one-phase commit.

  2 one-phase commit; read-only (only applicable to connections to DRDA1 databases in a TP Monitor environment).

  3 two-phase commit.
- The SQLERRD(5) field of the SQLCA returns the authentication type for the connection. The value is one of:

  0 Authenticated on the server.

  1 Authenticated on the client.

2 Authenticated using DB2 Connect.

4 Authenticated on the server with encryption.

5 Authenticated using DB2 Connect with encryption.

7 Authenticated using an external Kerberos security mechanism.

8 Authenticated using an external Kerberos security mechanism or on the server with encryption.

9 Authenticated using an external GSS API plug-in security mechanism.

10 Authenticated using an external GSS API plug-in security mechanism or on the server with encryption.

255 Authentication not specified.

- The SQLERRD(6) field of the SQLCA returns the partition number of the partition to which the connection was made if the database is partitioned. Otherwise, a value of 0 is returned.
- The SQLWARN1 field in the SQLCA will be set to 'A' if the authorization ID of the successful connection is longer than 8 bytes. This indicates that truncation has occurred. The SQLWARN0 field in the SQLCA will be set to 'W' to indicate this warning.
- The SQLWARN7 field in the SQLCA will be set to 'E' if the database configuration parameter DYN_QUERY_MGMT for the database is enabled. The SQLWARN0 field in the SQLCA will be set to 'W' to indicate this warning.

Unsuccessful Connection:

If the CONNECT TO statement is unsuccessful:

- The SQLERRP field of the SQLCA is set to the name of the module at the application requester that detected the error. The first three characters of the module name identify the product.
- If the CONNECT TO statement is unsuccessful because the application process is not in the connectable state, the connection state of the application process is unchanged.
- If the CONNECT TO statement is unsuccessful because the *server-name* is not listed in the local directory, an error message (SQLSTATE 08001) is issued and the connection state of the application process remains unchanged:
  - If the application requester was not connected to an application server then the application process remains unconnected.
  - If the application requester was already connected to an application server, the application process remains connected to that application server. Any further statements are executed at that application server.
- If the CONNECT TO statement is unsuccessful for any other reason, the application process is placed into the unconnected state.

IN SHARE MODE
Allows other concurrent connections to the database and prevents other users from connecting to the database in exclusive mode.

IN EXCLUSIVE MODE
Prevents concurrent application processes from executing any operations at the application server, unless they have the same authorization ID as the user holding the exclusive lock. This option is not supported by DB2 Connect.

ON SINGLE DBPARTITIONNUM
Specifies that the coordinator database partition is connected in exclusive mode and all other database partitions are connected in share mode. This option is only effective in a partitioned database.

RESET
Disconnects the application process from the current server. A commit operation is performed. If implicit connect is available, the application process remains unconnected until an SQL statement is issued.

USER *authorization-name/host-variable*
Identifies the user ID trying to connect to the application server. If a host-variable is specified, it must be a character string variable with a length attribute that is not greater than 8, and it must not include an indicator variable. The user ID that is contained within the *host-variable* must be left justified and must not be delimited by quotation marks.

USING *password/host-variable*
Identifies the password of the user ID trying to connect to the application server. *Password* or *host-variable* may be up to 18 characters. If a host variable is specified, it must be a character string variable with a length attribute not greater than 18 and it must not include an indicator variable.

NEW *password/host-variable* **CONFIRM** *password*
Identifies the new password that should be assigned to the user ID identified by the USER option. *Password* or *host-variable* may be up to 18 characters. If a host variable is specified, it must be a character string variable with a length attribute not greater than 18 and it must not include an indicator variable. The system on which the password will be changed depends on how user authentication is set up.

Notes:

- ***Compatibilities***
  - For compatibility with previous versions of DB2:
    - NODE can be specified in place of DBPARTITIONNUM
- It is good practice for the first SQL statement executed by an application process to be the CONNECT TO statement.
- If a CONNECT TO statement is issued to the current application server with a different user ID and password then the conversation is deallocated and reallocated. All cursors are closed by the database manager (with the loss of the cursor position if the WITH HOLD option was used).
- If a CONNECT TO statement is issued to the current application server with the same user ID and password then the conversation is not deallocated and reallocated. Cursors, in this case, are not closed.
- To use a multiple-partition database environment, the user or application must connect to one of the partitions listed in the `db2nodes.cfg` file. You should try to ensure that not all users use the same partition as the coordinator partition.
- The *authorization-name* SYSTEM cannot be explicitly specified in the CONNECT statement. However, on Windows operating systems, local applications running under the Local System Account can implicitly connect to the database, such that the user ID is SYSTEM.
- When connecting to Windows Server explicitly, the *authorization-name* or user *host-variable* can be specified using the Microsoft Windows NT Security Account Manager (SAM)-compatible name; for example, 'Domain\User'.

Examples:

Example 1: In a C program, connect to the application server TOROLAB, using database alias TOROLAB, user ID FERMAT, and password THEOREM.

```
EXEC SQL  CONNECT TO TOROLAB USER FERMAT USING THEOREM;
```

Example 2: In a C program, connect to an application server whose database alias is stored in the host variable APP_SERVER (varchar(8)). Following a successful connection, copy the 3-character product identifier of the application server to the variable PRODUCT (char(3)).

```
EXEC SQL  CONNECT TO :APP_SERVER;
if (strncmp(SQLSTATE,'00000',5))
  strncpy(PRODUCT,sqlca.sqlerrp,3);
```

Related concepts:

- "Distributed relational databases" in the *SQL Reference, Volume 1*
- "Data partitioning across multiple partitions" in the *SQL Reference, Volume 1*

Related samples:

- "advsql.sqb -- How to read table data using CASE (MF COBOL)"
- "dbmcon.sqc -- How to use multiple databases (C)"
- "dbmcon.sqC -- How to use multiple databases (C++)"

CONNECT (Type 2)

The CONNECT (Type 2) statement connects an application process to the identified application server and establishes the rules for application-directed distributed unit of work. This server is then the current server for the process.

Most aspects of a CONNECT (Type 1) statement also apply to a CONNECT (Type 2) statement. Rather than repeating that material here, this section describes only those elements of Type 2 that differ from Type 1.

Invocation:

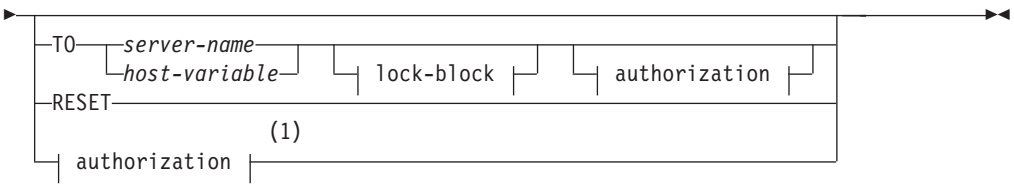
Although an interactive SQL facility might provide an interface that gives the appearance of interactive execution, this statement can only be embedded within an application program. It is an executable statement that cannot be dynamically prepared.

Authorization:

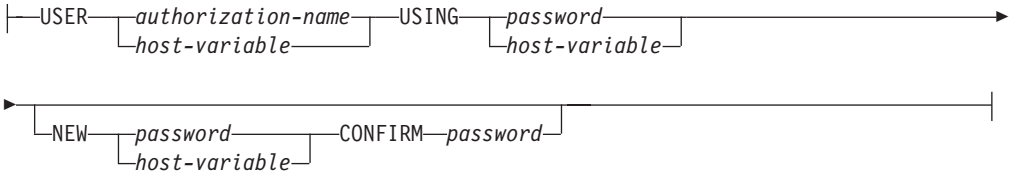
The authorization ID of the statement must be authorized to connect to the identified application server. Depending on the authentication setting for the database, the authorization check may be performed by either the client or the server. For a partitioned database, the user and group definitions must be identical across partitions.

Syntax:

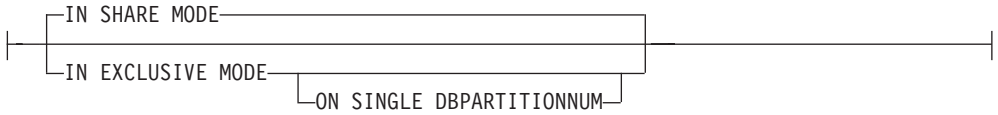
The selection between Type 1 and Type 2 is determined by precompiler options. For an overview of these options, see "Distributed relational databases".



authorization:



lock-block:



Notes:

1 This form is only valid if implicit connect is enabled.

Description:

TO *server-name/host-variable*

The rules for coding the name of the server are the same as for Type 1.

If the SQLRULES(STD) option is in effect, the *server-name* must not identify an existing connection of the application process, otherwise an error (SQLSTATE 08002) is raised.

If the SQLRULES(DB2) option is in effect and the *server-name* identifies an existing connection of the application process, that connection is made current and the old connection is placed into the dormant state. That is, the effect of the CONNECT statement in this situation is the same as that of a SET CONNECTION statement.

For information about the specification of SQLRULES, see "Options that Govern Distributed Unit of Work Semantics".

Successful Connection

If the CONNECT TO statement is successful:

- A connection to the application server is either created (or made non-dormant) and placed into the current and held states.
- If the CONNECT TO is directed to a different server than the current server, then the current connection is placed into the dormant state.
- The CURRENT SERVER special register and the SQLCA are updated in the same way as for CONNECT (Type 1).

Unsuccessful Connection

If the CONNECT TO statement is unsuccessful:

- No matter what the reason for failure, the connection state of the application process and the states of its connections are unchanged.
- As with an unsuccessful Type 1 CONNECT, the SQLERRP field of the SQLCA is set to the name of the module at the application requester or server that detected the error.

CONNECT (with no operand)**, IN SHARE/EXCLUSIVE MODE, USER,** and **USING**

If a connection exists, Type 2 behaves like a Type 1. The authorization ID and database alias are placed in the SQLERRMC field of the SQLCA. If a connection does not exist, no attempt to make an implicit connection is made and the SQLERRP and SQLERRMC fields return a blank. (Applications can check if a current connection exists by checking these fields.)

A CONNECT with no operand that includes USER and USING can still connect an application process to a database using the DB2DBDFT environment variable. This method is equivalent to a Type 2 CONNECT RESET, but permits the use of a user ID and password.

RESET

Equivalent to an explicit connect to the default database if it is available. If a default database is not available, the connection state of the application process and the states of its connections are unchanged.

Availability of a default database is determined by installation options, environment variables, and authentication settings.

Rules:

- As outlined in "Options that Govern Distributed Unit of Work Semantics", a set of connection options governs the semantics of connection management. Default values are assigned to every preprocessed source file. An application can consist of multiple source files precompiled with different connection options.

  Unless a SET CLIENT command or API has been executed first, the connection options used when preprocessing the source file containing the first SQL statement executed at run time become the effective connection options.

  If a CONNECT statement from a source file preprocessed with different connection options is subsequently executed without the execution of any intervening SET CLIENT command or API, an error (SQLSTATE 08001) is returned. Note that once a SET CLIENT command or API has been executed, the connection options used when preprocessing all source files in the application are ignored.

  Example 1 in the "Examples" section of this statement illustrates these rules.
- Although the CONNECT TO statement can be used to establish or switch connections, CONNECT TO with the USER/USING clause will only be accepted when there is no current or dormant connection to the named server. The connection must be released before issuing a connection to the same server with the USER/USING clause, otherwise it will be rejected (SQLSTATE 51022). Release the connection by issuing a DISCONNECT statement or a RELEASE statement followed by a COMMIT statement.

Notes:

- Implicit connect is supported for the first SQL statement in an application with Type 2 connections. In order to execute SQL statements on the default database, first the CONNECT RESET or the CONNECT USER/USING statement must be used to establish the connection. The CONNECT statement with no operands

will display information about the current connection if there is one, but will not connect to the default database if there is no current connection.

- The *authorization-name* SYSTEM cannot be explicitly specified in the CONNECT statement. However, on Windows operating systems, local applications running under the Local System Account can implicitly connect to the database, such that the user ID is SYSTEM.
- When connecting to Windows Server explicitly, the *authorization-name* or user *host-variable* can be specified using the Microsoft Windows NT Security Account Manager (SAM)-compatible name; for example, 'Domain\User'.

Comparing Type 1 and Type 2 CONNECT Statements:

The semantics of the CONNECT statement are determined by the CONNECT precompiler option or the SET CLIENT API (see "Options that Govern Distributed Unit of Work Semantics"). CONNECT Type 1 or CONNECT Type 2 can be specified and the CONNECT statements in those programs are known as Type 1 and Type 2 CONNECT statements, respectively. Their semantics are described below:

Use of **CONNECT TO**:

Type 1	Type 2
Each unit of work can only establish connection to one application server.	Each unit of work can establish connection to multiple application servers.
The current unit of work must be committed or rolled back before allowing a connection to another application server.	The current unit of work need not be committed or rolled back before connecting to another application server.
The CONNECT statement establishes the current connection. Subsequent SQL requests are forwarded to this connection until changed by another CONNECT.	Same as Type 1 CONNECT if establishing the first connection. If switching to a dormant connection and SQLRULES is set to STD, then the SET CONNECTION statement must be used instead.
Connecting to the current connection is valid and does not change the current connection.	Same as Type 1 CONNECT if the SQLRULES precompiler option is set to DB2. If SQLRULES is set to STD, then the SET CONNECTION statement must be used instead.
Connecting to another application server disconnects the current connection. The new connection becomes the current connection. Only one connection is maintained in a unit of work.	Connecting to another application server puts the current connection into the *dormant state*. The new connection becomes the current connection. Multiple connections can be maintained in a unit of work. If the CONNECT is for an application server on a dormant connection, it becomes the current connection. Connecting to a dormant connection using CONNECT is only allowed if SQLRULES(DB2) was specified. If SQLRULES(STD) was specified, then the SET CONNECTION statement must be used instead.
SET CONNECTION statement is supported for Type 1 connections, but the only valid target is the current connection.	SET CONNECTION statement is supported for Type 2 connections to change the state of a connection from dormant to current.

Use of **CONNECT...USER...USING**:

Type 1	Type 2
Connecting with the USER...USING clauses disconnects the current connection and establishes a new connection with the given authorization name and password.	Connecting with the USER/USING clause will only be accepted when there is no current or dormant connection to the same named server.

Use of **Implicit CONNECT, CONNECT RESET**, and **Disconnecting**:

Type 1	Type 2
CONNECT RESET can be used to disconnect the current connection.	CONNECT RESET is equivalent to connecting to the default application server explicitly if one has been defined in the system. Connections can be disconnected by the application at a successful COMMIT. Prior to the commit, use the RELEASE statement to mark a connection as release-pending. All such connections will be disconnected at the next COMMIT. An alternative is to use the precompiler options DISCONNECT(EXPLICIT), DISCONNECT(CONDITIONAL), DISCONNECT(AUTOMATIC), or the DISCONNECT statement instead of the RELEASE statement.
After using CONNECT RESET to disconnect the current connection, if the next SQL statement is not a CONNECT statement, then it will perform an implicit connect to the default application server if one has been defined in the system.	CONNECT RESET is equivalent to an explicit connect to the default application server if one has been defined in the system.
It is an error to issue consecutive CONNECT RESETs.	It is an error to issue consecutive CONNECT RESETs ONLY if SQLRULES(STD) was specified because this option disallows the use of CONNECT to existing connection.
CONNECT RESET also implicitly commits the current unit of work.	CONNECT RESET does not commit the current unit of work.
If an existing connection is disconnected by the system for whatever reasons, then subsequent non-CONNECT SQL statements to this database will receive an SQLSTATE of 08003.	If an existing connection is disconnected by the system, COMMIT, ROLLBACK, and SET CONNECTION statements are still permitted.
The unit of work will be implicitly committed when the application process terminates successfully.	Same as Type 1.
All connections (only one) are disconnected when the application process terminates.	All connections (current, dormant, and those marked for release pending) are disconnected when the application process terminates.

CONNECT Failures:

Type 1	Type 2
Regardless of whether there is a current connection when a CONNECT fails (with an error other than server-name not defined in the local directory), the application process is placed in the unconnected state. Subsequent non-CONNECT statements receive an SQLSTATE of 08003.	If there is a current connection when a CONNECT fails, the current connection is unaffected. If there was no current connection when the CONNECT fails, then the program is then in an unconnected state. Subsequent non-CONNECT statements receive an SQLSTATE of 08003.

Examples:

Example 1:

This example illustrates the use of multiple source programs (shown in the boxes), some preprocessed with different connection options (shown above the code), and one of which contains a SET CLIENT API call.

PGM1: CONNECT(2) SQLRULES(DB2) DISCONNECT(CONDITIONAL)

```
...
exec sql CONNECT TO OTTAWA;
exec sql SELECT col1 INTO :hv1
FROM tbl1;
...
```

PGM2: CONNECT(2) SQLRULES(STD) DISCONNECT(AUTOMATIC)

```
...
exec sql CONNECT TO QUEBEC;
exec sql SELECT col1 INTO :hv1
FROM tbl2;
...
```

PGM3: CONNECT(2) SQLRULES(STD) DISCONNECT(EXPLICIT)

```
...
SET CLIENT CONNECT 2  SQLRULES DB2  DISCONNECT EXPLICIT [1]
exec sql CONNECT TO LONDON;
exec sql SELECT col1 INTO :hv1
FROM tbl3;
...
```

1 Note: not the actual syntax of the SET CLIENT API

PGM4: CONNECT(2) SQLRULES(DB2) DISCONNECT(CONDITIONAL)

```
...
exec sql CONNECT TO REGINA;
exec sql SELECT col1 INTO :hv1
FROM tbl4;
...
```

If the application executes PGM1 then PGM2:

- connect to OTTAWA runs: connect=2, sqlrules=DB2, disconnect=CONDITIONAL
- connect to QUEBEC fails with SQLSTATE 08001 because both SQLRULES and DISCONNECT are different.

If the application executes PGM1 then PGM3:

- connect to OTTAWA runs: connect=2, sqlrules=DB2, disconnect=CONDITIONAL
- connect to LONDON runs: connect=2, sqlrules=DB2, disconnect=EXPLICIT

This is OK because the SET CLIENT API is run before the second CONNECT statement.

If the application executes PGM1 then PGM4:

- connect to OTTAWA runs: connect=2, sqlrules=DB2, disconnect=CONDITIONAL
- connect to REGINA runs: connect=2, sqlrules=DB2, disconnect=CONDITIONAL

This is OK because the preprocessor options for PGM1 are the same as those for PGM4.

Example 2:

This example shows the interrelationships of the CONNECT (Type 2), SET CONNECTION, RELEASE, and DISCONNECT statements. S0, S1, S2, and S3 represent four servers.

Sequence	Statement	Current Server	Dormant Connections	Release Pending
0	No statement	None	None	None
1	SELECT * FROM TBLA	S0 (default)	None	None
2	CONNECT TO S1	S1	S0	None
	SELECT * FROM TBLB	S1	S0	None
3	CONNECT TO S2	S2	S0, S1	None
	UPDATE TBLC SET ...	S2	S0, S1	None
4	CONNECT TO S3	S3	S0, S1, S2	None
	SELECT * FROM TBLD	S3	S0, S1, S2	None
5	SET CONNECTION S2	S2	S0, S1, S3	None
6	RELEASE S3	S2	S0, S1	S3
7	COMMIT	S2	S0, S1	None
8	SELECT * FROM TBLE	S2	S0, S1	None
9	DISCONNECT S1	S2	S0	None
	SELECT * FROM TBLF	S2	S0	None

Related concepts:

- "Distributed relational databases" in the *SQL Reference, Volume 1*

Related reference:

- "CONNECT (Type 1)" on page 887

Related samples:

- "dbmcon.sqc -- How to use multiple databases (C)"
- "dbmcon.sqC -- How to use multiple databases (C++)"

ROLLBACK

The ROLLBACK statement is used to back out of the database changes that were made within a unit of work or a savepoint.

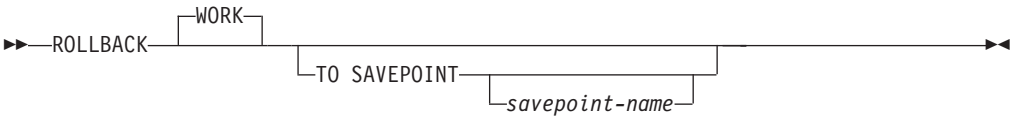
Invocation:

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared.

Authorization:

None required.

Syntax:



Description:
The unit of work in which the ROLLBACK statement is executed is terminated and a new unit of work is initiated. All changes made to the database during the unit of work are backed out.

The following statements, however, are not under transaction control, and changes made by them are independent of the ROLLBACK statement:

- SET CONNECTION
- SET CURRENT DEFAULT TRANSFORM GROUP
- SET CURRENT DEGREE
- SET CURRENT EXPLAIN MODE
- SET CURRENT EXPLAIN SNAPSHOT
- SET CURRENT LOCK TIMEOUT
- SET CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION
- SET CURRENT PACKAGESET
- SET CURRENT QUERY OPTIMIZATION
- SET CURRENT REFRESH AGE
- SET ENCRYPTION PASSWORD
- SET EVENT MONITOR STATE
- SET PASSTHRU

  Note: Although the SET PASSTHRU statement is not under transaction control, the passthru session initiated by the statement is under transaction control.
- SET PATH
- SET SCHEMA
- SET SERVER OPTION

The generation of sequence and identity values is not under transaction control. Values generated and consumed by the *nextval-expression* or by inserting rows into a table that has an identity column are independent of issuing the ROLLBACK statement. Also, issuing the ROLLBACK statement does not affect the value returned by the *prevval-expression*, nor the IDENTITY_VAL_LOCAL function.

TO SAVEPOINT
Specifies that a partial rollback (ROLLBACK TO SAVEPOINT) is to be performed. If no savepoint is active in the current savepoint level (see the "Rules" section in the description of the SAVEPOINT statement), an error is returned (SQLSTATE 3B502). After a successful rollback, the savepoint continues to exist, but any nested savepoints are released and no longer exist. The nested savepoints, if any, are considered to have been rolled back and then released as part of the rollback to the current savepoint. If a *savepoint-name* is not provided, rollback occurs to the most recently set savepoint within the current savepoint level.

If this clause is omitted, the ROLLBACK statement rolls back the entire transaction. Furthermore, savepoints within the transaction are released.

savepoint-name
Specifies the savepoint that is to be used in the rollback operation. The specified *savepoint-name* cannot begin with 'SYS' (SQLSTATE 42939). After a successful rollback operation, the named savepoint continues to exist. If the savepoint name does not exist, an error (SQLSTATE 3B001) is returned. Data and schema changes made since the savepoint was set are undone.

Notes:

- All locks held are released on a ROLLBACK of the unit of work. All open cursors are closed. All LOB locators are freed.
- Executing a ROLLBACK statement does not affect either the SET statements that change special register values or the RELEASE statement.
- If the program terminates abnormally, the unit of work is implicitly rolled back.
- Statement caching is affected by the rollback operation.
- The impact on cursors resulting from a ROLLBACK TO SAVEPOINT depends on the statements within the savepoint
  - If the savepoint contains DDL on which a cursor is dependent, the cursor is marked invalid. Attempts to use such a cursor results in an error (SQLSTATE 57007).
  - Otherwise:
    - If the cursor is referenced in the savepoint, the cursor remains open and is positioned before the next logical row of the result table. (A FETCH must be performed before a positioned UPDATE or DELETE statement is issued.)
    - Otherwise, the cursor is not affected by the ROLLBACK TO SAVEPOINT (it remains open and positioned).
- Dynamically prepared statement names are still valid, although the statement may be implicitly prepared again, as a result of DDL operations that are rolled back within the savepoint.
- A ROLLBACK TO SAVEPOINT operation will drop any declared temporary tables named within the savepoint. If a declared temporary table is modified within the savepoint, then all rows in the table are deleted.
- All locks are retained after a ROLLBACK TO SAVEPOINT statement.
- All LOB locators are preserved following a ROLLBACK TO SAVEPOINT operation.

Example:

Delete the alterations made since the last commit point or rollback.

```
ROLLBACK WORK
```

Related reference:

- "EXECUTE statement" in the *SQL Reference, Volume 2*
- "SAVEPOINT statement" in the *SQL Reference, Volume 2*

Related samples:

- "delet.sqb -- How to delete table data (MF COBOL)"
- "spclient.sqc -- Call various stored procedures (C)"
- "tut_use.sqc -- How to modify a database (C)"
- "spclient.sqC -- Call various stored procedures (C++)"
- "tut_use.sqC -- How to modify a database (C++)"

SELECT

The SELECT statement is a form of query. It can be embedded in an application program or issued interactively.

Related reference:

- "Subselect" on page 904
- "Select-statement" in the *SQL Reference, Volume 1*

Related samples:

- "dynamic.sqb -- How to update table data with cursor dynamically (MF COBOL)"
- "static.sqb -- Get table data using static SQL statement (MF COBOL)"
- "tbread.c -- How to read data from tables"
- "tut_read.c -- How to read data from tables"
- "tbread.sqc -- How to read tables (C)"
- "tut_read.sqc -- How to read tables (C)"
- "tbread.sqC -- How to read tables (C++)"
- "tut_read.sqC -- How to read tables (C++)"
- "TbRead.java -- How to read table data (JDBC)"
- "TutRead.java -- Read data in a table (JDBC)"
- "TbRead.sqlj -- How to read table data (SQLj)"
- "TutRead.sqlj -- Read data in a table (SQLj)"

SET SCHEMA

The SET SCHEMA statement changes the value of the CURRENT SCHEMA special register. It is not under transaction control. If the package is bound with the DYNAMICRULES BIND option, this statement does not affect the qualifier used for unqualified database object references.

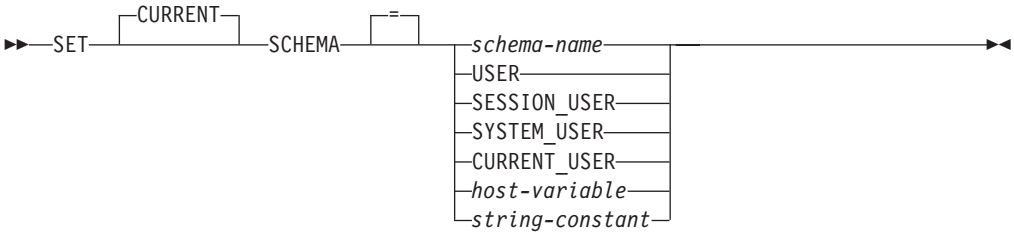
Invocation:

The statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared.

Authorization:

No authorization is required to execute this statement.

Syntax:



Description:

schema-name
This one-part name identifies a schema that exists at the application server. The length must not exceed 30 bytes (SQLSTATE 42815). No validation that the schema exists is made at the time that the schema is set. If a *schema-name* is misspelled, it will not be caught, and it could affect the way subsequent SQL operates.

USER
The value in the USER special register.

SESSION_USER
The value in the SESSION_USER special register.

SYSTEM_USER
The value in the SYSTEM_USER special register.

CURRENT_USER
The value in the CURRENT_USER special register.

host-variable
A variable of type CHAR or VARCHAR. The length of the contents of the *host-variable* must not exceed 30 (SQLSTATE 42815). It cannot be set to null. If *host-variable* has an associated indicator variable, the value of that indicator variable must not indicate a null value (SQLSTATE 42815).

The characters of the *host-variable* must be left justified. When specifying the *schema-name* with a *host-variable*, all characters must be specified in the exact case intended as there is no conversion to uppercase characters.

string-constant
A character string constant with a maximum length of 30.

Rules:

- If the value specified does not conform to the rules for a *schema-name*, an error (SQLSTATE 3F000) is raised.
- The value of the CURRENT SCHEMA special register is used as the schema name in all dynamic SQL statements, with the exception of the CREATE SCHEMA statement, where an unqualified reference to a database object exists.
- The QUALIFIER bind option specifies the schema name for use as the qualifier for unqualified database object names in static SQL statements.

Notes:

- The initial value of the CURRENT SCHEMA special register is equivalent to USER.
- Setting the CURRENT SCHEMA special register does not effect the CURRENT PATH special register. Hence, the CURRENT SCHEMA will not be included in the SQL path and functions, procedures and user-defined type resolution may not find these objects. To include the current schema value in the SQL path, whenever the SET SCHEMA statement is issued, also issue the SET PATH statement including the schema name from the SET SCHEMA statement.
- CURRENT SQLID is accepted as a synonym for CURRENT SCHEMA and the effect of a SET CURRENT SQLID statement will be identical to that of a SET CURRENT SCHEMA statement. No other effects, such as statement authorization changes, will occur.

Examples:

Example 1: The following statement sets the CURRENT SCHEMA special register.

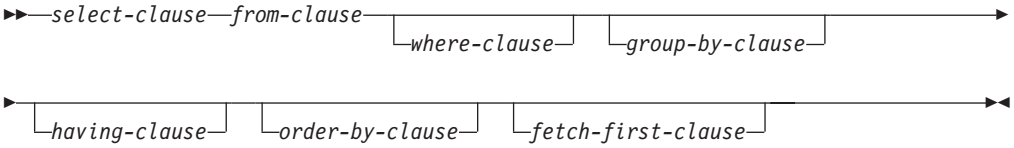
```
SET SCHEMA RICK
```

Example 2: The following example retrieves the current value of the CURRENT SCHEMA special register into the host variable called CURSCHEMA.

```
EXEC SQL VALUES (CURRENT SCHEMA) INTO :CURSCHEMA;
```

The value would be RICK, set by the previous example.

Subselect



The *subselect* is a component of the fullselect.

A subselect specifies a result table derived from the tables, views or nicknames identified in the FROM clause. The derivation can be described as a sequence of operations in which the result of each operation is input for the next. (This is only a way of describing the subselect. The method used to perform the derivation may be quite different from this description.)

The clauses of the subselect are processed in the following sequence:

1. FROM clause
2. WHERE clause
3. GROUP BY clause
4. HAVING clause
5. SELECT clause
6. ORDER BY clause
7. FETCH FIRST clause

A subselect that contains an ORDER BY or FETCH FIRST clause cannot be specified:

- In the outermost fullselect of a view.
- In a materialized query table.
- Unless the subselect is enclosed in parenthesis.

For example, the following is not valid (SQLSTATE 428FJ):

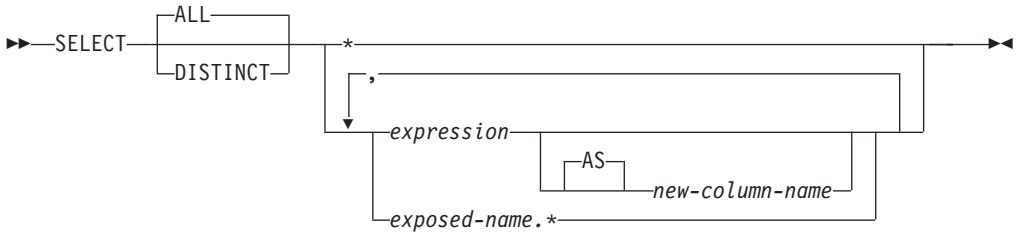
```
SELECT * FROM T1
   ORDER BY C1
UNION
SELECT * FROM T2
   ORDER BY C1
```

The following example *is* valid:

```
(SELECT * FROM T1
   ORDER BY C1)
UNION
(SELECT * FROM T2
   ORDER BY C1)
```

Note: An ORDER BY clause in a subselect does not affect the order of the rows returned by a query. An ORDER BY clause only affects the order of the rows returned if it is specified in the outermost fullselect.

select-clause



The SELECT clause specifies the columns of the final result table. The column values are produced by the application of the *select list* to R. The select list is the names or expressions specified in the SELECT clause, and R is the result of the previous operation of the subselect. For example, if the only clauses specified are SELECT, FROM, and WHERE, R is the result of that WHERE clause.

ALL
Retains all rows of the final result table, and does not eliminate redundant duplicates. This is the default.

DISTINCT
Eliminates all but one of each set of duplicate rows of the final result table. If DISTINCT is used, no string column of the result table can be a LONG VARCHAR, LONG VARGRAPHIC, DATALINK, LOB type, distinct type on any of these types, or structured type. DISTINCT may be used more than once in a subselect. This includes SELECT DISTINCT, the use of DISTINCT in a column function of the select list or HAVING clause, and subqueries of the subselect.

Two rows are duplicates of one another only if each value in the first is equal to the corresponding value of the second. For determining duplicates, two null values are considered equal.

Select list notation:

* Represents a list of names that identify the columns of table R. The first name in the list identifies the first column of R, the second name identifies the second column of R, and so on.

The list of names is established when the program containing the SELECT clause is bound. Hence * (the asterisk) does not identify any columns that have been added to a table after the statement containing the table reference has been bound.

expression
Specifies the values of a result column. Can be any expression that is a valid SQL language element, but commonly includes column names. Each column name used in the select list must unambiguously identify a column of R.

new-column-name or **AS** *new-column-name*
Names or renames the result column. The name must not be qualified and does not have to be unique. Subsequent usage of column-name is limited as follows:

- A new-column-name specified in the AS clause can be used in the order-by-clause, provided the name is unique.
- A new-column-name specified in the AS clause of the select list cannot be used in any other clause within the subselect (where-clause, group-by-clause or having-clause).
- A new-column-name specified in the AS clause cannot be used in the update-clause.
- A new-column-name specified in the AS clause is known outside the fullselect of nested table expressions, common table expressions and CREATE VIEW.

*name.**
Represents the list of names that identify the columns of the result table identified by *exposed-name*. The *exposed-name* may be a table name, view name, nickname, or correlation name, and must designate a table, view or nickname named in the FROM clause. The first name in the list identifies the first column of the table, view or nickname, the second name in the list identifies the second column of the table, view or nickname, and so on.

The list of names is established when the statement containing the SELECT clause is bound. Therefore, * does not identify any columns that have been added to a table after the statement has been bound.

The number of columns in the result of SELECT is the same as the number of expressions in the operational form of the select list (that is, the list established when the statement is prepared), and cannot exceed 500 for a 4K page size or 1012 for an 8K, 16K, or 32K page size.

Limitations on string columns

For limitations on the select list, see “Restrictions Using Varying-Length Character Strings”.

Applying the select list

Some of the results of applying the select list to R depend on whether or not GROUP BY or HAVING is used. The results are described in two separate lists:

If GROUP BY or HAVING is used:

- An expression *X* (not a column function) used in the select list must have a GROUP BY clause with:
  - a *grouping-expression* in which each column-name unambiguously identifies a column of R (see "group-by-clause" on page 918) or
  - each column of R referenced in *X* as a separate *grouping-expression*.
- The select list is applied to each group of R, and the result contains as many rows as there are groups in R. When the select list is applied to a group of R, that group is the source of the arguments of the column functions in the select list.

If neither GROUP BY nor HAVING is used:

- Either the select list must not include any column functions, or each *column-name* in the select list must be specified within a column function or must be a correlated column reference.
- If the select does not include column functions, then the select list is applied to each row of R and the result contains as many rows as there are rows in R.
- If the select list is a list of column functions, then R is the source of the arguments of the functions and the result of applying the select list is one row.

In either case the *n*th column of the result contains the values specified by applying the *n*th expression in the operational form of the select list.

Null attributes of result columns: Result columns do not allow null values if they are derived from:

- A column that does not allow null values
- A constant
- The COUNT or COUNT_BIG function
- A host variable that does not have an indicator variable
- A scalar function or expression that does not include an operand that allows nulls.

Result columns allow null values if they are derived from:

- Any column function except COUNT or COUNT_BIG
- A column that allows null values
- A scalar function or expression that includes an operand that allows nulls
- A NULLIF function with arguments containing equal values.
- A host variable that has an indicator variable.
- A result of a set operation if at least one of the corresponding items in the select list is nullable.
- An arithmetic expression or view column that is derived from an arithmetic expression and the database is configured with DFT_SQLMATHWARN set to yes
- A dereference operation.

Names of result columns:

- If the AS clause is specified, the name of the result column is the name specified on the AS clause.
- If the AS clause is not specified and the result column is derived from a column, then the result column name is the unqualified name of that column.

- If the AS clause is not specified and the result column is derived using a dereference operation, then the result column name is the unqualified name of the target column of the dereference operation.
- All other result column names are unnamed. The system assigns temporary numbers (as character strings) to these columns.

Data types of result columns: Each column of the result of SELECT acquires a data type from the expression from which it is derived.

When the expression is ...	The data type of the result column is ...
the name of any numeric column	the same as the data type of the column, with the same precision and scale for DECIMAL columns.
an integer constant	INTEGER.
a decimal constant	DECIMAL, with the precision and scale of the constant.
a floating-point constant	DOUBLE.
the name of any numeric variable	the same as the data type of the variable, with the same precision and scale for DECIMAL variables.
a hexadecimal constant representing n bytes	VARCHAR(n); the code page is the database code page.
the name of any string column	the same as the data type of the column, with the same length attribute.
the name of any string variable	the same as the data type of the variable, with the same length attribute; if the data type of the variable is not identical to an SQL data type (for example, a NUL-terminated string in C), the result column is a varying-length string.
a character string constant of length *n*	VARCHAR(*n*).
a graphic string constant of length *n*	VARGRAPHIC(*n*).
the name of a datetime column	the same as the data type of the column.
the name of a user-defined type column	the same as the data type of the column.
the name of a reference type column	the same as the data type of the column.

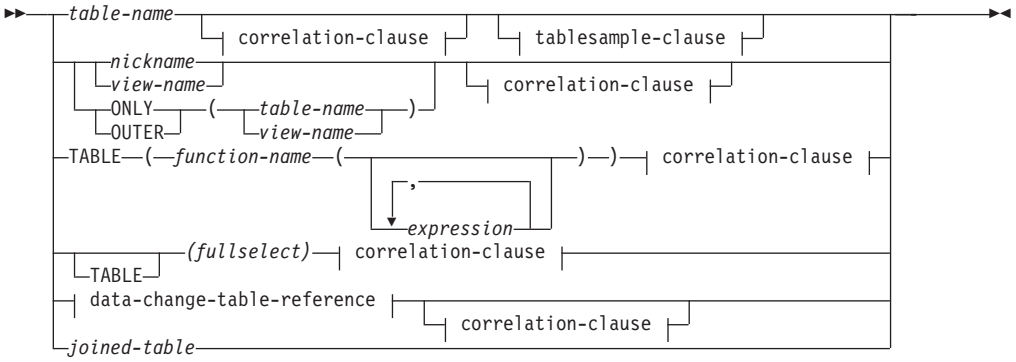
from-clause

The FROM clause specifies an intermediate result table.

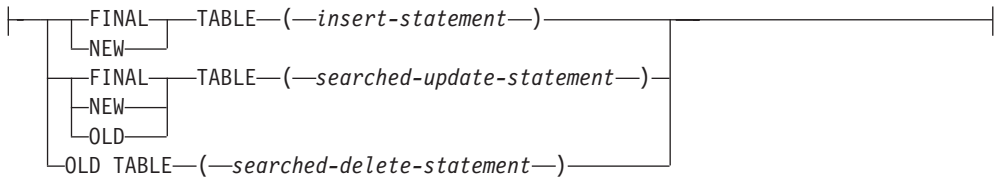
If one table-reference is specified, the intermediate result table is simply the result of that table-reference. If more than one table-reference is specified, the intermediate result table consists of all possible combinations of the rows of the specified table-references (the Cartesian product). Each row of the result is a row from the first table-reference concatenated with a row from the second table-reference, concatenated in turn with a row from the third, and so on. The

number of rows in the result is the product of the number of rows in all the individual table-references. For a description of *table-reference*, see "table-reference."

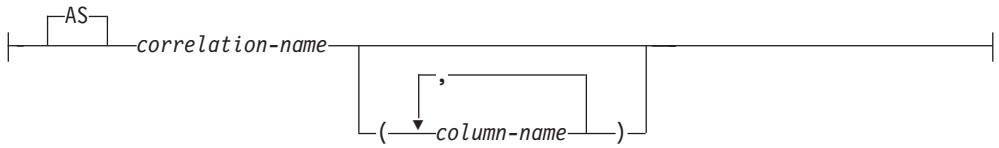
table-reference



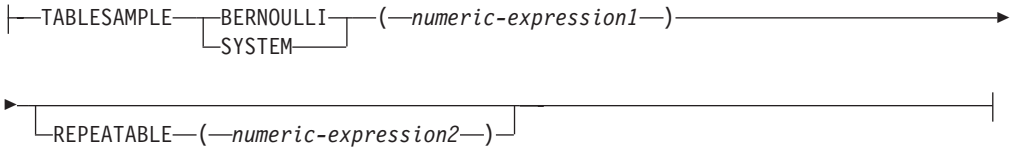
data-change-table-reference:



correlation-clause:



tablesample-clause:



Each *table-name*, *view-name* or *nickname* specified as a table-reference must identify an existing table, view or nickname at the application server or the *table-name* of a common table expression defined preceding the fullselect containing the table-reference. If the *table-name* references a typed table, the name denotes the UNION ALL of the table with all its subtables, with only the columns of the *table-name*. Similarly, if the *view-name* references a typed view, the name denotes the UNION ALL of the view with all its subviews, with only the columns of the *view-name*.

The use of ONLY(*table-name*) or ONLY(*view-name*) means that the rows of the proper subtables or subviews are not included. If the *table-name* used with ONLY does not have subtables, then ONLY(*table-name*) is equivalent to specifying *table-name*. If the *view-name* used with ONLY does not have subviews, then ONLY(*view-name*) is equivalent to specifying *view-name*.

The use of OUTER(*table-name*) or OUTER(*view-name*) represents a virtual table. If the *table-name* or *view-name* used with OUTER does not have subtables or subviews, then specifying OUTER is equivalent to not specifying OUTER. OUTER(*table-name*) is derived from *table-name* as follows:

- The columns include the columns of *table-name* followed by the additional columns introduced by each of its subtables (if any). The additional columns are added on the right, traversing the subtable hierarchy in depth-first order. Subtables that have a common parent are traversed in creation order of their types.
- The rows include all the rows of *table-name* and all the rows of its subtables. Null values are returned for columns that are not in the subtable for the row.

The previous points also apply to OUTER(*view-name*), substituting *view-name* for *table-name* and subview for subtable.

The use of ONLY or OUTER requires the SELECT privilege on every subtable of *table-name* or subview of *view-name*.

Each *function-name* together with the types of its arguments, specified as a table reference must resolve to an existing table function at the application server.

A fullselect in parentheses followed by a correlation name is called a *nested table expression*.

A *joined-table* specifies an intermediate result set that is the result of one or more join operations. For more information, see "joined-table" on page 916.

The exposed names of all table references should be unique. An exposed name is:

- A *correlation-name*,
- A *table-name* that is not followed by a *correlation-name*,
- A *view-name* that is not followed by a *correlation-name*,
- A *nickname* that is not followed by a *correlation-name*,
- An *alias-name* that is not followed by a *correlation-name*.

Each *correlation-name* is defined as a designator of the immediately preceding *table-name*, *view-name*, *nickname*, *function-name* reference or nested table expression. Any qualified reference to a column for a table, view, table function or nested table expression must use the exposed name. If the same table name, view or nickname name is specified twice, at least one specification should be followed by a *correlation-name*. The *correlation-name* is used to qualify references to the columns of the table, view or nickname. When a *correlation-name* is specified, *column-name*s can also be specified to give names to the columns of the *table-name*, *view-name*, *nickname*, *function-name* reference or nested table expression.

In general, table functions and nested table expressions can be specified on any from-clause. Columns from the table functions and nested table expressions can be referenced in the select list and in the rest of the subselect using the correlation

name which must be specified. The scope of this correlation name is the same as correlation names for other table, view or nickname in the FROM clause. A nested table expression can be used:

- In place of a view to avoid creating the view (when general use of the view is not required)
- When the desired result table is based on host variables

An expression in the select list of a nested table expression that is referenced within, or is the target of, a data change statement within a fullselect is only valid when it does not include:

- A function that reads or modifies SQL data
- A function that is non-deterministic
- A function that has external action
- An OLAP function

If a view is referenced directly in, or as the target of a nested table expression in a data change statement within a FROM clause, the view must either be symmetric (have WITH CHECK OPTION specified) or satisfy the restriction for a WITH CHECK OPTION view.

If the target of a data change statement within a FROM clause is a nested table expression, the modified rows are not requalified, WHERE clause predicates are not re-evaluated, and ORDER BY or FETCH FIRST operations are not redone.

The optional tablesample-clause can be used to obtain a random subset (a sample) of the rows from the specified *table-name*, rather than the entire contents of that *table-name*, for this query. This sampling is in addition to any predicates that are specified in the *where-clause*. Unless the optional REPEATABLE clause is specified, each execution of the query will usually yield a different sample, except in degenerate cases where the table is so small relative to the sample size that any sample must return the same rows. The size of the sample is controlled by the *numeric-expression1* in parentheses, representing an approximate percentage (P) of the table to be returned. The method by which the sample is obtained is specified after the TABLESAMPLE keyword, and can be either BERNOULLI or SYSTEM. For both methods, the exact number of rows in the sample may be different for each execution of the query, but on average should be approximately P percent of the table, before any predicates further reduce the number of rows.

The *table-name* must be a stored table. It can be a materialized query table (MQT) name, but not a subselect or table expression for which an MQT has been defined, because there is no guarantee that the database manager will route to the MQT for that subselect.

Semantically, sampling of a table occurs before any other query processing, such as applying predicates or performing joins. Repeated accesses of a sampled table within a single execution of a query (such as in a nested-loop join or a correlated subquery) will return the same sample. More than one table may be sampled in a query.

BERNOULLI sampling considers each row individually. It includes each row in the sample with probability P/100 (where P is the value of *numeric-expression1*), and excludes each row with probability 1 - P/100, independently of the other rows. So

if the *numeric-expression1* evaluated to the value 10, representing a ten percent sample, each row would be included with probability 0.1, and excluded with probability 0.9.

SYSTEM sampling permits the database manager to determine the most efficient manner in which to perform the sampling. In most cases, SYSTEM sampling applied to a *table-name* means that each page of *table-name* is included in the sample with probability P/100, and excluded with probability 1 - P/100. All rows on each page that is included qualify for the sample. SYSTEM sampling of a *table-name* generally executes much faster than BERNOULLI sampling, because fewer data pages need to be retrieved; however, it can often yield less accurate estimates for aggregate functions (SUM(SALES), for example), especially if the rows of *table-name* are clustered on any columns referenced in that query. The optimizer may in certain circumstances decide that it is more efficient to perform SYSTEM sampling as if it were BERNOULLI sampling, for example when a predicate on *table-name* can be applied by an index and is much more selective than the sampling rate P.

The *numeric-expression1* specifies the size of the sample to be obtained from *table-name*, expressed as a percentage. It must be a constant numeric expression that cannot contain columns, parameter markers, or host variables. The expression must evaluate to a positive number that is less than or equal to 100, but can be between 1 and 0. For example, a value of 0.01 represents one one-hundredth of a percent, meaning that 1 row in 10 000 would be sampled, on average. A *numeric-expression1* that evaluates to 100 is handled as if the tablesample-clause were not specified. If *numeric-expression1* evaluates to the null value, or to a value that is greater than 100 or less than 0, an error is returned (SQLSTATE 2202H).

It is sometimes desirable for sampling to be repeatable from one execution of the query to the next; for example, during regression testing or query ″debugging″. This can be accomplished by specifying the REPEATABLE clause. The REPEATABLE clause requires the specification of a *numeric-expression2* in parentheses, which serves the same role as the seed in a random number generator. Adding the REPEATABLE clause to the tablesample-clause of any *table-name* ensures that repeated executions of that query (using the same value for *numeric-expression2*) return the same sample, assuming, of course, that the data itself has not been updated, reorganized, or repartitioned. To guarantee that the same sample of *table-name* is used across multiple queries, use of a global temporary table is recommended. Alternatively, the multiple queries could be combined into one query, with multiple references to a sample that is defined using the WITH clause.

Following are some examples:

Example 1: Request a 10% Bernoulli sample of the Sales table for auditing purposes.

```
SELECT * FROM Sales
  TABLESAMPLE BERNOULLI(10)
```

Example 2: Compute the total sales revenue in the Northeast region for each product category, using a random 1% SYSTEM sample of the Sales table. The semantics of SUM are for the sample itself, so to extrapolate the sales to the entire Sales table, the query must divide that SUM by the sampling rate (0.01).

```
SELECT SUM(Sales.Revenue) / (0.01)
  FROM Sales TABLESAMPLE SYSTEM(1)
  WHERE Sales.RegionName = 'Northeast'
  GROUP BY Sales.ProductCategory
```

Example 3: Using the REPEATABLE clause, modify the previous query to ensure that the same (yet random) result is obtained each time the query is executed. (The value of the constant enclosed by parentheses is arbitrary.)

```
SELECT SUM(Sales.Revenue) / (0.01)
  FROM Sales TABLESAMPLE SYSTEM(1) REPEATABLE(3578231)
  WHERE Sales.RegionName = 'Northeast'
  GROUP BY Sales.ProductCategory
```

Table function references

In general, a table function, together with its argument values, can be referenced in the FROM clause of a SELECT in exactly the same way as a table or view. There are, however, some special considerations which apply.

- Table Function Column Names

  Unless alternate column names are provided following the *correlation-name*, the column names for the table function are those specified in the RETURNS clause of the CREATE FUNCTION statement. This is analogous to the names of the columns of a table, which are defined in the CREATE TABLE statement.

- Table Function Resolution

  The arguments specified in a table function reference, together with the function name, are used by an algorithm called *function resolution* to determine the exact function to be used. This is no different from what happens with other functions (such as scalar functions) that are used in a statement.

- Table Function Arguments

  As with scalar function arguments, table function arguments can in general be any valid SQL expression. The following examples are valid syntax:

```
Example 1:  SELECT c1
            FROM TABLE( tf1('Zachary') ) AS z
            WHERE c2 = 'FLORIDA';

Example 2:  SELECT c1
            FROM TABLE( tf2 (:hostvar1, CURRENT DATE) ) AS z;

Example 3:  SELECT c1
            FROM t
            WHERE c2 IN
                     (SELECT c3 FROM
                      TABLE( tf5(t.c4) ) AS z  -- correlated reference
                     )                         -- to previous FROM clause
```

- Table Functions That Modify SQL Data

  Table functions that are specified with the MODIFIES SQL DATA option can only be used as the last table reference in a *select-statement*, *common-table-expression*, or RETURN statement that is a subselect, a SELECT INTO, or a *row-fullselect* in a SET statement. Only one table function is allowed in one FROM clause, and the table function arguments must be correlated to all other table references in the subselect (SQLSTATE 429BL). The following examples have valid syntax for a table function with the MODIFIES SQL DATA property:

```
Example 1:  SELECT c1
            FROM TABLE( tfmod('Jones') ) AS z

Example 2:  SELECT c1
            FROM t1, t2, TABLE( tfmod(t1.c1, t2.c1) ) AS z

Example 3:  SET var =
            (SELECT c1
            FROM TABLE( tfmod('Jones') ) AS z

Example 4:  RETURN SELECT c1
            FROM TABLE( tfmod('Jones') ) AS z
```

```
Example 5:  WITH v1(c1) AS
            (SELECT c1
            FROM TABLE( tfmod(:hostvar1) ) AS z)
            SELECT c1
            FROM v1, t1 WHERE v1.c1 = t1.c1
```

Correlated references in table-references

Correlated references can be used in nested table expressions or as arguments to table functions. The basic rule that applies for both these cases is that the correlated reference must be from a *table-reference* at a higher level in the hierarchy of subqueries. This hierarchy includes the table-references that have already been resolved in the left-to-right processing of the FROM clause. For nested table expressions, the TABLE keyword must appear before the fullselect. So the following examples are valid syntax:

```
Example 1:  SELECT t.c1, z.c5
            FROM t, TABLE( tf3(t.c2) ) AS z      -- t precedes tf3
            WHERE t.c3 = z.c4;                   -- in FROM, so t.c2
                                                 -- is known

Example 2:  SELECT t.c1, z.c5
            FROM t, TABLE( tf4(2 * t.c2) ) AS z  -- t precedes tf3
            WHERE t.c3 = z.c4;                   -- in FROM, so t.c2
                                                 -- is known

Example 3:  SELECT d.deptno, d.deptname,
                   empinfo.avgsal, empinfo.empcount
            FROM department d,
                 TABLE (SELECT AVG(e.salary) AS avgsal,
                               COUNT(*) AS empcount
                        FROM employee e           -- department precedes
                        WHERE e.workdept=d.deptno -- and TABLE is
                       ) AS empinfo;              -- specified, so
                                                  -- d.deptno is known
```

But the following examples are not valid:

```
Example 4:  SELECT t.c1, z.c5
            FROM TABLE( tf6(t.c2) ) AS z, t  -- cannot resolve t in t.c2!
            WHERE t.c3 = z.c4;               -- compare to Example 1 above.

Example 5:  SELECT a.c1, b.c5
            FROM TABLE( tf7a(b.c2) ) AS a, TABLE( tf7b(a.c6) ) AS b
            WHERE a.c3 = b.c4;               -- cannot resolve b in b.c2!

Example 6:  SELECT d.deptno, d.deptname,
                   empinfo.avgsal, empinfo.empcount
            FROM department d,
                 (SELECT AVG(e.salary) AS avgsal,
                               COUNT(*) AS empcount
                        FROM employee e           -- department precedes
                        WHERE e.workdept=d.deptno -- but TABLE is not
                       ) AS empinfo;              -- specified, so
                                                  -- d.deptno is unknown
```

Data change table references

A *data-change-table-reference* clause specifies an intermediate result table. This table is based on the rows that are directly changed by the searched UPDATE, searched DELETE, or INSERT statement that is included in the clause. A *data-change-table-reference* can be specified as the only *table-reference* in the FROM clause of the outer fullselect that is used in a *select-statement*, a SELECT INTO statement, or a common table expression. A *data-change-table-reference* can be specified as the only table reference in the only fullselect in a SET Variable

statement (SQLSTATE 428FL). The target table or view of the data change statement is considered to be a table or view that is referenced in the query; therefore, the authorization ID of the query must have SELECT privilege on that target table or view.

The target of the UPDATE, DELETE, or INSERT statement cannot be a temporary view defined in a common table expression (SQLSTATE 42807).

FINAL TABLE
Specifies that the rows of the intermediate result table represent the set of rows that are changed by the SQL data change statement as they appear at the completion of the data change statement. If there are AFTER triggers or referential constraints that result in further operations on the table that is the target of the SQL data change statement, an error is returned (SQLSTATE 57058, SQLSTATE 560C6). If the target of the SQL data change statement is a view that is defined with an INSTEAD OF trigger for the type of data change, an error is returned (SQLSTATE 428G3).

NEW TABLE
Specifies that the rows of the intermediate result table represent the set of rows that are changed by the SQL data change statement prior to the application of referential constraints and AFTER triggers. Data in the target table at the completion of the statement might not match the data in the intermediate result table because of additional processing for referential constraints and AFTER triggers.

OLD TABLE
Specifies that the rows of the intermediate result table represent the set of rows that are changed by the SQL data change statement as they existed prior to the application of the data change statement.

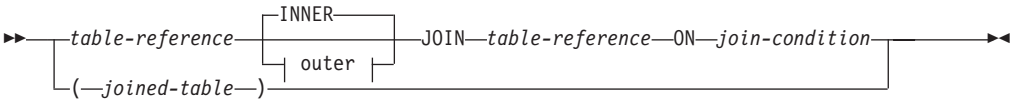
(searched-update-statement)
Specifies a searched UPDATE statement. A WHERE clause or a SET clause in the UPDATE statement cannot contain correlated references to columns outside of the UPDATE statement.

(searched-delete-statement)
Specifies a searched DELETE statement. A WHERE clause in the DELETE statement cannot contain correlated references to columns outside of the DELETE statement.

(insert-statement)
Specifies an INSERT statement. A fullselect in the INSERT statement cannot contain correlated references to columns outside of the fullselect of the INSERT statement.

The content of the intermediate result table for a *data-change-table-reference* is determined when the cursor opens. The intermediate result table contains all manipulated rows, including all the columns in the specified target table or view. All the columns of the target table or view for an SQL data change statement are accessible using the column names from the target table or view. If an INCLUDE clause was specified within a data change statement, the intermediate result table will contain these additional columns.

joined-table



outer:

A *joined table* specifies an intermediate result table that is the result of either an inner join or an outer join. The table is derived by applying one of the join operators: INNER, LEFT OUTER, RIGHT OUTER, or FULL OUTER to its operands.

Inner joins can be thought of as the cross product of the tables (combine each row of the left table with every row of the right table), keeping only the rows where the join condition is true. The result table may be missing rows from either or both of the joined tables. Outer joins include the inner join and preserve these missing rows. There are three types of outer joins:

- ***left outer join*** includes rows from the left table that were missing from the inner join.
- ***right outer join*** includes rows from the right table that were missing from the inner join.
- ***full outer join*** includes rows from both the left and right tables that were missing from the inner join.

If a join-operator is not specified, INNER is implicit. The order in which multiple joins are performed can affect the result. Joins can be nested within other joins. The order of processing for joins is generally from left to right, but based on the position of the required join-condition. Parentheses are recommended to make the order of nested joins more readable. For example:

```
TB1 LEFT JOIN TB2 ON TB1.C1=TB2.C1
  RIGHT JOIN TB3 LEFT JOIN TB4 ON TB3.C1=TB4.C1
    ON TB1.C1=TB3.C1
```

is the same as:

```
(TB1 LEFT JOIN TB2 ON TB1.C1=TB2.C1)
  RIGHT JOIN (TB3 LEFT JOIN TB4 ON TB3.C1=TB4.C1)
    ON TB1.C1=TB3.C1
```

A joined table can be used in any context in which any form of the SELECT statement is used. A view or a cursor is read-only if its SELECT statement includes a joined table.

A *join-condition* is a *search-condition* except that:

- it cannot contain any subqueries, scalar or otherwise
- it cannot include any dereference operations or the DEREF function where the reference value is other than the object identifier column.
- it cannot include an SQL function

- any column referenced in an expression of the *join-condition* must be a column of one of the operand tables of the associated join (in the scope of the same joined-table clause)
- any function referenced in an expression of the *join-condition* of a full outer join must be deterministic and have no external action.

An error occurs if the join condition does not comply with these rules (SQLSTATE 42972).

Column references are resolved using the rules for resolution of column name qualifiers. The same rules that apply to predicates apply to join conditions.

Join operations

A *join-condition* specifies pairings of T1 and T2, where T1 and T2 are the left and right operand tables of the JOIN operator of the *join-condition*. For all possible combinations of rows of T1 and T2, a row of T1 is paired with a row of T2 if the *join-condition* is true. When a row of T1 is joined with a row of T2, a row in the result consists of the values of that row of T1 concatenated with the values of that row of T2. The execution might involve the generation of a null row. The null row of a table consists of a null value for each column of the table, regardless of whether the columns allow null values.

The following summarizes the result of the join operations:

- The result of T1 INNER JOIN T2 consists of their paired rows where the join-condition is true.
- The result of T1 LEFT OUTER JOIN T2 consists of their paired rows where the join-condition is true and, for each unpaired row of T1, the concatenation of that row with the null row of T2. All columns derived from T2 allow null values.
- The result of T1 RIGHT OUTER JOIN T2 consists of their paired rows where the join-condition is true and, for each unpaired row of T2, the concatenation of that row with the null row of T1. All columns derived from T1 allow null values.
- The result of T1 FULL OUTER JOIN T2 consists of their paired rows and, for each unpaired row of T2, the concatenation of that row with the null row of T1 and, for each unpaired row of T1, the concatenation of that row with the null row of T2. All columns derived from T1 and T2 allow null values.

where-clause

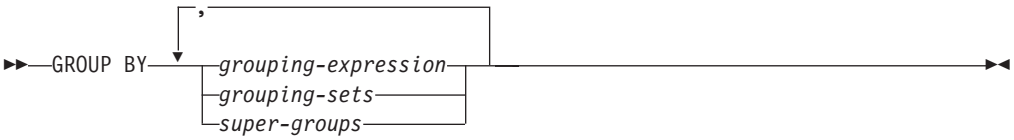
The WHERE clause specifies an intermediate result table that consists of those rows of R for which the *search-condition* is true. R is the result of the FROM clause of the subselect.

The *search-condition* must conform to the following rules:

- Each *column-name* must unambiguously identify a column of R or be a correlated reference. A *column-name* is a correlated reference if it identifies a column of a *table-reference* in an outer subselect.
- A column function must not be specified unless the WHERE clause is specified in a subquery of a HAVING clause and the argument of the function is a correlated reference to a group.

Any subquery in the *search-condition* is effectively executed for each row of R, and the results are used in the application of the *search-condition* to the given row of R. A subquery is actually executed for each row of R only if it includes a correlated reference. In fact, a subquery with no correlated references may be executed just once, whereas a subquery with a correlated reference may have to be executed once for each row.

group-by-clause



The GROUP BY clause specifies an intermediate result table that consists of a grouping of the rows of R. R is the result of the previous clause of the subselect.

In its simplest form, a GROUP BY clause contains a *grouping expression*. A grouping expression is an *expression* used in defining the grouping of R. Each *column name* included in grouping-expression must unambiguously identify a column of R (SQLSTATE 42702 or 42703). A grouping expression cannot include a scalar-fullselect (SQLSTATE 42822) or any function that is variant or has an external action (SQLSTATE 42845).

More complex forms of the GROUP BY clause include *grouping-sets* and *super-groups*. For a description of these forms, see "grouping-sets" on page 919 and "super-groups" on page 920, respectively.

The result of GROUP BY is a set of groups of rows. Each row in this result represents the set of rows for which the *grouping-expression* is equal. For grouping, all null values from a *grouping-expression* are considered equal.

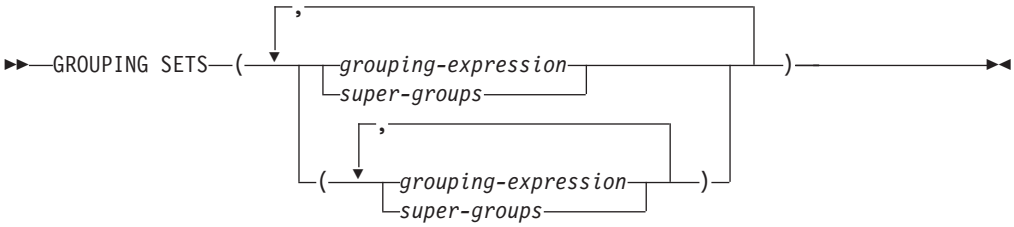
A *grouping-expression* can be used in a search condition in a HAVING clause, in an expression in a SELECT clause or in a *sort-key-expression* of an ORDER BY clause (see "order-by-clause" on page 924 for details). In each case, the reference specifies only one value for each group. For example, if the *grouping-expression* is *col1+col2*, then an allowed expression in the select list would be *col1+col2+3*. Associativity rules for expressions would disallow the similar expression, *3+col1+col2*, unless parentheses are used to ensure that the corresponding expression is evaluated in the same order. Thus, *3+(col1+col2)* would also be allowed in the select list. If the concatenation operator is used, the *grouping-expression* must be used exactly as the expression was specified in the select list.

If the *grouping-expression* contains varying-length strings with trailing blanks, the values in the group can differ in the number of trailing blanks and may not all have the same length. In that case, a reference to the *grouping-expression* still specifies only one value for each group, but the value for a group is chosen arbitrarily from the available set of values. Thus, the actual length of the result value is unpredictable.

As noted, there are some cases where the GROUP BY clause cannot refer directly to a column that is specified in the SELECT clause as an expression (scalar-fullselect, variant or external action functions). To group using such an expression, use a nested table expression or a common table expression to first

provide a result table with the expression as a column of the result. For an example using nested table expressions, see "Example A9" on page 929.

grouping-sets



A *grouping-sets* specification allows multiple grouping clauses to be specified in a single statement. This can be thought of as the union of two or more groups of rows into a single result set. It is logically equivalent to the union of multiple subselects with the group by clause in each subselect corresponding to one grouping set. A grouping set can be a single element or can be a list of elements delimited by parentheses, where an element is either a grouping-expression or a super-group. Using *grouping-sets* allows the groups to be computed with a single pass over the base table.

The *grouping-sets* specification allows either a simple *grouping-expression* to be used, or the more complex forms of *super-groups*. For a description of *super-groups*, see "super-groups" on page 920.

Note that grouping sets are the fundamental building blocks for GROUP BY operations. A simple GROUP BY with a single column can be considered a grouping set with one element. For example:

```
GROUP BY a
```

is the same as

```
GROUP BY GROUPING SETS((a))
```

and

```
GROUP BY a,b,c
```

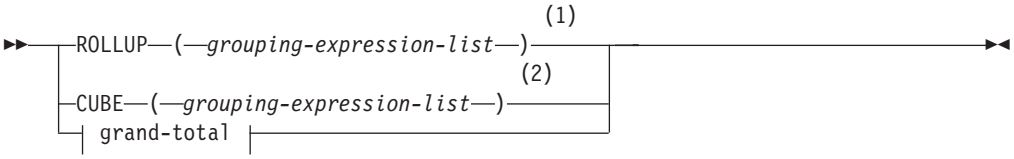
is the same as

```
GROUP BY GROUPING SETS((a,b,c))
```

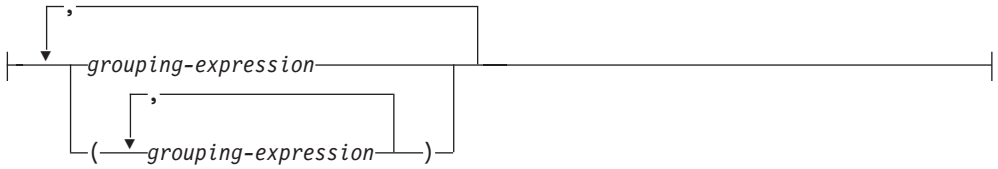
Non-aggregation columns from the select list of the subselect that are excluded from a grouping set will return a null for such columns for each row generated for that grouping set. This reflects the fact that aggregation was done without considering the values for those columns.

"Example C2" on page 933 through "Example C7" on page 936 illustrate the use of grouping sets.

super-groups



grouping-expression-list:



grand-total:

Notes:

1 Alternate specification when used alone in group-by-clause is: grouping-expression-list WITH ROLLUP.

2 Alternate specification when used alone in group-by-clause is: grouping-expression-list WITH CUBE.

ROLLUP (*grouping-expression-list* **)**

A *ROLLUP grouping* is an extension to the GROUP BY clause that produces a result set containing *sub-total* rows in addition to the "regular" grouped rows. *Sub-total* rows are "super-aggregate" rows that contain further aggregates whose values are derived by applying the same column functions that were used to obtain the grouped rows. These rows are called sub-total rows, because that is their most common use; however, any column function can be used for the aggregation. For instance, MAX and AVG are used in "Example C8" on page 938.

A ROLLUP grouping is a series of *grouping-sets*. The general specification of a ROLLUP with *n* elements

```
GROUP BY ROLLUP(C1,C2,...,Cn-1,Cn)
```

is equivalent to

```
GROUP BY GROUPING SETS((C1,C2,...,Cn-1,Cn)
                       (C1,C2,...,Cn-1)
                       ...
                       (C1,C2)
                       (C1)
                       () )
```

Note that the *n* elements of the ROLLUP translate to *n*+1 grouping sets. Note also that the order in which the *grouping-expressions* is specified is significant for ROLLUP. For example:

```
GROUP BY ROLLUP(a,b)
```

is equivalent to

```
GROUP BY GROUPING SETS((a,b)
                       (a)
                       () )
```

while

```
GROUP BY ROLLUP(b,a)
```

is the same as

```
GROUP BY GROUPING SETS((b,a)
                       (b)
                       () )
```

The ORDER BY clause is the only way to guarantee the order of the rows in the result set. "Example C3" on page 933 illustrates the use of ROLLUP.

CUBE (*grouping-expression-list*)

A *CUBE grouping* is an extension to the GROUP BY clause that produces a result set that contains all the rows of a ROLLUP aggregation and, in addition, contains "cross-tabulation" rows. *Cross-tabulation* rows are additional "super-aggregate" rows that are not part of an aggregation with sub-totals.

Like a ROLLUP, a CUBE grouping can also be thought of as a series of *grouping-sets*. In the case of a CUBE, all permutations of the cubed *grouping-expression-list* are computed along with the grand total. Therefore, the *n* elements of a CUBE translate to 2**n* (2 to the power *n*) *grouping-sets*. For instance, a specification of

```
GROUP BY CUBE(a,b,c)
```

is equivalent to

```
GROUP BY GROUPING SETS((a,b,c)
                       (a,b)
                       (a,c)
                       (b,c)
                       (a)
                       (b)
                       (c)
                       () )
```

Notice that the 3 elements of the CUBE translate to 8 grouping sets.

The order of specification of elements does not matter for CUBE. 'CUBE (DayOfYear, Sales_Person)' and 'CUBE (Sales_Person, DayOfYear)' yield the same result sets. The use of the word 'same' applies to content of the result set, not to its order. The ORDER BY clause is the only way to guarantee the order of the rows in the result set. "Example C4" on page 933 illustrates the use of CUBE.

grouping-expression-list

A *grouping-expression-list* is used within a CUBE or ROLLUP clause to define the number of elements in the CUBE or ROLLUP operation. This is controlled by using parentheses to delimit elements with multiple *grouping-expression*s.

The rules for a *grouping-expression* are described in "group-by-clause" on page 918. For example, suppose that a query is to return the total expenses for the ROLLUP of City within a Province but not within a County. However the clause:

```
GROUP BY ROLLUP(Province, County, City)
```

results in unwanted sub-total rows for the County. In the clause

```
GROUP BY ROLLUP(Province, (County, City))
```

the composite (County, City) forms one element in the ROLLUP and, therefore, a query that uses this clause will yield the desired result. In other words, the two element ROLLUP

```
GROUP BY ROLLUP(Province, (County, City))
```

generates

```
GROUP BY GROUPING SETS((Province, County, City)
                       (Province)
                       () )
```

while the 3 element ROLLUP would generate

```
GROUP BY GROUPING SETS((Province, County, City)
                       (Province, County)
                       (Province)
                       () )
```

"Example C2" on page 933 also utilizes composite column values.

grand-total
Both CUBE and ROLLUP return a row which is the overall (grand total) aggregation. This may be separately specified with empty parentheses within the GROUPING SET clause. It may also be specified directly in the GROUP BY clause, although there is no effect on the result of the query. "Example C4" on page 933 uses the grand-total syntax.

Combining grouping sets

This can be used to combine any of the types of GROUP BY clauses. When simple *grouping-expression* fields are combined with other groups, they are ″appended″ to the beginning of the resulting *grouping sets*. When ROLLUP or CUBE expressions are combined, they operate like ″multipliers″ on the remaining expression, forming additional grouping set entries according to the definition of either ROLLUP or CUBE.

For instance, combining *grouping-expression* elements acts as follows:

```
GROUP BY a, ROLLUP(b,c)
```

is equivalent to

```
GROUP BY GROUPING SETS((a,b,c)
                       (a,b)
                       (a) )
```

Or similarly,

```
GROUP BY a, b, ROLLUP(c,d)
```

is equivalent to

```
GROUP BY GROUPING SETS((a,b,c,d)
                       (a,b,c)
                       (a,b) )
```

Combining of *ROLLUP* elements acts as follows:

```
GROUP BY ROLLUP(a), ROLLUP(b,c)
```

is equivalent to

```
GROUP BY GROUPING SETS((a,b,c)
                       (a,b)
                       (a)
                       (b,c)
                       (b)
                       () )
```

Similarly,

```
GROUP BY ROLLUP(a), CUBE(b,c)
```

is equivalent to

```
GROUP BY GROUPING SETS((a,b,c)
                       (a,b)
                       (a,c)
                       (a)
                       (b,c)
                       (b)
                       (c)
                       () )
```

Combining of *CUBE* and *ROLLUP* elements acts as follows:

```
GROUP BY CUBE(a,b), ROLLUP(c,d)
```

is equivalent to

```
GROUP BY GROUPING SETS((a,b,c,d)
                       (a,b,c)
                       (a,b)
                       (a,c,d)
                       (a,c)
                       (a)
                       (b,c,d)
                       (b,c)
                       (b)
                       (c,d)
                       (c)
                       () )
```

Like a simple *grouping-expression*, combining grouping sets also eliminates duplicates within each grouping set. For instance,

```
GROUP BY a, ROLLUP(a,b)
```

is equivalent to

```
GROUP BY GROUPING SETS((a,b)
                       (a) )
```

A more complete example of combining grouping sets is to construct a result set that eliminates certain rows that would be returned for a full CUBE aggregation.

For example, consider the following GROUP BY clause:

```
GROUP BY Region,
         ROLLUP(Sales_Person, WEEK(Sales_Date)),
         CUBE(YEAR(Sales_Date), MONTH (Sales_Date))
```

The column listed immediately to the right of GROUP BY is simply grouped, those within the parenthesis following ROLLUP are rolled up, and those within the parenthesis following CUBE are cubed. Thus, the above clause results in a cube of MONTH within YEAR which is then rolled up within WEEK within Sales_Person

within the Region aggregation. It does not result in any grand total row or any cross-tabulation rows on Region, Sales_Person or WEEK(Sales_Date) so produces fewer rows than the clause:

```
GROUP BY ROLLUP (Region, Sales_Person, WEEK(Sales_Date),
                 YEAR(Sales_Date), MONTH(Sales_Date) )
```

having-clause

The HAVING clause specifies an intermediate result table that consists of those groups of R for which the *search-condition* is true. R is the result of the previous clause of the subselect. If this clause is not GROUP BY, R is considered to be a single group with no grouping columns.

Each *column-name* in the search condition must do one of the following:

- Unambiguously identify a grouping column of R.
- Be specified within a column function.
- Be a correlated reference. A *column-name* is a correlated reference if it identifies a column of a *table-reference* in an outer subselect.

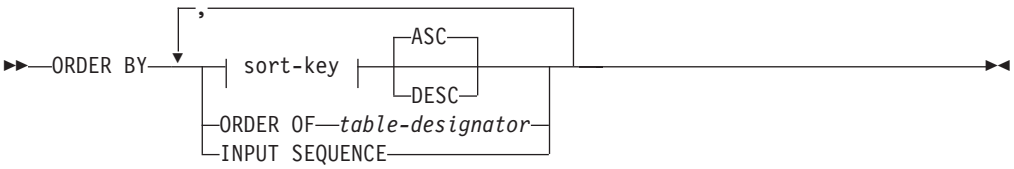
A group of R to which the search condition is applied supplies the argument for each column function in the search condition, except for any function whose argument is a correlated reference.

If the search condition contains a subquery, the subquery can be thought of as being executed each time the search condition is applied to a group of R, and the results used in applying the search condition. In actuality, the subquery is executed for each group only if it contains a correlated reference. For an illustration of the difference, see "Example A6" on page 928 and "Example A7" on page 928.

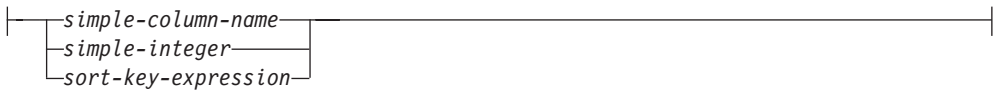
A correlated reference to a group of R must either identify a grouping column or be contained within a column function.

When HAVING is used without GROUP BY, the select list can only be a column name within a column function, a correlated column reference, a literal, or a special register.

order-by-clause



sort-key:



The ORDER BY clause specifies an ordering of the rows of the result table. If a single sort specification (one *sort-key* with associated direction) is identified, the rows are ordered by the values of that sort specification. If more than one sort specification is identified, the rows are ordered by the values of the first identified sort specification, then by the values of the second identified sort specification, and so on. Each *sort-key* cannot have a data type of LONG VARCHAR, CLOB, LONG VARGRAPHIC, DBCLOB, BLOB, DATALINK, distinct type on any of these types, or structured type (SQLSTATE 42907).

A named column in the select list may be identified by a *sort-key* that is a *simple-integer* or a *simple-column-name*. An unnamed column in the select list must be identified by an *simple-integer* or, in some cases, by a *sort-key-expression* that matches the expression in the select list (see details of *sort-key-expression*). A column is unnamed if the AS clause is not specified and it is derived from a constant, an expression with operators, or a function.

Ordering is performed in accordance with comparison rules. The null value is higher than all other values. If the ORDER BY clause does not completely order the rows, rows with duplicate values of all identified columns are displayed in an arbitrary order.

simple-column-name
Usually identifies a column of the result table. In this case, *simple-column-name* must be the column name of a named column in the select list.

The *simple-column-name* may also identify a column name of a table, view, or nested table identified in the FROM clause if the query is a subselect. An error occurs if the subselect:

- Specifies DISTINCT in the select-clause (SQLSTATE 42822)
- Produces a grouped result and the *simple-column-name* is not a *grouping-expression* (SQLSTATE 42803).

Determining which column is used for ordering the result is described under "Column names in sort keys" below.

simple-integer
Must be greater than 0 and not greater than the number of columns in the result table (SQLSTATE 42805). The integer *n* identifies the *n*th column of the result table.

sort-key-expression
An expression that is not simply a column name or an unsigned integer constant. The query to which ordering is applied must be a *subselect* to use this form of sort-key. The *sort-key-expression* cannot include a correlated scalar-fullselect (SQLSTATE 42703) or a function with an external action (SQLSTATE 42845).

Any column-name within a *sort-key-expression* must conform to the rules described under "Column names in sort keys" below.

There are a number of special cases that further restrict the expressions that can be specified.

- DISTINCT is specified in the SELECT clause of the subselect (SQLSTATE 42822).

  The sort-key-expression must match exactly with an expression in the select list of the subselect (scalar-fullselects are never matched).
- The subselect is grouped (SQLSTATE 42803).

The sort-key-expression can:

– be an expression in the select list of the subselect,

– include a *grouping-expression* from the GROUP BY clause of the subselect

– include a column function, constant or host variable.

ASC
Uses the values of the column in ascending order. This is the default.

DESC
Uses the values of the column in descending order.

ORDER OF *table-designator*
Specifies that the same ordering used in *table-designator* should be applied to the result table of the subselect. There must be a table reference matching *table-designator* in the FROM clause of the subselect that specifies this clause (SQLSTATE 42703). The subselect (or fullselect) corresponding to the specified *table-designator* must include an ORDER BY clause that is dependant on the data (SQLSTATE 428FI). The ordering that is applied is the same as if the columns of the ORDER BY clause in the nested subselect (or fullselect) were included in the outer subselect (or fullselect), and these columns were specified in place of the ORDER OF clause.

Note that this form is not allowed in a fullselect (other than the degenerative form of a fullselect). For example, the following is not valid:

```
(SELECT C1 FROM T1
   ORDER BY C1)
UNION
SELECT C1 FROM T2
   ORDER BY ORDER OF T1
```

The following example *is* valid:

```
SELECT C1 FROM
   (SELECT C1 FROM T1
      UNION
    SELECT C1 FROM T2
    ORDER BY C1 ) AS UTABLE
ORDER BY ORDER OF UTABLE
```

INPUT SEQUENCE
Specifies that, for an INSERT statement, the result table will reflect the input order of ordered data rows. INPUT SEQUENCE ordering can only be specified if an INSERT statement is used in a FROM clause (SQLSTATE 428G4). See "table-reference" on page 909. If INPUT SEQUENCE is specified and the input data is not ordered, the INPUT SEQUENCE clause is ignored.

Notes:

- **Column names in sort keys**:
  – The column name is qualified.

    The query must be a *subselect* (SQLSTATE 42877). The column name must unambiguously identify a column of some table, view or nested table in the FROM clause of the subselect (SQLSTATE 42702). The value of the column is used to compute the value of the sort specification.
  – The column name is unqualified.
    - The query is a subselect.

      If the column name is identical to the name of more than one column of the result table, the column name must unambiguously identify a column of some table, view or nested table in the FROM clause of the ordering

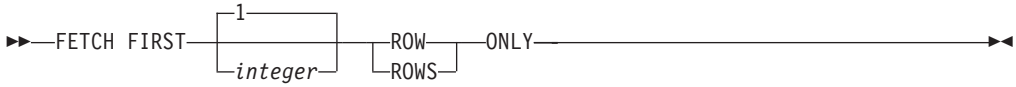
subselect (SQLSTATE 42702). If the column name is identical to one column, that column is used to compute the value of the sort specification. If the column name is not identical to a column of the result table, then it must unambiguously identify a column of some table, view or nested table in the FROM clause of the fullselect in the select-statement (SQLSTATE 42702).

- The query is not a subselect (it includes set operations such as union, except or intersect).

  The column name must not be identical to the name of more than one column of the result table (SQLSTATE 42702). The column name must be identical to exactly one column of the result table (SQLSTATE 42707), and this column is used to compute the value of the sort specification.

- **Limits**: The use of a *sort-key-expression* or a *simple-column-name* where the column is not in the select list may result in the addition of the column or expression to the temporary table used for sorting. This may result in reaching the limit of the number of columns in a table or the limit on the size of a row in a table. Exceeding these limits will result in an error if a temporary table is required to perform the sorting operation.

fetch-first-clause



The *fetch-first-clause* sets a maximum number of rows that can be retrieved. It lets the database manager know that the application does not want to retrieve more than *integer* rows, regardless of how many rows there might be in the result table when this clause is not specified. An attempt to fetch beyond *integer* rows is handled the same way as normal end of data (SQLSTATE 02000). The value of *integer* must be a positive integer (not zero).

Limiting the result table to the first *integer* rows can improve performance. The database manager will cease processing the query once it has determined the first *integer* rows. If both the *fetch-first-clause* and the *optimize-for-clause* are specified, the lower of the *integer* values from these clauses is used to influence the communications buffer size. The values are considered independently for optimization purposes.

If the fullselect contains an SQL data change statement in the FROM clause, all the rows are modified regardless of the limit on the number of rows to fetch.

Examples of subselects

Example A1: Select all columns and rows from the EMPLOYEE table.

```
SELECT * FROM EMPLOYEE
```

Example A2: Join the EMP_ACT and EMPLOYEE tables, select all the columns from the EMP_ACT table and add the employee's surname (LASTNAME) from the EMPLOYEE table to each row of the result.

```
SELECT EMP_ACT.*, LASTNAME
  FROM EMP_ACT, EMPLOYEE
  WHERE EMP_ACT.EMPNO = EMPLOYEE.EMPNO
```

Example A3: Join the EMPLOYEE and DEPARTMENT tables, select the employee number (EMPNO), employee surname (LASTNAME), department number (WORKDEPT in the EMPLOYEE table and DEPTNO in the DEPARTMENT table) and department name (DEPTNAME) of all employees who were born (BIRTHDATE) earlier than 1930.

```
SELECT EMPNO, LASTNAME, WORKDEPT, DEPTNAME
   FROM EMPLOYEE, DEPARTMENT
   WHERE WORKDEPT = DEPTNO
   AND YEAR(BIRTHDATE) < 1930
```

Example A4: Select the job (JOB) and the minimum and maximum salaries (SALARY) for each group of rows with the same job code in the EMPLOYEE table, but only for groups with more than one row and with a maximum salary greater than or equal to 27000.

```
SELECT JOB, MIN(SALARY), MAX(SALARY)
  FROM EMPLOYEE
  GROUP BY JOB
  HAVING COUNT(*) > 1
  AND MAX(SALARY) >= 27000
```

Example A5: Select all the rows of EMP_ACT table for employees (EMPNO) in department (WORKDEPT) 'E11'. (Employee department numbers are shown in the EMPLOYEE table.)

```
SELECT *
  FROM EMP_ACT
  WHERE EMPNO IN
           (SELECT EMPNO
               FROM EMPLOYEE
               WHERE WORKDEPT = 'E11')
```

Example A6: From the EMPLOYEE table, select the department number (WORKDEPT) and maximum departmental salary (SALARY) for all departments whose maximum salary is less than the average salary for all employees.

```
SELECT WORKDEPT, MAX(SALARY)
  FROM EMPLOYEE
  GROUP BY WORKDEPT
  HAVING MAX(SALARY) < (SELECT AVG(SALARY)
                           FROM EMPLOYEE)
```

The subquery in the HAVING clause would only be executed once in this example.

Example A7: Using the EMPLOYEE table, select the department number (WORKDEPT) and maximum departmental salary (SALARY) for all departments whose maximum salary is less than the average salary in all other departments.

```
SELECT WORKDEPT, MAX(SALARY)
   FROM EMPLOYEE EMP_COR
    GROUP BY WORKDEPT
    HAVING MAX(SALARY) < (SELECT AVG(SALARY)
                             FROM EMPLOYEE
                             WHERE NOT WORKDEPT = EMP_COR.WORKDEPT)
```

In contrast to "Example A6," the subquery in the HAVING clause would need to be executed for each group.

Example A8: Determine the employee number and salary of sales representatives along with the average salary and head count of their departments.

This query must first create a nested table expression (DINFO) in order to get the AVGSALARY and EMPCOUNT columns, as well as the DEPTNO column that is used in the WHERE clause.

```
SELECT THIS_EMP.EMPNO, THIS_EMP.SALARY, DINFO.AVGSALARY, DINFO.EMPCOUNT
 FROM EMPLOYEE THIS_EMP,
       (SELECT OTHERS.WORKDEPT AS DEPTNO,
                          AVG(OTHERS.SALARY) AS AVGSALARY,
                          COUNT(*) AS EMPCOUNT
         FROM EMPLOYEE OTHERS
         GROUP BY OTHERS.WORKDEPT
        ) AS DINFO
 WHERE THIS_EMP.JOB = 'SALESREP'
  AND THIS_EMP.WORKDEPT = DINFO.DEPTNO
```

Using a nested table expression for this case saves the overhead of creating the DINFO view as a regular view. During statement preparation, accessing the catalog for the view is avoided and, because of the context of the rest of the query, only the rows for the department of the sales representatives need to be considered by the view.

Example A9: Display the average education level and salary for 5 random groups of employees.

This query requires the use of a nested table expression to set a random value for each employee so that it can subsequently be used in the GROUP BY clause.

```
SELECT RANDID , AVG(EDLEVEL), AVG(SALARY)
  FROM ( SELECT EDLEVEL, SALARY, INTEGER(RAND()*5) AS RANDID
           FROM EMPLOYEE
       ) AS EMPRAND
  GROUP BY RANDID
```

Example A10: Query the EMP_ACT table and return those project numbers that have an employee whose salary is in the top 10 of all employees.

```
SELECT EMP_ACT.EMPNO,PROJNO
   FROM EMP_ACT
   WHERE EMP_ACT.EMPNO IN
       (SELECT EMPLOYEE.EMPNO
        FROM EMPLOYEE
        ORDER BY SALARY DESC
        FETCH FIRST 10 ROWS ONLY)
```

Examples of joins

Example B1: This example illustrates the results of the various joins using tables J1 and J2. These tables contain rows as shown.

```
SELECT * FROM J1

W   X
--- ------
A       11
B       12
C       13

SELECT * FROM J2

Y   Z
--- ------
A       21
C       22
D       23
```

The following query does an inner join of J1 and J2 matching the first column of both tables.

```
SELECT * FROM J1 INNER JOIN J2 ON W=Y

W   X      Y   Z
--- ------ --- ------
A       11 A       21
C       13 C       22
```

In this inner join example the row with column W='C' from J1 and the row with column Y='D' from J2 are not included in the result because they do not have a match in the other table. Note that the following alternative form of an inner join query produces the same result.

```
SELECT * FROM J1, J2 WHERE W=Y
```

The following left outer join will get back the missing row from J1 with nulls for the columns of J2. Every row from J1 is included.

```
SELECT * FROM J1 LEFT OUTER JOIN J2 ON W=Y

W   X      Y   Z
--- ------ --- ------
A       11 A       21
B       12 -        -
C       13 C       22
```

The following right outer join will get back the missing row from J2 with nulls for the columns of J1. Every row from J2 is included.

```
SELECT * FROM J1 RIGHT OUTER JOIN J2 ON W=Y

W   X      Y   Z
--- ------ --- ------
A       11 A       21
C       13 C       22
-        - D       23
```

The following full outer join will get back the missing rows from both J1 and J2 with nulls where appropriate. Every row from both J1 and J2 is included.

```
SELECT * FROM J1 FULL OUTER JOIN J2 ON W=Y

W   X      Y   Z
--- ------ --- ------
A       11 A       21
C       13 C       22
-        - D       23
B       12 -        -
```

Example B2: Using the tables J1 and J2 from the previous example, examine what happens when and additional predicate is added to the search condition.

```
SELECT * FROM J1 INNER JOIN J2 ON W=Y AND X=13

W   X      Y   Z
--- ------ --- ------
C       13 C       22
```

The additional condition caused the inner join to select only 1 row compared to the inner join in "Example B1" on page 929.

Notice what the impact of this is on the full outer join.

```
SELECT * FROM J1 FULL OUTER JOIN J2 ON W=Y AND X=13

W   X      Y   Z
--- ------ --- ------
-        - A       21
C       13 C       22
-        - D       23
A       11 -        -
B       12 -        -
```

The result now has 5 rows (compared to 4 without the additional predicate) since there was only 1 row in the inner join and all rows of both tables must be returned.

The following query illustrates that placing the same additional predicate in WHERE clause has completely different results.

```
SELECT * FROM J1 FULL OUTER JOIN J2 ON W=Y
   WHERE X=13

W   X      Y   Z
--- ------ --- ------
C       13 C       22
```

The WHERE clause is applied after the intermediate result of the full outer join. This intermediate result would be the same as the result of the full outer join query in "Example B1" on page 929. The WHERE clause is applied to this intermediate result and eliminates all but the row that has X=13. Choosing the location of a predicate when performing outer joins can have significant impact on the results. Consider what happens if the predicate was X=12 instead of X=13. The following inner join returns no rows.

```
SELECT * FROM J1 INNER JOIN J2 ON W=Y AND X=12
```

Hence, the full outer join would return 6 rows, 3 from J1 with nulls for the columns of J2 and 3 from J2 with nulls for the columns of J1.

```
SELECT * FROM J1 FULL OUTER JOIN J2 ON W=Y AND X=12

W   X      Y   Z
--- ------ --- ------
-        - A       21
-        - C       22
-        - D       23
A       11 -        -
B       12 -        -
C       13 -        -
```

If the additional predicate is in the WHERE clause instead, 1 row is returned.

```
SELECT * FROM J1 FULL OUTER JOIN J2 ON W=Y
   WHERE X=12

W   X      Y   Z
--- ------ --- ------
B       12 -        -
```

Example B3: List every department with the employee number and last name of the manager, including departments without a manager.

```
SELECT DEPTNO, DEPTNAME, EMPNO, LASTNAME
  FROM DEPARTMENT LEFT OUTER JOIN EMPLOYEE
       ON MGRNO = EMPNO
```

Example B4: List every employee number and last name with the employee number and last name of their manager, including employees without a manager.

```
SELECT E.EMPNO, E.LASTNAME, M.EMPNO, M.LASTNAME
  FROM EMPLOYEE E LEFT OUTER JOIN
                                        DEPARTMENT INNER JOIN EMPLOYEE M
       ON MGRNO = M.EMPNO
       ON E.WORKDEPT = DEPTNO
```

The inner join determines the last name for any manager identified in the DEPARTMENT table and the left outer join guarantees that each employee is listed even if a corresponding department is not found in DEPARTMENT.

Examples of grouping sets, cube, and rollup

The queries in "Example C1" through "Example C4" on page 933 use a subset of the rows in the SALES tables based on the predicate 'WEEK(SALES_DATE) = 13'.

```
SELECT WEEK(SALES_DATE) AS WEEK,
       DAYOFWEEK(SALES_DATE) AS DAY_WEEK,
       SALES_PERSON, SALES AS UNITS_SOLD
FROM SALES
WHERE WEEK(SALES_DATE) = 13
```

which results in:

```
WEEK        DAY_WEEK    SALES_PERSON    UNITS_SOLD
----------- ----------- --------------- -----------
         13           6 LUCCHESSI                 3
         13           6 LUCCHESSI                 1
         13           6 LEE                       2
         13           6 LEE                       2
         13           6 LEE                       3
         13           6 LEE                       5
         13           6 GOUNOT                    3
         13           6 GOUNOT                    1
         13           6 GOUNOT                    7
         13           7 LUCCHESSI                 1
         13           7 LUCCHESSI                 2
         13           7 LUCCHESSI                 1
         13           7 LEE                       7
         13           7 LEE                       3
         13           7 LEE                       7
         13           7 LEE                       4
         13           7 GOUNOT                    2
         13           7 GOUNOT                   18
         13           7 GOUNOT                    1
```

Example C1: Here is a query with a basic GROUP BY clause over 3 columns:

```
SELECT WEEK(SALES_DATE) AS WEEK,
       DAYOFWEEK(SALES_DATE) AS DAY_WEEK,
       SALES_PERSON, SUM(SALES) AS UNITS_SOLD
FROM SALES
WHERE WEEK(SALES_DATE) = 13
GROUP BY WEEK(SALES_DATE), DAYOFWEEK(SALES_DATE), SALES_PERSON
ORDER BY WEEK, DAY_WEEK, SALES_PERSON
```

This results in:

```
WEEK        DAY_WEEK    SALES_PERSON    UNITS_SOLD
----------- ----------- --------------- -----------
         13           6 GOUNOT                   11
         13           6 LEE                      12
         13           6 LUCCHESSI                 4
```

```
          13           7 GOUNOT                  21
          13           7 LEE                     21
          13           7 LUCCHESSI                4
```

Example C2: Produce the result based on two different grouping sets of rows from the SALES table.

```
SELECT WEEK(SALES_DATE) AS WEEK,
       DAYOFWEEK(SALES_DATE) AS DAY_WEEK,
       SALES_PERSON, SUM(SALES) AS UNITS_SOLD
FROM SALES
WHERE WEEK(SALES_DATE) = 13
GROUP BY GROUPING SETS ( (WEEK(SALES_DATE), SALES_PERSON),
                         (DAYOFWEEK(SALES_DATE), SALES_PERSON))
ORDER BY WEEK, DAY_WEEK, SALES_PERSON
```

This results in:

```
WEEK        DAY_WEEK    SALES_PERSON    UNITS_SOLD
----------- ----------- --------------- -----------
         13           - GOUNOT                  32
         13           - LEE                     33
         13           - LUCCHESSI                8
          -           6 GOUNOT                  11
          -           6 LEE                     12
          -           6 LUCCHESSI                4
          -           7 GOUNOT                  21
          -           7 LEE                     21
          -           7 LUCCHESSI                4
```

The rows with WEEK 13 are from the first grouping set and the other rows are from the second grouping set.

Example C3: If you use the 3 distinct columns involved in the grouping sets of "Example C2" and perform a ROLLUP, you can see grouping sets for (WEEK,DAY_WEEK,SALES_PERSON), (WEEK, DAY_WEEK), (WEEK) and grand total.

```
SELECT WEEK(SALES_DATE) AS WEEK,
       DAYOFWEEK(SALES_DATE) AS DAY_WEEK,
       SALES_PERSON, SUM(SALES) AS UNITS_SOLD
FROM SALES
WHERE WEEK(SALES_DATE) = 13
GROUP BY ROLLUP ( WEEK(SALES_DATE), DAYOFWEEK(SALES_DATE), SALES_PERSON )
ORDER BY WEEK, DAY_WEEK, SALES_PERSON
```

This results in:

```
WEEK        DAY_WEEK    SALES_PERSON    UNITS_SOLD
----------- ----------- --------------- -----------
         13           6 GOUNOT                  11
         13           6 LEE                     12
         13           6 LUCCHESSI                4
         13           6 -                       27
         13           7 GOUNOT                  21
         13           7 LEE                     21
         13           7 LUCCHESSI                4
         13           7 -                       46
         13           - -                       73
          -           - -                       73
```

Example C4: If you run the same query as "Example C3" only replace ROLLUP with CUBE, you can see additional grouping sets for (WEEK,SALES_PERSON), (DAY_WEEK,SALES_PERSON), (DAY_WEEK), (SALES_PERSON) in the result.

```
SELECT WEEK(SALES_DATE) AS WEEK,
       DAYOFWEEK(SALES_DATE) AS DAY_WEEK,
       SALES_PERSON, SUM(SALES) AS UNITS_SOLD
FROM SALES
WHERE WEEK(SALES_DATE) = 13
GROUP BY CUBE ( WEEK(SALES_DATE), DAYOFWEEK(SALES_DATE), SALES_PERSON )
ORDER BY WEEK, DAY_WEEK, SALES_PERSON
```

This results in:

```
WEEK        DAY_WEEK    SALES_PERSON    UNITS_SOLD
----------- ----------- --------------- -----------
         13           6 GOUNOT                   11
         13           6 LEE                      12
         13           6 LUCCHESSI                 4
         13           6 -                        27
         13           7 GOUNOT                   21
         13           7 LEE                      21
         13           7 LUCCHESSI                 4
         13           7 -                        46
         13           - GOUNOT                   32
         13           - LEE                      33
         13           - LUCCHESSI                 8
         13           - -                        73
          -           6 GOUNOT                   11
          -           6 LEE                      12
          -           6 LUCCHESSI                 4
          -           6 -                        27
          -           7 GOUNOT                   21
          -           7 LEE                      21
          -           7 LUCCHESSI                 4
          -           7 -                        46
          -           - GOUNOT                   32
          -           - LEE                      33
          -           - LUCCHESSI                 8
          -           - -                        73
```

Example C5: Obtain a result set which includes a grand-total of selected rows from the SALES table together with a group of rows aggregated by SALES_PERSON and MONTH.

```
SELECT SALES_PERSON,
       MONTH(SALES_DATE) AS MONTH,
       SUM(SALES) AS UNITS_SOLD
FROM SALES
GROUP BY GROUPING SETS ( (SALES_PERSON, MONTH(SALES_DATE)),
                         ()
                       )
ORDER BY SALES_PERSON, MONTH
```

This results in:

```
SALES_PERSON    MONTH       UNITS_SOLD
--------------- ----------- -----------
GOUNOT                    3          35
GOUNOT                    4          14
GOUNOT                   12           1
LEE                       3          60
LEE                       4          25
LEE                      12           6
LUCCHESSI                 3           9
LUCCHESSI                 4           4
LUCCHESSI                12           1
-                         -         155
```

Example C6: This example shows two simple ROLLUP queries followed by a query which treats the two ROLLUPs as grouping sets in a single result set and specifies row ordering for each column involved in the grouping sets.

Example C6-1:

```
SELECT WEEK(SALES_DATE) AS WEEK,
       DAYOFWEEK(SALES_DATE) AS DAY_WEEK,
       SUM(SALES) AS UNITS_SOLD
FROM SALES
GROUP BY ROLLUP ( WEEK(SALES_DATE), DAYOFWEEK(SALES_DATE) )
ORDER BY WEEK, DAY_WEEK
```

results in:

```
WEEK        DAY_WEEK    UNITS_SOLD
----------- ----------- -----------
         13           6          27
         13           7          46
         13           -          73
         14           1          31
         14           2          43
         14           -          74
         53           1           8
         53           -           8
          -           -         155
```

Example C6-2:

```
SELECT MONTH(SALES_DATE) AS MONTH,
       REGION,
       SUM(SALES) AS UNITS_SOLD
FROM SALES
GROUP BY ROLLUP ( MONTH(SALES_DATE), REGION );
ORDER BY MONTH, REGION
```

results in:

```
MONTH       REGION          UNITS_SOLD
----------- --------------- -----------
          3 Manitoba                 22
          3 Ontario-North             8
          3 Ontario-South            34
          3 Quebec                   40
          3 -                       104
          4 Manitoba                 17
          4 Ontario-North             1
          4 Ontario-South            14
          4 Quebec                   11
          4 -                        43
         12 Manitoba                  2
         12 Ontario-South             4
         12 Quebec                    2
         12 -                         8
          - -                       155
```

Example C6-3:

```
SELECT WEEK(SALES_DATE) AS WEEK,
       DAYOFWEEK(SALES_DATE) AS DAY_WEEK,
       MONTH(SALES_DATE) AS MONTH,
       REGION,
       SUM(SALES) AS UNITS_SOLD
FROM SALES
GROUP BY GROUPING SETS ( ROLLUP( WEEK(SALES_DATE), DAYOFWEEK(SALES_DATE) ),
                         ROLLUP( MONTH(SALES_DATE), REGION )  )
ORDER BY WEEK, DAY_WEEK, MONTH, REGION
```

results in:

```
WEEK        DAY_WEEK    MONTH       REGION          UNITS_SOLD
----------- ----------- ----------- --------------- -----------
         13           6           - -                        27
         13           7           - -                        46
         13           -           - -                        73
         14           1           - -                        31
         14           2           - -                        43
         14           -           - -                        74
         53           1           - -                         8
         53           -           - -                         8
          -           -           3 Manitoba                 22
          -           -           3 Ontario-North             8
          -           -           3 Ontario-South            34
          -           -           3 Quebec                   40
          -           -           3 -                       104
          -           -           4 Manitoba                 17
          -           -           4 Ontario-North             1
          -           -           4 Ontario-South            14
          -           -           4 Quebec                   11
          -           -           4 -                        43
          -           -          12 Manitoba                  2
          -           -          12 Ontario-South             4
          -           -          12 Quebec                    2
          -           -          12 -                         8
          -           -           - -                       155
          -           -           - -                       155
```

Using the two ROLLUPs as grouping sets causes the result to include duplicate rows. There are even two grand total rows.

Observe how the use of ORDER BY has affected the results:

- In the first grouped set, week 53 has been repositioned to the end.
- In the second grouped set, month 12 has now been positioned to the end and the regions now appear in alphabetic order.
- Null values are sorted high.

Example C7: In queries that perform multiple ROLLUPs in a single pass (such as "Example C6-3" on page 935) you may want to be able to indicate which grouping set produced each row. The following steps demonstrate how to provide a column (called GROUP) which indicates the origin of each row in the result set. By origin, we mean which one of the two grouping sets produced the row in the result set.

Step 1: Introduce a way of "generating" new data values, using a query which selects from a VALUES clause (which is an alternate form of a fullselect). This query shows how a table can be derived called "X" having 2 columns "R1" and "R2" and 1 row of data.

```
SELECT R1,R2
FROM (VALUES('GROUP 1','GROUP 2')) AS X(R1,R2);
```

results in:

```
R1      R2
------- -------
GROUP 1 GROUP 2
```

Step 2: Form the cross product of this table "X" with the SALES table. This add columns "R1" and "R2" to every row.

```
SELECT R1, R2, WEEK(SALES_DATE) AS WEEK,
       DAYOFWEEK(SALES_DATE) AS DAY_WEEK,
       MONTH(SALES_DATE) AS MONTH,
```

```
       REGION,
       SALES AS UNITS_SOLD
FROM SALES,(VALUES('GROUP 1','GROUP 2')) AS X(R1,R2)
```

This add columns ″R1″ and ″R2″ to every row.

Step 3: Now we can combine these columns with the grouping sets to include these columns in the rollup analysis.

```
SELECT R1, R2,
       WEEK(SALES_DATE) AS WEEK,
       DAYOFWEEK(SALES_DATE) AS DAY_WEEK,
       MONTH(SALES_DATE) AS MONTH,
       REGION, SUM(SALES) AS UNITS_SOLD
FROM SALES,(VALUES('GROUP 1','GROUP 2')) AS X(R1,R2)
GROUP BY GROUPING SETS ((R1, ROLLUP(WEEK(SALES_DATE),
                         DAYOFWEEK(SALES_DATE))),
                       (R2,ROLLUP( MONTH(SALES_DATE), REGION ) )  )
ORDER BY WEEK, DAY_WEEK, MONTH, REGION
```

results in:

```
R1      R2      WEEK     DAY_WEEK  MONTH     REGION          UNITS_SOLD
------- ------- -------- --------- --------- --------------- -----------
GROUP 1 -             13         6         - -                          27
GROUP 1 -             13         7         - -                          46
GROUP 1 -             13         -         - -                          73
GROUP 1 -             14         1         - -                          31
GROUP 1 -             14         2         - -                          43
GROUP 1 -             14         -         - -                          74
GROUP 1 -             53         1         - -                           8
GROUP 1 -             53         -         - -                           8
-       GROUP 2        -         -         3 Manitoba                   22
-       GROUP 2        -         -         3 Ontario-North               8
-       GROUP 2        -         -         3 Ontario-South              34
-       GROUP 2        -         -         3 Quebec                     40
-       GROUP 2        -         -         3 -                         104
-       GROUP 2        -         -         4 Manitoba                   17
-       GROUP 2        -         -         4 Ontario-North               1
-       GROUP 2        -         -         4 Ontario-South              14
-       GROUP 2        -         -         4 Quebec                     11
-       GROUP 2        -         -         4 -                          43
-       GROUP 2        -         -        12 Manitoba                    2
-       GROUP 2        -         -        12 Ontario-South               4
-       GROUP 2        -         -        12 Quebec                      2
-       GROUP 2        -         -        12 -                           8
-       GROUP 2        -         -         - -                         155
GROUP 1 -              -         -         - -                         155
```

Step 4: Notice that because R1 and R2 are used in different grouping sets, whenever R1 is non-null in the result, R2 is null and whenever R2 is non-null in the result, R1 is null. That means you can consolidate these columns into a single column using the COALESCE function. You can also use this column in the ORDER BY clause to keep the results of the two grouping sets together.

```
SELECT COALESCE(R1,R2) AS GROUP,
       WEEK(SALES_DATE) AS WEEK,
       DAYOFWEEK(SALES_DATE) AS DAY_WEEK,
       MONTH(SALES_DATE) AS MONTH,
       REGION, SUM(SALES) AS UNITS_SOLD
FROM SALES,(VALUES('GROUP 1','GROUP 2')) AS X(R1,R2)
GROUP BY GROUPING SETS ((R1, ROLLUP(WEEK(SALES_DATE),
                         DAYOFWEEK(SALES_DATE))),
                       (R2,ROLLUP( MONTH(SALES_DATE), REGION ) )  )
ORDER BY GROUP, WEEK, DAY_WEEK, MONTH, REGION;
```

results in:

```
GROUP   WEEK        DAY_WEEK    MONTH       REGION          UNITS_SOLD
------- ----------- ----------- ----------- --------------- -----------
GROUP 1          13           6           - -                          27
GROUP 1          13           7           - -                          46
GROUP 1          13           -           - -                          73
GROUP 1          14           1           - -                          31
GROUP 1          14           2           - -                          43
GROUP 1          14           -           - -                          74
GROUP 1          53           1           - -                           8
GROUP 1          53           -           - -                           8
GROUP 1           -           -           - -                         155
GROUP 2           -           -           3 Manitoba                   22
GROUP 2           -           -           3 Ontario-North               8
GROUP 2           -           -           3 Ontario-South              34
GROUP 2           -           -           3 Quebec                     40
GROUP 2           -           -           3 -                         104
GROUP 2           -           -           4 Manitoba                   17
GROUP 2           -           -           4 Ontario-North               1
GROUP 2           -           -           4 Ontario-South              14
GROUP 2           -           -           4 Quebec                     11
GROUP 2           -           -           4 -                          43
GROUP 2           -           -          12 Manitoba                    2
GROUP 2           -           -          12 Ontario-South               4
GROUP 2           -           -          12 Quebec                      2
GROUP 2           -           -          12 -                           8
GROUP 2           -           -           - -                         155
```

Example C8: The following example illustrates the use of various column functions when performing a CUBE. The example also makes use of cast functions and rounding to produce a decimal result with reasonable precision and scale.

```
SELECT MONTH(SALES_DATE) AS MONTH,
       REGION,
       SUM(SALES) AS UNITS_SOLD,
       MAX(SALES) AS BEST_SALE,
       CAST(ROUND(AVG(DECIMAL(SALES)),2) AS DECIMAL(5,2)) AS AVG_UNITS_SOLD
FROM SALES
GROUP BY CUBE(MONTH(SALES_DATE),REGION)
ORDER BY MONTH, REGION
```

This results in:

```
MONTH       REGION          UNITS_SOLD  BEST_SALE   AVG_UNITS_SOLD
----------- --------------- ----------- ----------- --------------
          3 Manitoba                 22           7           3.14
          3 Ontario-North             8           3           2.67
          3 Ontario-South            34          14           4.25
          3 Quebec                   40          18           5.00
          3 -                       104          18           4.00
          4 Manitoba                 17           9           5.67
          4 Ontario-North             1           1           1.00
          4 Ontario-South            14           8           4.67
          4 Quebec                   11           8           5.50
          4 -                        43           9           4.78
         12 Manitoba                  2           2           2.00
         12 Ontario-South             4           3           2.00
         12 Quebec                    2           1           1.00
         12 -                         8           3           1.60
          - Manitoba                 41           9           3.73
          - Ontario-North             9           3           2.25
          - Ontario-South            52          14           4.00
          - Quebec                   53          18           4.42
          - -                       155          18           3.87
```

Related reference:

- Chapter 20, “Identifiers,” on page 809
- “Functions” in the *SQL Reference, Volume 1*
- “GROUPING aggregate function” in the *SQL Reference, Volume 1*
- “Fullselect” in the *SQL Reference, Volume 1*
- “Select-statement” in the *SQL Reference, Volume 1*
- “DELETE” on page 670
- “INSERT” on page 724
- “UPDATE” on page 757
- “CREATE FUNCTION (SQL Scalar, Table, or Row) statement” in the *SQL Reference, Volume 2*
- “CREATE FUNCTION (External Table) statement” in the *SQL Reference, Volume 2*
- “Character strings” in the *SQL Reference, Volume 1*
- “Assignments and comparisons” in the *SQL Reference, Volume 1*
- “Predicates” in the *SQL Reference, Volume 1*

Examples of grouping sets, cube, and rollup

Chapter 27. Application Considerations

© Copyright IBM Corp. 1993-2004

Security Considerations when Using SQL in Applications

Package Creation for Embedded SQL

To run applications written in compiled host languages, you must create the packages needed by the database manager at execution time. This involves the following steps as shown in the following figure:

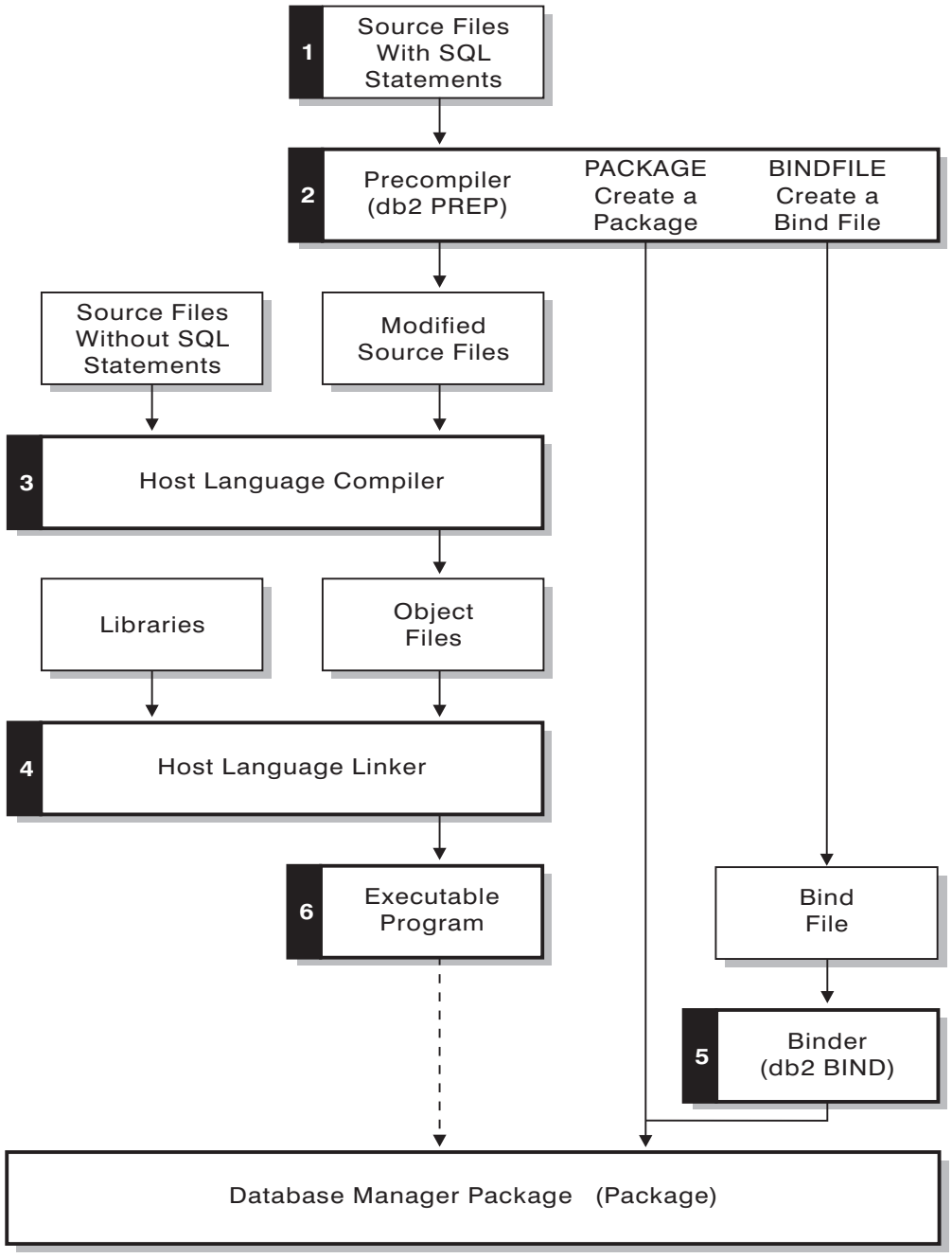


Figure 19. Preparing Programs Written in Compiled Host Languages

- Precompiling (step 2), to convert embedded SQL source statements into a form the database manager can use,

- Compiling and linking (steps 3 and 4), to create the required object modules, and,
- Binding (step 5), to create the package to be used by the database manager when the program is run.

Related concepts:

- "Precompilation of Source Files Containing Embedded SQL" on page 943
- "Source File Requirements for Embedded SQL Applications" on page 945
- "Compilation and Linkage of Source Files Containing Embedded SQL" on page 946
- "Package Creation Using the BIND Command" on page 947
- "Package Versioning" in the *Application Development Guide: Programming Client Applications*
- "Effect of Special Registers on Bound Dynamic SQL" in the *Application Development Guide: Programming Client Applications*
- "Resolution of Unqualified Table Names" in the *Application Development Guide: Programming Client Applications*
- "Additional Considerations when Binding" in the *Application Development Guide: Programming Client Applications*
- "Advantages of Deferred Binding" in the *Application Development Guide: Programming Client Applications*
- "Application, Bind File, and Package Relationships" in the *Application Development Guide: Programming Client Applications*
- "Precompiler-Generated Timestamps" in the *Application Development Guide: Programming Client Applications*
- "Package Rebinding" in the *Application Development Guide: Programming Client Applications*

Related reference:

- "db2bfd - Bind File Description Tool Command" in the *Command Reference*

Precompilation of Source Files Containing Embedded SQL

After you create the source files, you must precompile each host language file containing SQL statements with the `PREP` command for host-language source files. The precompiler converts SQL statements contained in the source file to comments, and generates the DB2 run-time API calls for those statements.

Before precompiling an application you must connect to a server, either implicitly or explicitly. Although you precompile application programs at the client workstation and the precompiler generates modified source and messages on the client, the precompiler uses the server connection to perform some of the validation.

The precompiler also creates the information the database manager needs to process the SQL statements against a database. This information is stored in a package, in a bind file, or in both, depending on the precompiler options selected.

A typical example of using the precompiler follows. To precompile a C embedded SQL source file called *filename.sqc*, you can issue the following command to create a C source file with the default name `filename.c` and a bind file with the default name `filename.bnd`:

```
DB2® PREP filename.sqc BINDFILE
```

The precompiler generates up to four types of output:

Modified Source
This file is the new version of the original source file after the precompiler converts the SQL statements into DB2 run-time API calls. It is given the appropriate host language extension.

Package If you use the PACKAGE option (the default), or do not specify any of the BINDFILE, SYNTAX, or SQLFLAG options, the package is stored in the connected database. The package contains all the information required to execute the static SQL statements of a particular source file against this database only. Unless you specify a different name with the PACKAGE USING option, the precompiler forms the package name from the first 8 characters of the source file name.

If you use the PACKAGE option without SQLERROR CONTINUE, the database used during the precompile process must contain all of the database objects referenced by the static SQL statements in the source file. For example, you cannot precompile a SELECT statement unless the table it references exists in the database.

With the VERSION option the bindfile, (if the BINDFILE option is used), and the package (either if bound at PREP time or if a bound separately) will be designated with a particular version identifier. Many versions of packages with the same name and creator can exit at once.

Bind File If you use the BINDFILE option, the precompiler creates a bind file (with extension `.bnd`) that contains the data required to create a package. This file can be used later with the `BIND` command to bind the application to one or more databases. If you specify BINDFILE and do not specify the PACKAGE option, binding is deferred until you invoke the `BIND` command. Note that for the command line processor (CLP), the default for `PREP` does not specify the BINDFILE option. Thus, if you are using the CLP and want the binding to be deferred, you need to specify the BINDFILE option.

Specifying SQLERROR CONTINUE creates a package, even if errors occur when binding SQL statements. Those statements that fail 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.

Message File If you use the MESSAGES option, the precompiler redirects messages to the indicated file. These messages include warnings and error messages that describe problems encountered during precompilation. If the source file does not precompile successfully, use the warning and error messages to determine the problem, correct the source file, and then attempt to precompile the source file again. If you do not use the MESSAGES option, precompilation messages are written to the standard output.

Related concepts:

- "Package Versioning" in the *Application Development Guide: Programming Client Applications*

Related reference:

- "PRECOMPILE" on page 842

Source File Requirements for Embedded SQL Applications

You must always precompile a source file against a specific database, even if eventually you do not use the database with the application. In practice, you can use a test database for development, and after you fully test the application, you can bind its bind file to one or more production databases. This practice is known as *deferred binding*.

If your application uses a code page that is not the same as your database code page, you need to consider which code page to use when precompiling.

If your application uses user-defined functions (UDFs) or user-defined distinct types (UDTs), you may need to use the FUNCPATH option when you precompile your application. This option specifies the function path that is used to resolve UDFs and UDTs for applications containing static SQL. If FUNCPATH is not specified, the default function path is *SYSIBM*, *SYSFUN*, *USER*, where *USER* refers to the current user ID.

To precompile an application program that accesses more than one server, you can do one of the following:

- Split the SQL statements for each database into separate source files. Do not mix SQL statements for different databases in the same file. Each source file can be precompiled against the appropriate database. This is the recommended method.
- Code your application using dynamic SQL statements only, and bind against each database your program will access.
- If all the databases look the same, that is, they have the same definition, you can group the SQL statements together into one source file.

The same procedures apply if your application will access a host, AS/400® or iSeries application server through DB2 Connect. Precompile it against the server to which it will be connecting, using the `PREP` options available for that server.

If you are precompiling an application that will run on DB2 Universal Database for z/OS and OS/390, consider using the flagger facility to check the syntax of the SQL statements. The flagger indicates SQL syntax that is supported by DB2 Universal Database, but not supported by DB2 Universal Database for z/OS and OS/390. You can also use the flagger to check that your SQL syntax conforms to the SQL92 Entry Level syntax. You can use the SQLFLAG option on the `PREP` command to invoke it and to specify the version of DB2 Universal Database for z/OS and OS/390 SQL syntax to be used for comparison. The flagger facility will not enforce any changes in SQL use; it only issues informational and warning messages regarding syntax incompatibilities, and does not terminate preprocessing abnormally.

Related concepts:

- "Advantages of Deferred Binding" in the *Application Development Guide: Programming Client Applications*

- "Character conversion between different code pages" in the *Application Development Guide: Programming Client Applications*
- "When code page conversion occurs" in the *Application Development Guide: Programming Client Applications*
- "Character Substitutions During Code Page Conversions" in the *Application Development Guide: Programming Client Applications*
- "Supported Code Page Conversions" in the *Application Development Guide: Programming Client Applications*
- "Code Page Conversion Expansion Factor" in the *Application Development Guide: Programming Client Applications*

Related reference:

- "PRECOMPILE" on page 842

Compilation and Linkage of Source Files Containing Embedded SQL

Compile the modified source files and any additional source files that do not contain SQL statements using the appropriate host language compiler. The language compiler converts each modified source file into an *object module*.

Refer to the programming documentation for your operating platform for any exceptions to the default compiler options. Refer to your compiler's documentation for a complete description of available compiler options.

The host language linker creates an executable application. For example:

- On Windows® operating systems, the application can be an executable file or a dynamic link library (DLL).
- On UNIX®-based systems, the application can be an executable load module or a shared library.

Note: Although applications can be DLLs on Windows operating systems, the DLLs are loaded directly by the application and not by the DB2® database manager. On Windows operating systems, the database manager can load DLLs. Stored procedures are normally built as DLLs or shared libraries.

To create the executable file, link the following:

- User object modules, generated by the language compiler from the modified source files and other files not containing SQL statements.
- Host language library APIs, supplied with the language compiler.
- The database manager library containing the database manager APIs for your operating environment. Refer to the appropriate programming documentation for your operating platform for the specific name of the database manager library you need for your database manager APIs.

Related concepts:

- "DB2 Stored Procedures" in the *Application Development Guide: Programming Client Applications*

Related tasks:

- "Building and Running REXX Applications" in the *Application Development Guide: Programming Client Applications*

- "Building JDBC applets" in the *Application Development Guide: Building and Running Applications*
- "Building JDBC applications" in the *Application Development Guide: Building and Running Applications*
- "Building SQLJ applets" in the *Application Development Guide: Building and Running Applications*
- "Building SQLJ applications" in the *Application Development Guide: Building and Running Applications*
- "Building UNIX C applications" in the *Application Development Guide: Building and Running Applications*
- "Building UNIX C++ applications" in the *Application Development Guide: Building and Running Applications*
- "Building IBM COBOL applications on AIX" in the *Application Development Guide: Building and Running Applications*
- "Building UNIX Micro Focus COBOL applications" in the *Application Development Guide: Building and Running Applications*

Package Creation Using the BIND Command

Binding is the process that creates the package the database manager needs to access the database when the application is executed. Binding can be done implicitly by specifying the PACKAGE option during precompilation, or explicitly by using the `BIND` command against the bind file created during precompilation.

A typical example of using the `BIND` command follows. To bind a bind file named *filename.bnd* to the database, you can issue the following command:

```
DB2® BIND filename.bnd
```

One package is created for each separately precompiled source code module. If an application has five source files, of which three require precompilation, three packages or bind files are created. By default, each package is given a name that is the same as the name of the source module from which the `.bnd` file originated, but truncated to 8 characters. To explicitly specify a different package name, you must use the PACKAGE USING option on the `PREP` command. The version of a package is given by the VERSION precompile option and defaults to the empty string. If the name and schema of this newly created package is the same as a package that currently exists in the target database, but the version identifier differs, a new package is created and the previous package still remains. However if a package exists that matches the name, schema and the version of the package being bound, then that package is dropped and replaced with the new package being bound (specifying ACTION ADD on the bind would prevent that and an error (SQL0719) would be returned instead).

Related reference:

- "BIND" on page 232
- "PRECOMPILE" on page 842

Generation of Sequential Values

Generating sequential values is a common database application development problem. The best solution to that problem is to use sequence objects and sequence expressions in SQL. Each *sequence object* is a uniquely named database object that can be accessed only by sequence expressions. There are two *sequence expressions*:

the PREVVAL expression and the NEXTVAL expression. The PREVVAL expression returns the value most recently generated in the application process for the specified sequence object. Any NEXTVAL expressions occuring in the same statement as the PREVVAL expression have no effect on the value generated by the PREVAL expression in that statement. The NEXTVAL sequence expression increments the value of the sequence object and returns the new value of the sequence object.

To create a sequence object, issue the CREATE SEQUENCE statement. For example, to create a sequence object called id_values using the default attributes, issue the following statement:

```
CREATE SEQUENCE id_values
```

To generate the first value in the application session for the sequence object, issue a VALUES statement using the NEXTVAL expression:

```
VALUES NEXTVAL FOR id_values

1
-----------
          1

  1 record(s) selected.
```

To display the current value of the sequence object, issue a VALUES statement using the PREVVAL expression:

```
VALUES PREVVAL FOR id_values

1
-----------
          1

  1 record(s) selected.
```

You can repeatedly retrieve the current value of the sequence object, and the value that the sequence object returns does not change until you issue a NEXTVAL expression. In the following example, the PREVVAL expression returns a value of 1, until the NEXTVAL expression in the current connection increments the value of the sequence object:

```
VALUES PREVVAL FOR id_values

1
-----------
          1

  1 record(s) selected.

VALUES PREVVAL FOR id_values

1
-----------
          1

  1 record(s) selected.

VALUES NEXTVAL FOR id_values

1
-----------
          2

  1 record(s) selected.
```

```
VALUES PREVVAL FOR id_values

1
-----------
          2

            1 record(s) selected.
```

To update the value of a column with the next value of the sequence object, include the NEXTVAL expression in the UPDATE statement, as follows:

```
UPDATE staff
  SET id = NEXTVAL FOR id_values
  WHERE id = 350
```

To insert a new row into a table using the next value of the sequence object, include the NEXTVAL expression in the INSERT statement, as follows:

```
INSERT INTO staff (id, name, dept, job)
  VALUES (NEXTVAL FOR id_values, 'Kandil', 51, 'Mgr')
```

Related reference:

- "CREATE SEQUENCE statement" in the *SQL Reference, Volume 2*

Related samples:

- "DbSeq.java -- How to create, alter and drop a sequence in a database (JDBC)"

Management of Sequence Behavior

You can tailor the behavior of sequence objects to meet the needs of your application. You change change the attributes of a sequence object when you issue the CREATE SEQUENCE statement to create a new sequence object, and when you issue the ALTER SEQUENCE statement for an existing sequence object. Following are some of the attributes of a sequence object that you can specify:

Data type
: The AS clause of the CREATE SEQUENCE statement specifies the numeric data type of the sequence object. The data type determines the possible minimum and maximum values of the sequence object (the minimum and maximum values for a data type are listed in the topic describing SQL limits). You cannot change the data type of a sequence object; instead, you must drop the sequence object by issuing the DROP SEQUENCE statement and issue a CREATE SEQUENCE statement with the new data type.

Start value
: The START WITH clause of the CREATE SEQUENCE statement sets the initial value of the sequence object. The RESTART WITH clause of the ALTER SEQUENCE statement resets the value of the sequence object to a specified value.

Minimum value
: The MINVALUE clause sets the minimum value of the sequence object.

Maximum value
: The MAXVALUE clause sets the maximum value of the sequence object.

Increment value
: The INCREMENT BY clause sets the value that each NEXTVAL expression adds to the current value of the sequence object. To decrement the value of the sequence object, specify a negative value.

Sequence cycling

The CYCLE clause causes the value of a sequence object that reaches its maximum or minimum value to generate its respective minimum value or maximum value on the following NEXTVAL expression.

For example, to create a sequence object called id_values that starts with a minimum value of 0, has a maximum value of 1000, increments by 2 with each NEXTVAL expression, and returns to its minimum value when the maximum value is reached, issue the following statement:

```
CREATE SEQUENCE id_values
  START WITH 0
  INCREMENT BY 2
  MAXVALUE 1000
  CYCLE
```

Related reference:

- "SQL limits" in the *SQL Reference, Volume 1*
- "ALTER SEQUENCE statement" in the *SQL Reference, Volume 2*
- "CREATE SEQUENCE statement" in the *SQL Reference, Volume 2*

Sequence Objects Compared to Identity Columns

Although sequence objects and identity columns appear to serve similar purposes for DB2® applications, there is an important difference. An identity column automatically generates values for a column in a single table. A sequence object generates sequential values upon request that can be used in any SQL statement.

Authorization Considerations for Embedded SQL

An *authorization* allows a user or group to perform a general task such as connecting to a database, creating tables, or administering a system. A *privilege* gives a user or group the right to access one specific database object in a specified way. DB2® uses a set of privileges to provide protection for the information that you store in it.

Most SQL statements require some type of privilege on the database objects which the statement utilizes. Most API calls usually do not require any privilege on the database objects which the call utilizes, however, many APIs require that you possess the necessary authority in order to invoke them. The DB2 APIs enable you to perform the DB2 administrative functions from within your application program. For example, to recreate a package stored in the database without the need for a bind file, you can use the `sqlarbnd` (or REBIND) API.

When you design your application, consider the privileges your users will need to run the application. The privileges required by your users depend on:

- Whether your application uses dynamic SQL, including JDBC and DB2 CLI, or static SQL. For information about the privileges required to issue a statement, see the description of that statement.
- Which APIs the application uses. For information about the privileges and authorities required for an API call, see the description of that API.

Consider two users, PAYROLL and BUDGET, who need to perform queries against the STAFF table. PAYROLL is responsible for paying the employees of the company, so it needs to issue a variety of SELECT statements when issuing paychecks. PAYROLL needs to be able to access each employee's salary. BUDGET

is responsible for determining how much money is needed to pay the salaries. BUDGET should not, however, be able to see any particular employee's salary.

Because PAYROLL issues many different SELECT statements, the application you design for PAYROLL could probably make good use of dynamic SQL. The dynamic SQL would require that PAYROLL have SELECT privilege on the STAFF table. This requirement is not a problem because PAYROLL requires full access to the table.

BUDGET, on the other hand, should not have access to each employee's salary. This means that you should not grant SELECT privilege on the STAFF table to BUDGET. Because BUDGET does need access to the total of all the salaries in the STAFF table, you could build a static SQL application to execute a SELECT SUM(SALARY) FROM STAFF, bind the application and grant the EXECUTE privilege on your application's package to BUDGET. This enables BUDGET to obtain the required information, without exposing the information that BUDGET should not see.

Related concepts:

- "Authorization Considerations for Dynamic SQL" on page 951
- "Authorization Considerations for Static SQL" on page 952
- "Authorization Considerations for APIs" on page 959
- "Authorization" on page 15

Authorization Considerations for Dynamic SQL

To use dynamic SQL in a package bound with DYNAMICRULES RUN (default), the person who runs a dynamic SQL application must have the privileges necessary to issue each SQL request performed, as well as the EXECUTE privilege on the package. The privileges may be granted to the user's authorization ID, to any group of which the user is a member, or to PUBLIC.

If you bind the application with the DYNAMICRULES BIND option, DB2 associates your authorization ID with the application packages. This allows any user who runs the application to inherit the privileges associated with your authorization ID.

If the program contains no static SQL, the person binding the application (for embedded dynamic SQL applications) only needs the BINDADD authority on the database. Again, this privilege can be granted to the user's authorization ID, to a group of which the user is a member, or to PUBLIC.

When a package exhibits bind or define behavior, the user that runs the application needs only the EXECUTE privilege on the package to run it. At run-time, the binder of a package that exhibits bind behavior must have the privileges necessary to execute all the dynamic statements generated by the package, because all authorization checking for dynamic statements is done using the ID of the binder and not the executors. Similarly, the definer of a routine whose package exhibits define behavior must have all the privileges necessary to execute all the dynamic statements generated by the define behavior package. If you have SYSADM or DBADM authority and create a bind behavior package, consider using the OWNER BIND option to designate a different authorization ID. The OWNER BIND option prevents a package from automatically inheriting SYSADM or DBADM privileges within dynamic SQL statements. For more information on the

DYNAMICRULES and OWNER bind options, refer to the BIND command. For more information on package behaviors, see the description of DYNAMICRULES effects on dynamic SQL statements.

Related concepts:

- "Authorization Considerations for Embedded SQL" on page 950
- "Authorization Considerations for Static SQL" on page 952
- "Authorization Considerations for APIs" on page 959
- "Authorizations and binding of routines that contain SQL" on page 32

Related reference:

- "BIND" on page 232

Authorization Considerations for Static SQL

To use static SQL, the user running the application only needs the EXECUTE privilege on the package. No privileges are required for each of the statements that make up the package. The EXECUTE privilege may be granted to the user's authorization ID, to any group of which the user is a member, or to PUBLIC.

Unless you specify the VALIDATE RUN option when binding the application, the authorization ID you use to bind the application must have the privileges necessary to perform all the statements in the application. If VALIDATE RUN was specified at BIND time, all authorization failures for any static SQL within this package will not cause the BIND to fail and those statements will be revalidated at run time. The person binding the application must always have BINDADD authority. The privileges needed to execute the statements must be granted to the user's authorization ID or to PUBLIC. Group privileges are not used when binding static SQL statements. As with dynamic SQL, the BINDADD privilege can be granted to the user authorization ID, to a group of which the user is a member, or to PUBLIC.

These properties of static SQL give you very precise control over access to information in DB2®.

Related concepts:

- "Authorization Considerations for Embedded SQL" on page 950
- "Authorization Considerations for Dynamic SQL" on page 951
- "Authorization Considerations for APIs" on page 959

Related reference:

- "BIND" on page 232

Effect of DYNAMICRULES bind option on dynamic SQL

The PRECOMPILE and BIND option DYNAMICRULES determines what values apply at run-time for the following dynamic SQL attributes:

- The authorization ID that is used during authorization checking.
- The qualifier that is used for qualification of unqualified objects.
- Whether the package can be used to dynamically prepare the following statements: GRANT, REVOKE, ALTER, CREATE, DROP, COMMENT ON, RENAME, SET INTEGRITY and SET EVENT MONITOR STATE statements.

In addition to the DYNAMICRULES value, the run-time environment of a package controls how dynamic SQL statements behave at run-time. The two possible run-time environments are:

- The package runs as part of a stand-alone program
- The package runs within a routine context

The combination of the DYNAMICRULES value and the run-time environment determine the values for the dynamic SQL attributes. That set of attribute values is called the dynamic SQL statement behavior. The four behaviors are:

Run behavior DB2® uses the authorization ID of the user (the ID that initially connected to DB2) executing the package as the value to be used for authorization checking of dynamic SQL statements and for the initial value used for implicit qualification of unqualified object references within dynamic SQL statements.

Bind behavior At run-time, DB2 uses all the rules that apply to static SQL for authorization and qualification. That is, take the authorization ID of the package owner as the value to be used for authorization checking of dynamic SQL statements and the package default qualifier for implicit qualification of unqualified object references within dynamic SQL statements.

Define behavior

Define behavior applies only if the dynamic SQL statement is in a package that is run within a routine context, and the package was bound with DYNAMICRULES DEFINEBIND or DYNAMICRULES DEFINERUN. DB2 uses the authorization ID of the routine definer (not the routine's package binder) as the value to be used for authorization checking of dynamic SQL statements and for implicit qualification of unqualified object references within dynamic SQL statements within that routine.

Invoke behavior

Invoke behavior applies only if the dynamic SQL statement is in a package that is run within a routine context, and the package was bound with DYNAMICRULES INVOKEBIND or DYNAMICRULES INVOKERUN. DB2 uses the current statement authorization ID in effect when the routine is invoked as the value to be used for authorization checking of dynamic SQL and for implicit qualification of unqualified object references within dynamic SQL statements within that routine. This is summarized by the following table:

Invoking Environment	ID Used
Any static SQL	Implicit or explicit value of the OWNER of the package the SQL invoking the routine came from.
Used in definition of view or trigger	Definer of the view or trigger.
Dynamic SQL from a run behavior package	ID used to make the initial connection to DB2.
Dynamic SQL from a define behavior package	Definer of the routine that uses the package that the SQL invoking the routine came from.
Dynamic SQL from an invoke behavior package	Current® authorization ID invoking the routine.

The following table shows the combination of the DYNAMICRULES value and the run-time environment that yields each dynamic SQL behavior.

Table 71. How DYNAMICRULES and the Run-Time Environment Determine Dynamic SQL Statement Behavior

DYNAMICRULES Value	Behavior of Dynamic SQL Statements in a Standalone Program Environment	Behavior of Dynamic SQL Statements in a Routine Environment
BIND	Bind behavior	Bind behavior
RUN	Run behavior	Run behavior
DEFINEBIND	Bind behavior	Define behavior
DEFINERUN	Run behavior	Define behavior
INVOKEBIND	Bind behavior	Invoke behavior
INVOKERUN	Run behavior	Invoke behavior

The following table shows the dynamic SQL attribute values for each type of dynamic SQL behavior.

Table 72. Definitions of Dynamic SQL Statement Behaviors

Dynamic SQL Attribute	Setting for Dynamic SQL Attributes: Bind Behavior	Setting for Dynamic SQL Attributes: Run Behavior	Setting for Dynamic SQL Attributes: Define Behavior	Setting for Dynamic SQL Attributes: Invoke Behavior
Authorization ID	The implicit or explicit value of the OWNER BIND option	ID of User Executing Package	Routine definer (not the routine's package owner)	Current statement authorization ID when routine is invoked.
Default qualifier for unqualified objects	The implicit or explicit value of the QUALIFIER BIND option	CURRENT SCHEMA Special Register	Routine definer (not the routine's package owner)	Current statement authorization ID when routine is invoked.
Can execute GRANT, REVOKE, ALTER, CREATE, DROP, COMMENT ON, RENAME, SET INTEGRITY and SET EVENT MONITOR STATE	No	Yes	No	No

Related concepts:

- "Authorization Considerations for Dynamic SQL" on page 951
- "Authorizations and binding of routines that contain SQL" on page 32

When to use DB2 CLI or embedded SQL

Which interface you choose depends on your application.

DB2 CLI is ideally suited for query-based graphical user interface (GUI) applications that require portability. The advantages listed above, may make using DB2 CLI seem like the obvious choice for any application. There is however, one

factor that must be considered, the comparison between static and dynamic SQL. It is much easier to use static SQL in embedded applications.

Static SQL has several advantages:

- Performance

  Dynamic SQL is prepared at run time, static SQL is prepared at precompile time. As well as requiring more processing, the preparation step may incur additional network-traffic at run time. The additional network traffic can be avoided if the DB2 CLI application makes use of deferred prepare (which is the default behavior).

  It is important to note that static SQL will not always have better performance than dynamic SQL. Dynamic SQL is prepared at runtime and uses the database statistics available at that time, whereas static SQL makes use of database statistics available at BIND time. Dynamic SQL can make use of changes to the database, such as new indexes, to choose the optimal access plan, resulting in potentially better performance than the same SQL executed as static SQL. In addition, precompilation of dynamic SQL statements can be avoided if they are cached.
- Encapsulation and Security

  In static SQL, the authorizations to access objects (such as a table, view) are associated with a package and are validated at package binding time. This means that database administrators need only to grant execute on a particular package to a set of users (thus encapsulating their privileges in the package) without having to grant them explicit access to each database object. In dynamic SQL, the authorizations are validated at run time on a per statement basis; therefore, users must be granted explicit access to each database object. This permits these users access to parts of the object that they do not have a need to access.
- Embedded SQL is supported in languages other than C or C++.
- For fixed query selects, embedded SQL is simpler.

If an application requires the advantages of both interfaces, it is possible to make use of static SQL within a DB2 CLI application by creating a stored procedure that contains the static SQL. The stored procedure is called from within a DB2 CLI application and is executed on the server. Once the stored procedure is created, any DB2 CLI or ODBC application can call it.

It is also possible to write a mixed application that uses both DB2 CLI and embedded SQL, taking advantage of their respective benefits. In this case, DB2 CLI is used to provide the base application, with key modules written using static SQL for performance or security reasons. This complicates the application design, and should only be used if stored procedures do not meet the applications requirements.

Ultimately, the decision on when to use each interface, will be based on individual preferences and previous experience rather than on any one factor.

Related concepts:

- "CLI/ODBC/JDBC trace facility" in the *CLI Guide and Reference, Volume 1*

Related tasks:

- "Preparing and executing SQL statements in CLI applications" in the *CLI Guide and Reference, Volume 1*

- "Issuing SQL statements in CLI applications" in the *CLI Guide and Reference, Volume 1*
- "Creating static SQL with CLI/ODBC/JDBC Static Profiling" in the *CLI Guide and Reference, Volume 1*

Units of work

A transaction is commonly referred to in DB2® Universal Database (DB2 UDB) as a *unit of work*. A unit of work is a recoverable sequence of operations within an application process. It is used by the database manager to ensure that a database is in a consistent state. Any reading from or writing to the database is done within a unit of work.

For example, a bank transaction might involve the transfer of funds from a savings account to a checking account. After the application subtracts an amount from the savings account, the two accounts are inconsistent, and remain so until the amount is added to the checking account. When *both* steps are completed, a point of consistency is reached. The changes can be committed and made available to other applications.

A unit of work starts when the first SQL statement is issued against the database. The application must end the unit of work by issuing either a COMMIT or a ROLLBACK statement. The COMMIT statement makes permanent all changes made within a unit of work. The ROLLBACK statement removes these changes from the database. If the application ends normally without either of these statements being explicitly issued, the unit of work is automatically committed. If it ends abnormally in the middle of a unit of work, the unit of work is automatically rolled back. Once issued, a COMMIT or a ROLLBACK cannot be stopped. With some multi-threaded applications, or some operating systems (such as Windows®), if the application ends normally without either of these statements being explicitly issued, the unit of work is automatically rolled back. It is recommended that your applications always explicitly commit or roll back complete units of work. If part of a unit of work does not complete successfully, the updates are rolled back, leaving the participating tables as they were before the transaction began. This ensures that requests are neither lost nor duplicated.

Related reference:

- "COMMIT" on page 885
- "ROLLBACK" on page 900

Remote unit of work

A *remote unit of work* lets a user or application program read or update data at one location per unit of work. It supports access to one database within a unit of work. While an application program can update several remote databases, it can only access one database within a unit of work.

Remote unit of work has the following characteristics:

- Multiple requests (SQL statements) per unit of work are supported.
- Multiple cursors per unit of work are supported.
- Each unit of work can update only one database.

- The application program either commits or rolls back the unit of work. In certain error circumstances, the database server or DB2 Connect may roll back the unit of work.

For example, Figure 20 shows a database client running a funds transfer application that accesses a database containing checking and savings account tables, as well as a banking fee schedule. The application must:

- Accept the amount to transfer from the user interface.
- Subtract the amount from the savings account, and determine the new balance.
- Read the fee schedule to determine the transaction fee for a savings account with the given balance.
- Subtract the transaction fee from the savings account.
- Add the amount of the transfer to the checking account.
- Commit the transaction (unit of work).

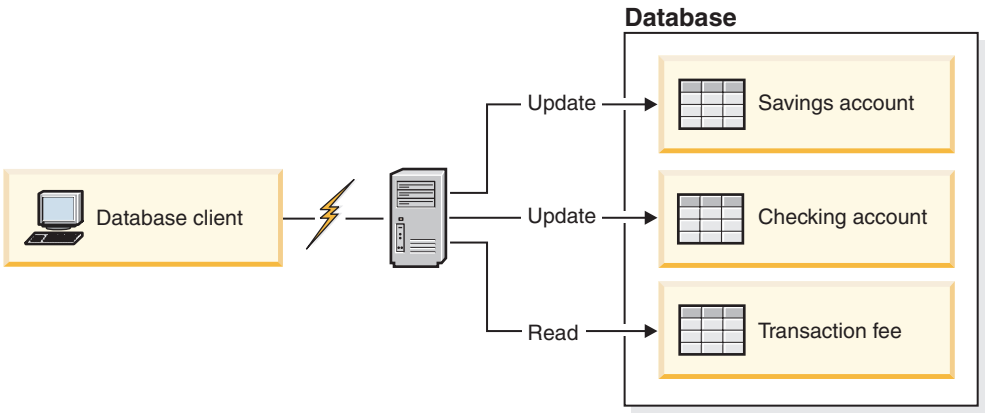


Figure 20. Using a Single Database in a Transaction

To set up such an application, you must:

1. Create the tables for the savings account, checking account and banking fee schedule in the same database.
2. If physically remote, set up the database server to use the appropriate communications protocol.
3. If physically remote, catalog the node and the database to identify the database on the database server.
4. Precompile your application program to specify a type 1 connection; that is, specify CONNECT(1) on the PREP command.

Related concepts:

- "Distributed Relational Database Architecture" in the *DB2 Connect User's Guide*
- "DB2 Connect and DRDA" in the *DB2 Connect User's Guide*
- "Distributed requests" in the *DB2 Connect User's Guide*
- "Remote Unit of Work" in the *Application Development Guide: Programming Client Applications*

Compound SQL guidelines

To reduce database manager overhead, you can group several SQL statements into a single executable block. Because the SQL statements in the block are substatements that could be executed individually, this kind of code is called *compound SQL*. In addition to reducing database manager overhead, compound SQL reduces the number of requests that have to be transmitted across the network for remote clients.

There are two types of compound SQL:

- **Atomic**

  The application receives a response from the database manager when all substatements have completed successfully or when one substatement ends in an error. If one substatement ends in an error, the entire block is considered to have ended in an error. Any changes made to the database within the block are rolled back.

  Atomic compound SQL is not supported with DB2 Connect
- **Not Atomic**

  The application receives a response from the database manager when all substatements have completed. All substatements within a block are executed regardless of whether or not the preceding substatement completed successfully. The group of statements can only be rolled back if the unit of work containing the NOT ATOMIC compound SQL is rolled back.

Compound SQL is supported in stored procedures, which are also known as DARI routines, and in the following application development processes:

- Embedded static SQL
- DB2 Call Level Interface
- JDBC

Dynamic Compound SQL Statements

Dynamic compound statements are compiled by DB2® as a single statement. This statement can be used effectively for short scripts that require little control flow logic but significant data flow. For larger constructs with nested complex control flow, consider using SQL procedures.

In a dynamic compound statement you can use the following elements in declarations:

- SQL variables in variable declarations of substatements
- Conditions in the substatements based on the SQLSTATE values of the condition declaration
- One or more SQL procedural statements

Dynamic compound statements can also use several flow logic statements, such as the FOR statement, the IF statement, the ITERATE statement, and the WHILE statement.

If an error occurs in a dynamic compound statement, all prior SQL statements are rolled back and the remaining SQL statements in the dynamic compound statement are not processed.

A dynamic compound statement can be embedded in a trigger, SQL function, or SQL method, or issued through dynamic SQL statements. This executable statement can be dynamically prepared. No privileges are required to invoke the statement but the authorization ID associated with the statement must have the necessary privileges to invoke the SQL statements in the compound statement.

Related concepts:

- "Query tuning guidelines" in the *Administration Guide: Performance*

Authorization Considerations for APIs

Most of the APIs provided by DB2® do not require the use of privileges, however, many do require some kind of authority to invoke. For the APIs that do require a privilege, the privilege must be granted to the user running the application. The privilege may be granted to the user's authorization ID, to any group of which the user is a member, or to PUBLIC. For information on the required privilege and authority to issue each API call, see the description of the API.

Some APIs can be accessed via a stored procedure interface. For information whether a specific API can be accessed via a stored procedure, see the description of that API.

Related concepts:

- "Authorization Considerations for Embedded SQL" on page 950
- "Authorization Considerations for Dynamic SQL" on page 951
- "Authorization Considerations for Static SQL" on page 952

Purpose of Multiple-Thread Database Access

One feature of some operating systems is the ability to run several threads of execution within a single process. The multiple threads allow an application to handle asynchronous events, and makes it easier to create event-driven applications, without resorting to polling schemes. The information that follows describes how the database manager works with multiple threads, and lists some design guidelines that you should keep in mind.

If you are not familiar with terms relating to the development of multithreaded applications (such as critical section and semaphore), consult the programming documentation for your operating system.

A DB2 application can execute SQL statements from multiple threads using *contexts*. A context is the environment from which an application runs all SQL statements and API calls. All connections, units of work, and other database resources are associated with a specific context. Each context is associated with one or more threads within an application.

For each executable SQL statement in a context, the first run-time services call always tries to obtain a latch. If it is successful, it continues processing. If not (because an SQL statement in another thread of the same context already has the latch), the call is blocked on a signaling semaphore until that semaphore is posted, at which point the call gets the latch and continues processing. The latch is held until the SQL statement has completed processing, at which time it is released by the last run-time services call that was generated for that particular SQL statement.

The net result is that each SQL statement within a context is executed as an atomic unit, even though other threads may also be trying to execute SQL statements at the same time. This action ensures that internal data structures are not altered by different threads at the same time. APIs also use the latch used by run-time services; therefore, APIs have the same restrictions as run-time services routines within each context.

For DB2® Version 8, all Version 8 applications are multithreaded by default, and are capable of using multiple contexts. (The behavior of pre-Version 8 applications remains unchanged.) If you want, you can use the following DB2 APIs to use multiple contexts. Specifically, your application can create a context for a thread, attach to or detach from a separate context for each thread, and pass contexts between threads. If your application does not call *any* of these APIs, DB2 will automatically manage the multiple contexts for your application:

- `sqleBeginCtx()`
- `sqleEndCtx()`
- `sqleAttachToCtx()`
- `sqleDetachFromCtx()`
- `sqleGetCurrentCtx()`
- `sqleInterruptCtx()`

Contexts may be exchanged between threads in a process, but not exchanged between processes. One use of multiple contexts is to provide support for concurrent transactions.

Related concepts:

- "Concurrent Transactions" in the *Application Development Guide: Programming Client Applications*

Related reference:

- "sqleAttachToCtx - Attach to Context" in the *Administrative API Reference*
- "sqleBeginCtx - Create and Attach to an Application Context" in the *Administrative API Reference*
- "sqleDetachFromCtx - Detach From Context" in the *Administrative API Reference*
- "sqleEndCtx - Detach and Destroy Application Context" in the *Administrative API Reference*
- "sqleGetCurrentCtx - Get Current Context" in the *Administrative API Reference*
- "sqleInterruptCtx - Interrupt Context" in the *Administrative API Reference*

Related samples:

- "dbthrds.sqc -- How to use multiple context APIs on UNIX (C)"
- "dbthrds.sqC -- How to use multiple context APIs on UNIX (C++)"

Ending a Transaction with the COMMIT Statement

The COMMIT statement ends the current transaction and makes the database changes performed during the transaction visible to other processes.

Procedure:

Commit changes as soon as application requirements permit. In particular, write your programs so that uncommitted changes are not held while waiting for input

from a terminal, as this can result in database resources being held for a long time. Holding these resources prevents other applications that need these resources from running.

Your application programs should explicitly end any transactions before terminating.

If you do not end transactions explicitly, DB2 automatically commits all the changes made during the program's pending transaction when the program ends successfully, except on Windows operating systems. On Windows operating systems, if you do not explicitly commit the transaction, the database manager always rolls back the changes.

DB2 rolls back the changes under the following conditions:

- A log full condition
- Any other system condition that causes database manager processing to end

The COMMIT statement has no effect on the contents of host variables.

Related concepts:

- "Implicit Ending of a Transaction in a Standalone Application" in the *Application Development Guide: Programming Client Applications*
- "Return Codes" in the *Application Development Guide: Programming Client Applications*
- "Error Information in the SQLCODE, SQLSTATE, and SQLWARN Fields" in the *Application Development Guide: Programming Client Applications*

Related tasks:

- "Ending an Application Program" in the *Application Development Guide: Programming Client Applications*

Related reference:

- "COMMIT" on page 885

Ending a Transaction with the ROLLBACK Statement

To ensure the consistency of data at the transaction level, the database manager ensures that either *all* operations within a transaction are completed, or *none* are completed. Suppose, for example, that the program is supposed to deduct money from one account and add it to another. If you place both of these updates in a single transaction, and a system failure occurs while they are in progress, when you restart the system the database manager automatically performs crash recovery to restore the data to the state it was in before the transaction began. If a program error occurs, the database manager restores all changes made by the statement in error. The database manager will not undo work performed in the transaction prior to execution of the statement in error, unless you specifically roll it back.

Procedure:

To prevent the changes that were effected by the transaction from being committed to the database, issue the ROLLBACK statement to end the transaction. The ROLLBACK statement returns the database to the state it was in before the transaction ran.

Note: On Windows operating systems, if you do not explicitly commit the transaction, the database manager always rolls back the changes.

If you use a ROLLBACK statement in a routine that was entered because of an error or warning and you use the SQL WHENEVER statement, then you should specify WHENEVER SQLERROR CONTINUE and WHENEVER SQLWARNING CONTINUE before the ROLLBACK. This avoids a program loop if the ROLLBACK fails with an error or warning.

In the event of a severe error, you will receive a message indicating that you cannot issue a ROLLBACK statement. Do not issue a ROLLBACK statement if a severe error occurs such as the loss of communications between the client and server applications, or if the database gets corrupted. After a severe error, the only statement you can issue is a CONNECT statement.

The ROLLBACK statement has no effect on the contents of host variables.

You can code one or more transactions within a single application program, and it is possible to access more than one database from within a single transaction. A transaction that accesses more than one database is called a multisite update.

Related concepts:

- "Implicit Ending of a Transaction in a Standalone Application" in the *Application Development Guide: Programming Client Applications*
- "Remote Unit of Work" in the *Application Development Guide: Programming Client Applications*
- "Multisite Update" in the *Application Development Guide: Programming Client Applications*

Related reference:

- "CONNECT (Type 1)" on page 887
- "CONNECT (Type 2)" on page 893
- "WHENEVER statement" in the *SQL Reference, Volume 2*

Security and Java Applications

The sections that follow describe security considerations for SQLJ, JDBC, the Type 2 JDBC driver, and the Universal JDBC driver.

SQLJ Considerations

Controlling the execution of SQL statements in SQLJ

You can use selected methods of the SQLJ `ExecutionContext` class to control or monitor the execution of SQL statements. Selected sqlj.runtime classes and interfaces describes those methods.

To use `ExecutionContext` methods, follow these steps:

1. Acquire an *execution context*.

   There are two ways to acquire an execution context:

   - Acquire the default execution context from the connection context. For example:

     ```
     ExecutionContext execCtx = connCtx.getExecutionContext();
     ```

- Create a new execution context by invoking the constructor for `ExecutionContext`. For example:

  ```
  ExecutionContext execCtx=new ExecutionContext();
  ```

2. Associate the execution context with an SQL statement.

   To do that, specify an execution context after the connection context in the execution clause that contains the SQL statement. For example:

   ```
   #sql [connCtx, execCtx] {DELETE FROM EMPLOYEE WHERE SALARY > 10000};
   ```

3. Invoke `ExecutionContext` methods.

   Some ExecutionContext methods are applicable before the associated SQL statement is executed, and some are applicable only after their associated SQL statement is executed.

   For example, you can use method `getUpdateCount` to count the number of rows that are deleted by a DELETE statement after you execute the DELETE statement:

   ```
   #sql [connCtx, execCtx] {DELETE FROM EMPLOYEE WHERE SALARY > 10000};
   System.out.println("Deleted " + execCtx.getUpdateCount() + " rows");
   ```

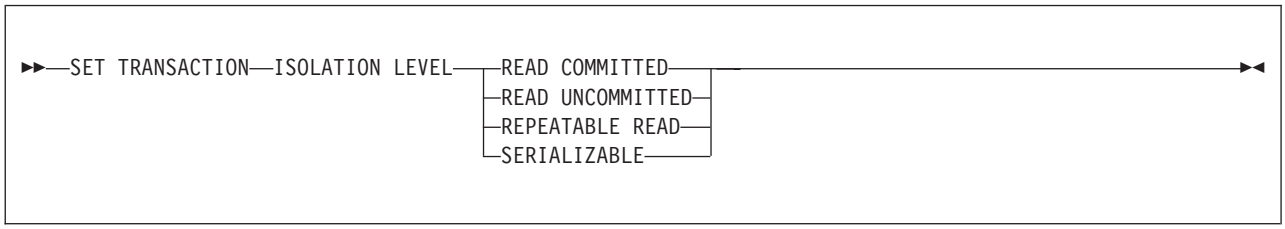
Related reference:

- "Selected sqlj.runtime classes and interfaces" in the *Application Development Guide: Programming Client Applications*

SQLJ SET-TRANSACTION-clause

The SET TRANSACTION clause sets the isolation level for the current unit of work.

Syntax:



Description:

ISOLATION LEVEL
Specifies one of the following isolation levels:

READ COMMITTED
Specifies that the current DB2 isolation level is cursor stability.

READ UNCOMMITTED
Specifies that the current DB2 isolation level is uncommitted read.

REPEATABLE READ
Specifies that the current DB2 isolation level is read stability.

SERIALIZABLE
Specifies that the current DB2 isolation level is repeatable read.

Usage notes:

You can execute SET TRANSACTION only at the beginning of a transaction.

Setting the isolation level for an SQLJ transaction

To set the isolation level for a unit of work within an SQLJ program, use the SET TRANSACTION ISOLATION LEVEL clause. Table 73 shows the values that you can specify in the SET TRANSACTION ISOLATION LEVEL clause and their DB2® equivalents.

Table 73. Equivalent SQLJ and DB2 isolation levels

SET TRANSACTION value	DB2 isolation level
SERIALIZABLE	Repeatable read
REPEATABLE READ	Read stability
READ COMMITTED	Cursor stability
READ UNCOMMITTED	Uncommitted read

The isolation level affects the underlying JDBC connection as well as the SQLJ connection. You can change the isolation level only at the beginning of a transaction.

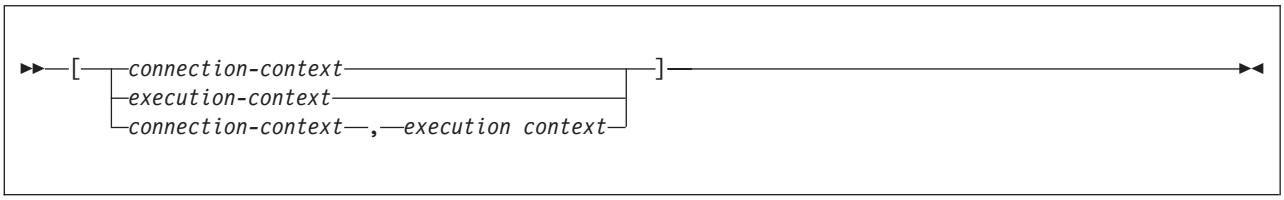
Related concepts:

- "Isolation levels" in the *SQL Reference, Volume 1*

SQLJ context-clause

A context clause specifies a connection context, an execution context, or both. You use a connection context to connect to a data source. You use an execution context to monitor and modify SQL statement execution.

Syntax:



Description:

connection-context
Specifies a valid Java identifier that is declared earlier in the SQLJ program. That identifier must be declared as an instance of the connection context class that SQLJ generates for a connection declaration clause.

execution-context
Specifies a valid Java identifier that is declared earlier in the SQLJ program. That identifier must be declared as an instance of class `sqlj.runtime.ExecutionContext`.

Usage notes:

- If you do not specify a connection context in an executable clause, SQLJ uses the default connection context.
- If you do not specify an execution context, SQLJ obtains the execution context from the connection context of the statement.

Related tasks:

- "Connecting to a data source using SQLJ" on page 965
- "Controlling the execution of SQL statements in SQLJ" on page 962

Connecting to a data source using SQLJ

In an SQLJ application, as in any other DB2® application, you must be connected to a database server before you can execute SQL statements. In SQLJ, as in JDBC, a database server is called a *data source*.

You can use one of five techniques to connect to a data source:

- Explicitly create a connection using the JDBC `DriverManager` interface. There are two techniques for doing this.
- Explicitly create a connection using the JDBC `DataSource` interface. There are two techniques for doing this.
- Implicitly create a connection.

Connection technique 1: This technique uses the JDBC `DriverManager` as the underlying means for creating the connection. Use it with any level of the JDBC driver.

1. Execute an SQLJ *connection declaration clause*.

   Doing this generates a *connection context class*. The simplest form of the connection declaration clause is:

   ```
   #sql context context-class-name;
   ```

   The name of the generated connection context class is *context-class-name*.
2. Load a JDBC driver by invoking the `Class.forName` method:
   - For the DB2 Universal JDBC Driver, invoke `Class.forName` this way:

     ```
     Class.forName("com.ibm.db2.jcc.DB2Driver");
     ```
   - For the DB2 JDBC Type 2 Driver, invoke `Class.forName` this way:

     ```
     Class.forName("COM.ibm.db2.jdbc.app.DB2Driver");
     ```
3. Invoke the constructor for the connection context class that you created in step 1.

   Doing this creates a connection context object that you specify in each SQL statement that you execute at the associated data source. The constructor invocation statement needs to be in one of the following forms:

   ```
   connection-context-class connection-context-object=
     new connection-context-class(String url, boolean autocommit);

   connection-context-class connection-context-object=
     new connection-context-class(String url, String user,
       String password, boolean autocommit);
   connection-context-class connection-context-object=
     new connection-context-class(String url, Properties info,
       boolean autocommit);
   ```

   The meanings of the parameters are:

   url A string that specifies the location name that is associated with the data source. That argument has one of the forms that are specified in Connect to a data source using the DriverManager interface with the JDBC Universal Driver. The form depends on which JDBC driver you are using.

   user **and** *password*
   Specify a user ID and password for connection to the data source, if the data source to which you are connecting requires them.

info
Specifies an object of type `java.util.Properties` that contains a set of driver properties for the connection. For the DB2 JDBC Type 2 Driver for Linux, UNIX® and Windows® (DB2 JDBC Type 2 Driver), you should specify only the `user` and `password` properties. For the DB2 Universal JDBC Driver, you can specify any of the properties listed in Properties for the DB2 Universal JDBC Driver.

autocommit
Specifies whether you want the database manager to issue a COMMIT after every statement. Possible values are `true` or `false`. If you specify `false`, you need to do explicit commit operations.

The following code uses connection technique 1 to create a connection to location NEWYORK. The connection requires a user ID and password, and does not require autocommit. The numbers to the right of selected statements correspond to the previously-described steps.

```
#sql context Ctx;             // Create connection context class Ctx        1
String userid="dbadm";        // Declare variables for user ID and password
String password="dbadm";
String empname;               // Declare a host variable
...
try {                         // Load the JDBC driver
  Class.forName("com.ibm.db2.jcc.DB2Driver");                              2
}
catch (ClassNotFoundException e) {
   e.printStackTrace();
}
Ctx myConnCtx=                                                             3
  new Ctx("jdbc:db2://sysmvs1.stl.ibm.com:5021/NEWYORK",
  userid,password,false);     // Create connection context object myConnCtx
                              // for the connection to NEWYORK
#sql [myConnCtx] {SELECT LASTNAME INTO :empname FROM EMPLOYEE
   WHERE EMPNO='000010'};
                              // Use myConnCtx for executing an SQL statement
```

Figure 21. Using connection technique 1 to connect to a data source

Connection technique 2: This technique uses the JDBC `DriverManager` interface for creating the connection. Use it with any level of the JDBC driver.

1. Execute an SQLJ connection declaration clause.

   This is the same as step 1 on page 965 in connection technique 1.

2. Load the driver.

   This is the same as step 2 on page 965 in connection technique 1.

3. Invoke the JDBC `DriverManager.getConnection` method.

   Doing this creates a JDBC connection object for the connection to the data source. You can use any of the forms of `getConnection` that are specified in Connect to a data source using the DriverManager interface with the JDBC Universal Driver.

   The meanings of the *url*, *user*, and *password* parameters are the same as the meanings of the parameters in step 3 on page 965 of connection technique 1.

4. Invoke the constructor for the connection context class that you created in step 1.

   Doing this creates a connection context object that you specify in each SQL statement that you execute at the associated data source. The constructor invocation statement needs to be in the following form:

```
connection-context-class connection-context-object=
  new connection-context-class(Connection JDBC-connection-object);
```

The *JDBC-connection-object* parameter is the `Connection` object that you created in step 3 on page 966.

The following code uses connection technique 2 to create a connection to location NEWYORK. The connection requires a user ID and password, and does not require autocommit. The numbers to the right of selected statements correspond to the previously-described steps.

```
#sql context Ctx;           // Create connection context class Ctx        1
String userid="dbadm";      // Declare variables for user ID and password
String password="dbadm";
String empname;             // Declare a host variable
...
try {                       // Load the JDBC driver
  Class.forName("com.ibm.db2.jcc.DB2Driver");                            2
}
catch (ClassNotFoundException e) {
   e.printStackTrace();
}
Connection jdbccon=                                                      3
  DriverManager.getConnection("jdbc:db2://sysmvs1.stl.ibm.com:5021/NEWYORK",
    userid,password);
                            // Create JDBC connection object jdbccon
jdbccon.setAutoCommit(false); // Do not autocommit                       4
Ctx myConnCtx=new Ctx(jdbccon);                                          5
                            // Create connection context object myConnCtx
                            // for the connection to NEWYORK
#sql [myConnCtx] {SELECT LASTNAME INTO :empname FROM EMPLOYEE
   WHERE EMPNO='000010'};
                            // Use myConnCtx for executing an SQL statement
```

Figure 22. Using connection technique 2 to connect to a data source

Connection technique 3: This technique uses the JDBC `DataSource` interface for creating the connection.

1. Execute an SQLJ connection declaration clause.

   This is the same as step 1 on page 965 in connection technique 1.

2. If your system administrator created a `DataSource` object in a different program:
   a. Obtain the logical name of the data source to which you need to connect.
   b. Create a context to use in the next step.
   c. In your application program, use the Java™ Naming and Directory Interface (JNDI) to get the `DataSource` object that is associated with the logical data source name.

   Otherwise, create a `DataSource` object and assign properties to it, as shown in "Creating and using a DataSource object in the same application" in Connect to a data source using the DataSource interface.

3. Invoke the JDBC `DataSource.getConnection` method.

   Doing this creates a JDBC connection object for the connection to the data source. You can use one of the following forms of `getConnection`:

   ```
   getConnection();
   getConnection(user, password);
   ```

   The meanings of *user* and *password* parameters are the same as the meanings of the parameters in step 3 on page 965 of connection technique 1.

4. If the default autocommit mode is not appropriate, invoke the JDBC `Connection.setAutoCommit` method.

   Doing this indicates whether you want the database manager to issue a COMMIT after every statement. The form of this method is:

   ```
   setAutoCommit(boolean autocommit);
   ```

5. Invoke the constructor for the connection context class that you created in step 1 on page 967.

   Doing this creates a connection context object that you specify in each SQL statement that you execute at the associated data source. The constructor invocation statement needs to be in the following form:

   ```
   connection-context-class connection-context-object=
     new connection-context-class(Connection JDBC-connection-object);
   ```

   The *JDBC-connection-object* parameter is the `Connection` object that you created in step 3 on page 967.

The following code uses connection technique 3 to create a connection to a location with logical name `jdbc/sampledb`. The numbers to the right of selected statements correspond to the previously-described steps.

```
import java.sql.*;
import javax.naming.*;
import javax.sql.*;
...
#sql context CtxSqlj;       // Create connection context class CtxSqlj   1
Context ctx=new InitialContext();                                         2b
DataSource ds=(DataSource)ctx.lookup("jdbc/sampledb");                    2c
Connection con=ds.getConnection();                                        3
String empname;             // Declare a host variable
...
con.setAutoCommit(false);   // Do not autocommit                          4
CtxSqlj myConnCtx=new CtxSqlj(con);                                       5
                            // Create connection context object myConnCtx
#sql [myConnCtx] {SELECT LASTNAME INTO :empname FROM EMPLOYEE
   WHERE EMPNO='000010'};
                            // Use myConnCtx for executing an SQL statement
```

Figure 23. Using connection technique 3 to connect to a data source

Connection technique 4 (DB2 Universal JDBC Driver only): This technique uses the JDBC `DataSource` interface for creating the connection. This technique **requires** that the `DataSource` is registered with JNDI.

1. From your system administrator, obtain the logical name of the data source to which you need to connect.
2. Execute an SQLJ connection declaration clause.

   For this type of connection, the connection declaration clause needs to be of this form:

   ```
   #sql public static context context-class-name
    with (dataSource="logical-name");
   ```

   The connection context must be declared as public and static. *logical-name* is the data source name that you obtained in step 1.
3. Invoke the constructor for the connection context class that you created in step 2.

   Doing this creates a connection context object that you specify in each SQL statement that you execute at the associated data source. The constructor invocation statement needs to be in one of the following forms:

```
connection-context-class connection-context-object=
  new connection-context-class();

connection-context-class connection-context-object=
  new connection-context-class (String user,
    String password);
```

The meanings of the *user* and *password* parameters are the same as the meanings of the parameters in step 3 on page 965 of connection technique 1.

The following code uses connection technique 4 to create a connection to a location with logical name `jdbc/sampledb`. The connection requires a user ID and password.

```
#sql public static context Ctx
  with (dataSource="jdbc/sampledb");                                  2
                                // Create connection context class Ctx
String userid="dbadm";          // Declare variables for user ID and password
String password="dbadm";

String empname;                 // Declare a host variable
...
Ctx myConnCtx=new Ctx(userid, password);                              3
                                // Create connection context object myConnCtx
                                // for the connection to jdbc/sampledb
#sql [myConnCtx] {SELECT LASTNAME INTO :empname FROM EMPLOYEE
   WHERE EMPNO='000010'};
                                // Use myConnCtx for executing an SQL statement
```

Figure 24. Using connection technique 4 to connect to a data source

Connection technique 5: This technique uses the default connection to connect to the data source. You use the default connection by specifying your SQL statements without a connection context object. When you use this technique, you do not need to load a JDBC driver unless you explicitly use JDBC interfaces in your program. For example:

```
#sql {SELECT LASTNAME INTO :empname FROM EMPLOYEE
   WHERE EMPNO='000010'};  // Use default connection for
                                      // executing an SQL statement
```

To create a default connection context, SQLJ does a JNDI lookup for `jdbc/defaultDataSource`. If nothing is registered, a null context exception is issued when SQLJ attempts to access the context.

Related concepts:

- "How JDBC applications connect to a data source" on page 971

Related tasks:

- "Connecting to a data source using the DriverManager interface with the DB2 Universal JDBC Driver" on page 986
- "Connecting to a data source using the DataSource interface" on page 972

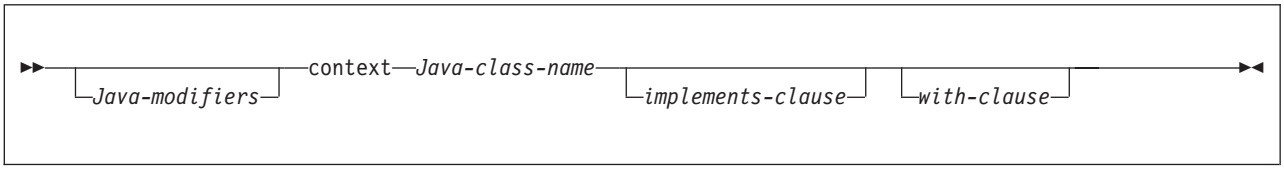
Related reference:

- "Properties for the DB2 Universal JDBC Driver" in the *Application Development Guide: Programming Client Applications*

SQLJ connection-declaration-clause

The connection declaration clause declares a connection to a data source in an SQLJ application program.

Syntax:



Description:

Java-modifiers
Specifies modifiers that are valid for Java class declarations, such as static, public, private, or protected.

Java-class-name
Specifies a valid Java identifier. During the program preparation process, SQLJ generates a connection context class whose name is this identifier.

implements-clause
See SQLJ implements-clause for a description of this clause. In a connection declaration clause, the interface class to which the implements clause refers must be a user-defined interface class.

with-clause
See SQLJ with-clause for a description of this clause.

Usage notes:

- SQLJ generates a connection class declaration for each connection declaration clause you specify. SQLJ data source connections are objects of those generated connection classes.
- You can specify a connection declaration clause anywhere that a Java class definition can appear in a Java program.

Related tasks:

- "Connecting to a data source using SQLJ" on page 965

Related reference:

- "SQLJ implements-clause" in the *Application Development Guide: Programming Client Applications*
- "SQLJ with-clause" in the *Application Development Guide: Programming Client Applications*

Closing the connection to a data source in an SQLJ application

When you have finished with a connection to a data source, you need to close the connection to the data source. Doing so releases the connection context object's DB2® and SQLJ resources immediately.

To close the connection to the data source, use the `ConnectionContext.close()` method. This closes the connection context, as well as the connection to the data source. For example:

```
...
ctx = new EzSqljctx(con0);          // Create a connection context object
                                    // from JDBC connection con0
...                                 // Perform various SQL operations
 EzSqljctx.close();                 // Close the connection context and
                                    // connection to the data source
```

Related tasks:

- "Connecting to a data source using SQLJ" on page 965

JDBC Considerations

How JDBC applications connect to a data source

Before you can execute SQL statements in any SQL program, you must connect to a database server. In JDBC, a database server is known as a *data source*.

Figure 25 shows how a Java™ application connects to a data source for a type 2 driver or DB2 Universal JDBC Driver type 2 connectivity.

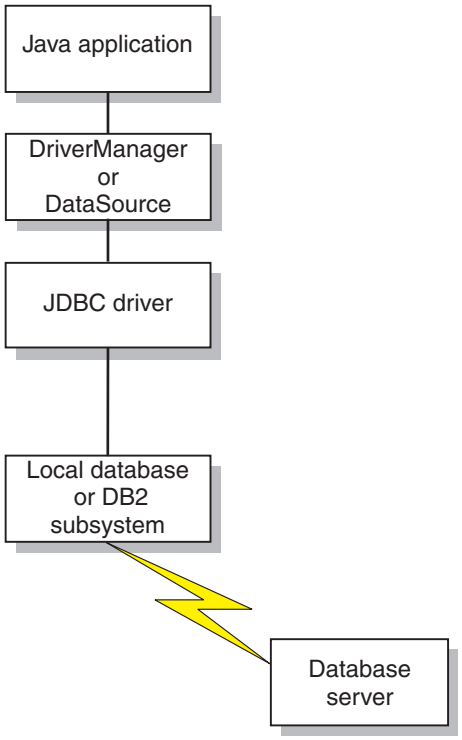
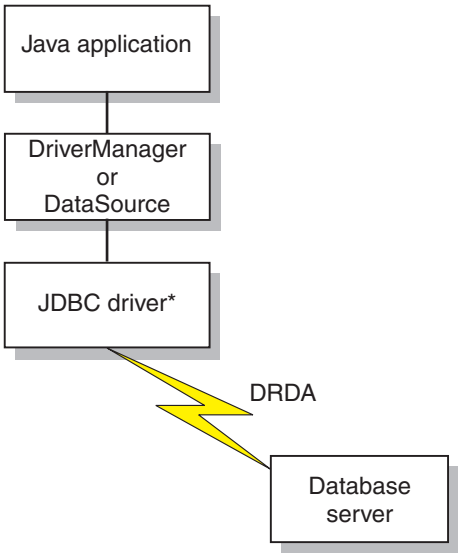


Figure 25. Java application flow for a type 2 driver or DB2 Universal JDBC Driver type 2 connectivity

Figure 26 on page 972 shows how a Java application connects to a data source for DB2 Universal JDBC Driver type 4 connectivity.



*Java byte code executed under JVM

Figure 26. Java application flow for DB2 Universal JDBC Driver type 4 connectivity

The way that you connect to a data source depends on the version of JDBC that you use. Connecting using the `DriverManager` interface is available for all levels of JDBC. Connecting using the `DataSource` interface is available with JDBC 2.0 and above.

Related concepts:

- "How DB2 applications connect to a data source using the DriverManager interface with the DB2 JDBC Type 2 Driver" on page 977

Related tasks:

- "Connecting to a data source using the DataSource interface" on page 972
- "Connecting to a data source using the DriverManager interface with the DB2 Universal JDBC Driver" on page 986

Connecting to a data source using the DataSource interface

Using `DriverManager` to connect to a data source reduces portability because the application must identify a specific JDBC driver class name and driver URL. The driver class name and driver URL are specific to a JDBC vendor, driver implementation, and data source. If your applications need to be portable among data sources, you should use the `DataSource` interface.

When you connect to a data source using the `DataSource` interface, you use a `DataSource` object. It is possible to create and use the `DataSource` object in the same application, as you do with the `DriverManager` interface. Figure 27 shows an example for the DB2 Universal JDBC Driver:

Figure 27. Creating and using a DataSource object in the same application

```
import java.sql.*;        // JDBC base
import javax.sql.*;       // JDBC 2.0 standard extension APIs
import com.ibm.db2.jcc.*; // DB2® Universal JDBC Driver   1
                          // interfaces
DB2SimpleDataSource db2ds=new DB2SimpleDataSource();     2
```

```
db2ds.setDatabaseName("db2loc1");                          3
                        // Assign the location name
db2ds.setDescription("Our Sample Database");
                        // Description for documentation
db2ds.setUser("john");
                        // Assign the user ID
db2ds.setPassword("db2");
                        // Assign the password
Connection con=db2ds.getConnection();                      4
                        // Create a Connection object
```

1 Import the package that contains the implementation of the DataSource interface.

2 Creates a `DB2DataSource` object. `DB2DataSource` is one of the DB2 implementations of the `DataSource` interface. See Create and deploy DataSource objects for information on DB2's `DataSource` implementations.

3 The `setDatabaseName`, `setDescription`, `setUser`, and `setPassword` methods assign attributes to the DB2DataSource object. See Properties for the DB2 Universal JDBC Driver for information about the attributes that you can set for a `DB2DataSource` object under the DB2 Universal JDBC Driver.

4 Establishes a connection to the data source that `DB2DataSource` object db2ds represents.

However, a more flexible way to use a `DataSource` object is for your system administrator to create and manage it separately, using WebSphere® or some other tool. The program that creates and manages a `DataSource` object also uses the Java™ Naming and Directory Interface (JNDI) to assign a logical name to the `DataSource` object. The JDBC application that uses the `DataSource` object can then refer to the object by its logical name, and does not need any information about the underlying data source. In addition, your system administrator can modify the data source attributes, and you do not need to change your application program.

To learn more about using WebSphere to deploy DataSource objects, go to this URL on the Web:

`http://www.ibm.com/software/webservers/appserv/`

To learn about deploying DataSource objects yourself, see Create and deploy DataSource objects.

You can use the `DataSource` interface and the `DriverManager` interface in the same application, but for maximum portability, it is recommended that you use only the `DataSource` interface to obtain connections.

The remainder of this topic explains how to create a connection using a `DataSource` object, given that the system administrator has already created the object and assigned a logical name to it.

To obtain a connection using a `DataSource` object, you need to follow these steps:

1. From your system administrator, obtain the logical name of the data source to which you need to connect.
2. Create a `Context` object to use in the next step. The `Context` interface is part of the Java Naming and Directory Interface (JNDI), not JDBC.
3. In your application program, use JNDI to get the `DataSource` object that is associated with the logical data source name.
4. Use the `DataSource.getConnection` method to obtain the connection.

   You can use one of the following forms of the `getConnection` method:

   ```
   getConnection();
   getConnection(String user, String password);
   ```

Use the second form if you need to specify a user ID and password for the connection that are different from the ones that were specified when the `DataSource` was deployed.

Figure 28 shows an example of the code that you need in your application program to obtain a connection using a `DataSource` object, given that the logical name of the data source that you need to connect to is `jdbc/sampledb`. The numbers to the right of selected statements correspond to the previously-described steps.

Figure 28. Obtaining a connection using a DataSource object

```
import java.sql.*;
import javax.naming.*;
import javax.sql.*;
...
Context ctx=new InitialContext();                          2
DataSource ds=(DataSource)ctx.lookup("jdbc/sampledb");     3
Connection con=ds.getConnection();                         4
```

Related tasks:

- "Creating and deploying DataSource objects" in the *Application Development Guide: Programming Client Applications*

Related reference:

- "Properties for the DB2 Universal JDBC Driver" in the *Application Development Guide: Programming Client Applications*

JDBC connection objects

When you connect to a data source by either connection method, you create a `Connection` object, which represents the connection to the data source. You use this `Connection` object to do the following things:

- Create `Statement`, `PreparedStatement`, and `CallableStatement` objects for executing SQL statements. These are discussed in Execute SQL in a JDBC application.
- Gather information about the data source to which you are connected. This process is discussed in Use DatabaseMetaData to learn about a data source.
- Commit or roll back transactions. You can commit transactions manually or automatically. These operations are discussed in Commit or roll back a JDBC transaction.
- Close the connection to the data source. This operation is discussed in Close a connection to a JDBC data source.

Related concepts:

- "JDBC interfaces for executing SQL" in the *Application Development Guide: Programming Client Applications*

Related tasks:

- "Closing a connection to a JDBC data source" on page 975
- "Committing or rolling back JDBC transactions" on page 975
- "Learning about a data source using DatabaseMetaData methods" in the *Application Development Guide: Programming Client Applications*

Committing or rolling back JDBC transactions

In JDBC, to commit or roll back transactions explicitly, use the `commit` or `rollback` methods. For example:

```
Connection con;
...
con.commit();
```

If autocommit mode is on, DB2® performs a commit operation after every SQL statement completes. To determine whether autocommit mode is on, invoke the `Connection.getAutoCommit` method. To set autocommit mode on, invoke the `Connection.setAutoCommit(true)` method. To set autocommit mode off, invoke the `Connection.setAutoCommit(false)` method.

Related concepts:

- "Savepoints in JDBC applications" in the *Application Development Guide: Programming Client Applications*

Related tasks:

- "Making batch updates in JDBC applications" in the *Application Development Guide: Programming Client Applications*
- "Closing a connection to a JDBC data source" on page 975

Closing a connection to a JDBC data source

When you have finished with a connection to a data source, it is *essential* that you close the connection to the data source. Doing this releases the `Connection` object's DB2® and JDBC resources immediately. To close the connection to the data source, use the `close` method. For example:

```
Connection con;
...
con.close();
```

If autocommit mode is not on, the connection needs to be on a unit-of-work boundary before you close the connection.

Related concepts:

- "How JDBC applications connect to a data source" on page 971

Type 2 JDBC Driver Considerations

Security under the DB2 JDBC Type 2 Driver

The DB2® JDBC Type 2 Driver for Linux, UNIX® and Windows® (DB2 JDBC Type 2 Driver) supports user ID and password security. You must set the user ID and the password, or set neither. If you do not set a user ID and password, the driver uses the user ID and password of the user who is currently logged on to the operating system.

To specify user ID and password security for a JDBC connection, use one of the following techniques.

For the `DriverManager` interface: you can specify the user ID and password directly in the `DriverManager.getConnection` invocation. For example:

```
import java.sql.*;        // JDBC base
...
String id = "db2adm";      // Set user ID
Sring pw = "db2adm";       // Set password
String url = "jdbc:db2:toronto";
                           // Set URL for the data source
Connection con = DriverManager.getConnection(url, id, pw);
                           // Create connection
```

Alternatively, you can set the user ID and password by setting the `user` and `password` properties in a `Properties` object, and then invoking the form of the `getConnection` method that includes the `Properties` object as a parameter. For example:

```
import java.sql.*;                          // JDBC base
import COM.ibm.db2.jdbc.*;                  // DB2 implementation of JDBC 2.0
...
Properties properties = new java.util.Properties();
                                            // Create Properties object
properties.put("user", "db2adm");           // Set user ID for the connection
properties.put("password", "db2adm");       // Set password for the connection
String url = "jdbc:db2:toronto";
                                            // Set URL for the data source
Connection con = DriverManager.getConnection(url, properties);
                                            // Create connection
```

For the `DataSource` interface: you can specify the user ID and password directly in the `DataSource.getConnection` invocation. For example:

```
import java.sql.*;                          // JDBC base
import COM.ibm.db2.jdbc.*;                  // DB2 implementation of JDBC 2.0
...
Context ctx=new InitialContext();           // Create context for JNDI
DataSource ds=(DataSource)ctx.lookup("jdbc/sampledb");
                                            // Get DataSource object
String id = "db2adm";                       // Set user ID
Sring pw = "db2adm";                        // Set password
Connection con = ds.getConnection(id, pw);
                                            // Create connection
```

Alternatively, if you create and deploy the `DataSource` object, you can set the user ID and password by invoking the `DataSource.setUser` and `DataSource.setPassword` methods after you create the `DataSource` object. For example:

```
import java.sql.*;                          // JDBC base
import COM.ibm.db2.jdbc.*;                  // DB2 implementation of JDBC 2.0
...
DB2DataSource db2ds = new DB2DataSource();
                                              // Create DataSource object
db2ds.setDatabaseName("toronto");             // Set location
db2ds.setUser("db2adm");                      // Set user ID
db2ds.setPassword("db2adm");                  // Set password
```

Related concepts:

- "How DB2 applications connect to a data source using the DriverManager interface with the DB2 JDBC Type 2 Driver" on page 977

Related tasks:

- "Connecting to a data source using the DataSource interface" on page 972
- "Creating and deploying DataSource objects" in the *Application Development Guide: Programming Client Applications*

How DB2 applications connect to a data source using the DriverManager interface with the DB2 JDBC Type 2 Driver

A JDBC application can establish a connection to a data source using the JDBC `DriverManager` interface, which is part of the `java.sql` package.

The Java™ application first loads the JDBC driver by invoking the `Class.forName` method. After the application loads the driver, it connects to a database server by invoking the `DriverManager.getConnection` method.

For the DB2® JDBC Type 2 Driver for Linux, UNIX® and Windows® (DB2 JDBC Type 2 Driver), you load the driver by invoking the `Class.forName` method with the following argument:

```
COM.ibm.db2.jdbc.app.DB2Driver
```

The following code demonstrates loading the DB2 JDBC Type 2 Driver:

```
try {
  // Load the DB2 JDBC Type 2 Driver with DriverManager
  Class.forName("COM.ibm.db2.jdbc.app.DB2Driver");
} catch (ClassNotFoundException e) {
     e.printStackTrace();
}
```

The `catch` block is used to print an error if the driver is not found.

After you load the driver, you connect to the data source by invoking the `DriverManager.getConnection` method. You can use one of the following forms of `getConnection`:

```
getConnection(String url);
getConnection(String url, user, password);
getConnection(String url, java.util.Properties info);
```

The *url* argument represents a data source.

For the DB2 JDBC Type 2 Driver, specify a URL of the following form:

Syntax for a URL for the DB2 JDBC Type 2 Driver:

The parts of the URL have the following meanings:

jdbc:db2:
jdbc:db2: indicates that the connection is to a DB2 UDB server.

database
A database alias. The alias refers to the DB2 database catalog entry on the DB2 client.

The *info* argument is an object of type `java.util.Properties` that contains a set of driver properties for the connection. Specifying the *info* argument is an alternative to specifying *property=value* strings in the URL.

Specifying a user ID and password for a connection: There are several ways to specify a user ID and password for a connection:

- Use the form of the `getConnection` method that specifies *user* and *password*.

- Use the form of the `getConnection` method that specifies *info*, after setting the user and password properties in a `java.util.Properties` object.

Example: Setting the user ID and password in user and password parameters:

```
String url = "jdbc:db2:toronto";
                                        // Set URL for data source
String user = "db2adm";
String password = "db2adm";
Connection con = DriverManager.getConnection(url, user, password);
                                        // Create connection
```

Example: Setting the user ID and password in a `java.util.Properties` object:

```
Properties properties = new Properties(); // Create Properties object
properties.put("user", "db2adm");         // Set user ID for connection
properties.put("password", "db2adm");     // Set password for connection
String url = "jdbc:db2:toronto";
                                        // Set URL for data source
Connection con = DriverManager.getConnection(url, properties);
                                        // Create connection
```

Related concepts:

- "Security under the DB2 JDBC Type 2 Driver" on page 975

Universal JDBC Driver Considerations

User ID and password security under the DB2 Universal JDBC Driver

To specify user ID and password security for a JDBC connection, use one of the following techniques.

For the `DriverManager` interface: You can specify the user ID and password directly in the `DriverManager.getConnection` invocation. For example:

```
import java.sql.*;          // JDBC base
...
String id = "db2adm";       // Set user ID
String pw = "db2adm";       // Set password
String url = "jdbc:db2://mvs1.sj.ibm.com:5021/san_jose";
                            // Set URL for the data source
Connection con = DriverManager.getConnection(url, id, pw);
                            // Create connection
```

Another method is to set the user ID and password directly in the URL string. For example:

```
import java.sql.*;          // JDBC base
...
String url =
  "jdbc:db2://mvs1.sj.ibm.com:5021/san_jose:user=db2adm;password=db2adm;";
                            // Set URL for the data source
Connection con = DriverManager.getConnection(url);
                            // Create connection
```

Alternatively, you can set the user ID and password by setting the `user` and `password` properties in a `Properties` object, and then invoking the form of the `getConnection` method that includes the `Properties` object as a parameter. Optionally, you can set the `securityMechanism` property to indicate that you are using user ID and password security. For example:

```
import java.sql.*;                          // JDBC base
import com.ibm.db2.jcc.*;                   // DB2® implementation of JDBC 2.0
...
Properties properties = new java.util.Properties();
                                            // Create Properties object
properties.put("user", "db2adm");           // Set user ID for the connection
properties.put("password", "db2adm");       // Set password for the connection
properties.put("securityMechanism",
  new String("" + com.ibm.db2.jcc.DB2BaseDataSource.CLEAR_TEXT_PASSWORD_SECURITY +
  ""));
                                            // Set security mechanism to
                                            // user ID and password
String url = "jdbc:db2://mvs1.sj.ibm.com:5021/san_jose";
                                            // Set URL for the data source
Connection con = DriverManager.getConnection(url, properties);
                                            // Create connection
```

For the DataSource interface: you can specify the user ID and password directly in the `DataSource.getConnection` invocation. For example:

```
import java.sql.*;                          // JDBC base
import com.ibm.db2.jcc.*;                   // DB2 implementation of JDBC 2.0
...
Context ctx=new InitialContext();           // Create context for JNDI
DataSource ds=(DataSource)ctx.lookup("jdbc/sampledb");
                                            // Get DataSource object
String id = "db2adm";                       // Set user ID
String pw = "db2adm";                        // Set password
Connection con = ds.getConnection(id, pw);
                                            // Create connection
```

Alternatively, if you create and deploy the `DataSource` object, you can set the user ID and password by invoking the `DataSource.setUser` and `DataSource.setPassword` methods after you create the `DataSource` object. Optionally, you can invoke the `DataSource.setSecurityMechanism` method property to indicate that you are using user ID and password security. For example:

```
...
com.ibm.db2.jcc.DB2SimpleDataSource db2ds =    // Create DB2SimpleDataSource object
  new com.ibm.db2.jcc.DB2SimpleDataSource();
db2ds.setDriverType(4);                         // Set driver type
db2ds.setDatabaseName("san_jose");              // Set location
db2ds.setServerName("mvs1.sj.ibm.com");         // Set server name
db2ds.setPortNumber(5021);                      // Set port number
db2ds.setUser("db2adm");                        // Set user ID
db2ds.setPassword("db2adm");                    // Set password
db2ds.setSecurityMechanism(
  com.ibm.db2.jcc.DB2BaseDataSource.CLEAR_TEXT_PASSWORD_SECURITY);
                                                // Set security mechanism to
                                                // user ID and password
```

Related tasks:

- "Connecting to a data source using the DataSource interface" on page 972
- "Creating and deploying DataSource objects" in the *Application Development Guide: Programming Client Applications*
- "Connecting to a data source using the DriverManager interface with the DB2 Universal JDBC Driver" on page 986

Related reference:

- "Properties for the DB2 Universal JDBC Driver" in the *Application Development Guide: Programming Client Applications*

User ID-only security under the DB2 Universal JDBC Driver

To specify user ID security for a JDBC connection, use one of the following techniques.

For the `DriverManager` interface: Set the user ID and security mechanism by setting the `user` and `securityMechanism` properties in a `Properties` object, and then invoking the form of the `getConnection` method that includes the `Properties` object as a parameter. For example:

```
import java.sql.*;                          // JDBC base
import com.ibm.db2.jcc.*;                   // DB2® implementation of JDBC 2.0
...
Properties properties = new Properties(); // Create a Properties object
properties.put("user", "db2adm");         // Set user ID for the connection
properties.put("securityMechanism",
  new String("" + com.ibm.db2.jcc.DB2BaseDataSource.USER_ONLY_SECURITY + ""));
                                          // Set security mechanism to
                                          // user ID only
String url = "jdbc:db2://mvs1.sj.ibm.com:5021/san_jose";
                                          // Set URL for the data source
Connection con = DriverManager.getConnection(url, properties);
                                          // Create the connection
```

For the `DataSource` interface: If you create and deploy the `DataSource` object, you can set the user ID and security mechanism by invoking the `DataSource.setUser` and `DataSource.setSecurityMechanism` methods after you create the `DataSource` object. For example:

```
import java.sql.*;                        // JDBC base
import com.ibm.db2.jcc.*;                 // DB2 implementation of JDBC 2.0
...
com.ibm.db2.jcc.DB2SimpleDataSource db2ds =
  new com.ibm.db2.jcc.DB2SimpleDataSource();
                                          // Create DB2SimpleDataSource object
db2ds.setDriverType(4);                   // Set the driver type
db2ds.setDatabaseName("san_jose");        // Set the location
db2ds.setServerName("mvs1.sj.ibm.com");   // Set the server name
db2ds.setPortNumber(5021);                // Set the port number
db2ds.setUser("db2adm");                  // Set the user ID
db2ds.setSecurityMechanism(
  com.ibm.db2.jcc.DB2BaseDataSource.USER_ONLY_SECURITY);
                                          // Set security mechanism to
                                          // user ID only
```

Kerberos security under the DB2 Universal JDBC Driver

Kerberos security is available for Universal Type 4 Connectivity only.

If you use Kerberos security when you access a DB2® for z/OS™ server, you need to install and configure the following products, or their equivalents:

- The SecureWay® Security Server for z/OS and OS/390®
- OS/390 SecureWay Security Server Network Authentication and Privacy Service, which is a component of the OS/390 SecureWay Security Server

  This is the IBM® OS/390 implementation of Kerberos Version 5.

  For more information, see *OS/390 SecureWay Server Network Authentication and Privacy Service Administration*.

You also need to enable the following components of the IBM Developer Kit for OS/390, Java™ 2 Technology Edition, or the IBM Developer Kit for z/OS, Java 2 Technology Edition:

- Java Cryptography Extension (IBMJCE) for OS/390
- IBM Java Generic Security Service (IBMJGSS)
- Java Authentication and Authorization Service (JAAS) for OS/390

For information on how to enable these components, go to this URL on the Web:
`http://www.ibm.com/servers/eserver/zseries/software/java/aboutj2.html`

There are three ways to specify Kerberos security for a connection:
- With a user ID and password
- Without a user ID or password
- With a delegated credential

Using Kerberos security with a user ID and password:

For this case, Kerberos uses the specified user ID and password to obtain a ticket-granting ticket (TGT) that lets you authenticate to the DB2 server.

You need to set the `user`, `password`, `kerberosServerPrincipal`, and `securityMechanism` properties. The `kerberosServerPrincipal` property specifies the address of the Kerberos server for the realm in which the client is registered.

For the `DriverManager` interface: Set the user ID, password, Kerberos server, and security mechanism by setting the `user`, `password`, `kerberosServerPrincipal`, and `securityMechanism` properties in a `Properties` object, and then invoking the form of the `getConnection` method that includes the `Properties` object as a parameter. For example, use code like this to set the Kerberos security mechanism with a user ID and password:

```
import java.sql.*;                          // JDBC base
import com.ibm.db2.jcc.*;                   // DB2 implementation of JDBC 2.0
...
Properties properties = new Properties(); // Create a Properties object
properties.put("user", "db2adm");         // Set user ID for the connection
properties.put("password", "db2adm");     // Set password for the connection
properties.put("kerberosServerPrincipal", "kdcsrv1.sj.ibm.com");
                                          // Set the Kerberos server
properties.put("securityMechanism",
  new String("" + com.ibm.db2.jcc.DB2BaseDataSource.KERBEROS_SECURITY + ""));
                                          // Set security mechanism to
                                          // Kerberos
String url = "jdbc:db2://mvs1.sj.ibm.com:5021/san_jose";
                                          // Set URL for the data source
Connection con = DriverManager.getConnection(url, properties);
                                          // Create the connection
```

For the `DataSource` interface: If you create and deploy the `DataSource` object, set the Kerberos server and security mechanism by invoking the `DataSource.setKerberosServerPrincipal` and `DataSource.setSecurityMechanism` methods after you create the `DataSource` object. For example:

```
import java.sql.*;                          // JDBC base
import com.ibm.db2.jcc.*;                   // DB2 implementation of JDBC 2.0
...
com.ibm.db2.jcc.DB2SimpleDataSource db2ds =
  new com.ibm.db2.jcc.DB2SimpleDataSource();
                                          // Create the DataSource object
db2ds.setDriverType(4);                   // Set the driver type
db2ds.setDatabaseName("san_jose");        // Set the location
db2ds.setUser("db2adm");                  // Set the user
db2ds.setPassword("db2adm");              // Set the password
db2ds.setServerName("mvs1.sj.ibm.com");
                                          // Set the server name
```

```
db2ds.setPortNumber(5021);                // Set the port number
db2ds.setKerberosServerPrincipal("kdcsrv1.sj.ibm.com");
                                          // Set the Kerberos server
db2ds.setSecurityMechanism(
  com.ibm.db2.jcc.DB2BaseDataSource.KERBEROS_SECURITY);
                                          // Set security mechanism to
                                          // Kerberos
```

Using Kerberos security with no user ID or password:

For this case, the Kerberos default credentials cache must contain a ticket-granting ticket (TGT) that lets you authenticate to the DB2 server.

You need to set the `kerberosServerPrincipal` and `securityMechanism` properties.

For the `DriverManager` interface: Set the Kerberos server and security mechanism by setting the `kerberosServerPrincipal` and `securityMechanism` properties in a `Properties` object, and then invoking the form of the `getConnection` method that includes the `Properties` object as a parameter. For example, use code like this to set the Kerberos security mechanism without a user ID and password:

```
import java.sql.*;                        // JDBC base
import com.ibm.db2.jcc.*;                 // DB2 implementation of JDBC 2.0
...
Properties properties = new Properties(); // Create a Properties object
properties.put("kerberosServerPrincipal", "kdcsrv1.sj.ibm.com");
                                          // Set the Kerberos server
properties.put("securityMechanism",
  new String("" + com.ibm.db2.jcc.DB2BaseDataSource.KERBEROS_SECURITY + ""));
                                          // Set security mechanism to
                                          // Kerberos
String url = "jdbc:db2://mvs1.sj.ibm.com:5021/san_jose";
                                          // Set URL for the data source
Connection con = DriverManager.getConnection(url, properties);
                                          // Create the connection
```

For the `DataSource` interface: If you create and deploy the `DataSource` object, set the Kerberos server and security mechanism by invoking the `DataSource.setKerberosServerPrincipal` and `DataSource.setSecurityMechanism` methods after you create the `DataSource` object. For example:

```
import java.sql.*;                        // JDBC base
import com.ibm.db2.jcc.*;                 // DB2 implementation of JDBC 2.0
...
DB2DataSource db2ds = new com.ibm.db2.jcc.DB2SimpleDataSource();
                                          // Create the DataSource object
db2ds.setDriverType(4);                   // Set the driver type
db2ds.setDatabaseName("san_jose");        // Set the location
db2ds.setServerName("mvs1.sj.ibm.com");
                                          // Set the server name
db2ds.setPortNumber(5021);                // Set the port number
db2ds.setKerberosServerPrincipal("kdcsrv1.sj.ibm.com");
                                          // Set the Kerberos server
db2ds.setSecurityMechanism(
  com.ibm.db2.jcc.DB2BaseDataSource.KERBEROS_SECURITY);
                                          // Set security mechanism to
                                          // Kerberos
```

Using Kerberos security with a delegated credential from another principal:

For this case, you authenticate to the DB2 server using a delegated credential that another principal passes to you.

You need to set the kerberosServerPrincipal, gssCredential, and securityMechanism properties.

For the DriverManager interface: Set the Kerberos server, delegated credential, and security mechanism by setting the kerberosServerPrincipal, and securityMechanism properties in a Properties object. Because the gssCredential property is not a string, you cannot use the Properties.put method to set it. Instead, use the DB2BaseDataSource.setGSSCredential method. Then invoke the form of the getConnection method that includes the Properties object as a parameter. For example, use code like this to set the Kerberos security mechanism without a user ID and password:

```
import java.sql.*;                          // JDBC base
import com.ibm.db2.jcc.*;                   // DB2 implementation of JDBC 2.0
...
Properties properties = new Properties(); // Create a Properties object
properties.put("kerberosServerPrincipal", "kdcsrv1.sj.ibm.com");
                                            // Set the Kerberos server
properties.put("gssCredential",delegatedCredential);
                                            // Set the delegated credential
properties.put("securityMechanism",
  new String("" + com.ibm.db2.jcc.DB2BaseDataSource.KERBEROS_SECURITY + ""));

                                            // Set security mechanism to
                                            // Kerberos
String url = "jdbc:db2://mvs1.sj.ibm.com:5021/san_jose";
                                            // Set URL for the data source
Connection con = DriverManager.getConnection(url, properties);
                                            // Create the connection
```

For the DataSource interface: If you create and deploy the DataSource object, set the Kerberos server, delegated credential, and security mechanism by invoking the DataSource.setKerberosServerPrincipal, DataSource.setGssCredential, and DataSource.setSecurityMechanism methods after you create the DataSource object. For example:

```
DB2DataSource db2ds = new com.ibm.db2.jcc.DB2SimpleDataSource();
                                               // Create the DataSource object
db2ds.setDriverType(4);                        // Set the driver type
db2ds.setDatabaseName("san_jose");             // Set the location
db2ds.setServerName("mvs1.sj.ibm.com");  // Set the server name
db2ds.setPortNumber(5021);                     // Set the port number
db2ds.setKerberosServerPrincipal("kdcsrv1.sj.ibm.com");
                                               // Set the Kerberos server
db2ds.setGssCredential(delegatedCredential);
                                               // Set the delegated credential
db2ds.setSecurityMechanism(
  com.ibm.db2.jcc.DB2BaseDataSource.KERBEROS_SECURITY);
                                               // Set security mechanism to
                                               // Kerberos
```

Related tasks:

- "Connecting to a data source using the DataSource interface" on page 972
- "Creating and deploying DataSource objects" in the *Application Development Guide: Programming Client Applications*
- "Connecting to a data source using the DriverManager interface with the DB2 Universal JDBC Driver" on page 986

Related reference:

- "Properties for the DB2 Universal JDBC Driver" in the *Application Development Guide: Programming Client Applications*

Encrypted user ID security or encrypted password security under the DB2 Universal JDBC Driver

If you use encrypted user ID security or encrypted password security when you access a DB2® for z/OS™ server, the Java™ Cryptography Extension, IBMJCE for z/OS needs to be enabled on the server. The Java Cryptography Extension is part of the IBM® Developer Kit for OS/390®, Java 2 Technology Edition, or the IBM Developer Kit for z/OS, Java 2 Technology Edition. For information on how to enable IBMJCE, go to this URL on the Web:

`http://www.ibm.com/servers/eserver/zseries/software/java/aboutj2.html`

To specify encrypted user ID or encrypted password security for a JDBC connection, use one of the following techniques.

For the `DriverManager` interface: Set the user ID, password, and security mechanism by setting the `user`, `password`, and `securityMechanism` properties in a `Properties` object, and then invoking the form of the `getConnection` method that includes the `Properties` object as a parameter. For example, use code like this to set the user ID and encrypted password security mechanism:

```
import java.sql.*;                          // JDBC base
import com.ibm.db2.jcc.*;                   // DB2 implementation of JDBC 2.0
...
Properties properties = new Properties(); // Create a Properties object
properties.put("user", "db2adm");           // Set user ID for the connection
properties.put("password", "db2adm");       // Set password for the connection
properties.put("securityMechanism",
  new String("" + com.ibm.db2.jcc.DB2BaseDataSource.ENCRYPTED_PASSWORD_SECURITY +
  ""));
                                            // Set security mechanism to
                                            // user ID and encrypted password
String url = "jdbc:db2://mvs1.sj.ibm.com:5021/san_jose";
                                            // Set URL for the data source
Connection con = DriverManager.getConnection(url, properties);
                                            // Create the connection
```

For the `DataSource` interface: If you create and deploy the `DataSource` object, you can set the user ID, password, and security mechanism by invoking the `DataSource.setUser`, `DataSource.setPassword`, and `DataSource.setSecurityMechanism` methods after you create the `DataSource` object. For example, use code like this to set the encrypted user ID and encrypted password security mechanism:

```
import java.sql.*;                          // JDBC base
import com.ibm.db2.jcc.*;                   // DB2 implementation of JDBC 2.0
...
com.ibm.db2.jcc.DB2SimpleDataSource db2ds =
  new com.ibm.db2.jcc.DB2SimpleDataSource();
                                            // Create the DataSource object
db2ds.setDriverType(4);                     // Set the driver type
db2ds.setDatabaseName("san_jose");          // Set the location
db2ds.setServerName("mvs1.sj.ibm.com");
                                            // Set the server name
db2ds.setPortNumber(5021);                  // Set the port number
db2ds.setUser("db2adm");                    // Set the user ID
db2ds.setPassword("db2adm");                // Set the password
db2ds.setSecurityMechanism(
  com.ibm.db2.jcc.DB2BaseDataSource.ENCRYPTED_PASSWORD_SECURITY);
                                            // Set security mechanism to
                                            // encrypted user ID and password
```

Related tasks:

- "Connecting to a data source using the DataSource interface" on page 972
- "Creating and deploying DataSource objects" in the *Application Development Guide: Programming Client Applications*
- "Connecting to a data source using the DriverManager interface with the DB2 Universal JDBC Driver" on page 986

Related reference:

- "Properties for the DB2 Universal JDBC Driver" in the *Application Development Guide: Programming Client Applications*

Security under the DB2 Universal JDBC Driver

When you use the DB2 Universal JDBC Driver, you choose a security mechanism by specifying a value for the `securityMechanism` property. You can set this property in one of the following ways:

- If you use the `DriverManager` interface, set `securityMechanism` in a `java.util.Properties` object before you invoke the form of the `getConnection` method that includes the `java.util.Properties` parameter.
- If you use the `DataSource` interface, and you are creating and deploying your own `DataSource` objects, invoke the `DataSource.setSecurityMechanism` method after you create a `DataSource` object.

Table 74 lists the security mechanisms that the DB2 Universal JDBC Driver supports, and the value that you need to specify for the `securityMechanism` property to specify each security mechanism. The default security mechanism is the user ID and password mechanism.

Table 74. Security mechanisms supported by the DB2 Universal JDBC Driver

Security mechanism	`securityMechanism` property value
User ID and password	`DB2BaseDataSource.CLEAR_TEXT_PASSWORD_SECURITY`
User ID only	`DB2BaseDataSource.USER_ONLY_SECURITY`
User ID and encrypted password	`DB2BaseDataSource.ENCRYPTED_PASSWORD_SECURITY`
Encrypted user ID and encrypted password	`DB2BaseDataSource.ENCRYPTED_USER_AND_PASSWORD_SECURITY`
Kerberos[1]	`DB2BaseDataSource.KERBEROS_SECURITY`

Note:

1. Available for Universal Type 4 Connectivity only.

Related concepts:

- "Encrypted user ID security or encrypted password security under the DB2 Universal JDBC Driver" on page 984
- "Kerberos security under the DB2 Universal JDBC Driver" on page 980
- "User ID-only security under the DB2 Universal JDBC Driver" on page 980
- "User ID and password security under the DB2 Universal JDBC Driver" on page 978

Related reference:

- "Properties for the DB2 Universal JDBC Driver" in the *Application Development Guide: Programming Client Applications*

Connecting to a data source using the DriverManager interface with the DB2 Universal JDBC Driver

A JDBC application can establish a connection to a data source using the JDBC `DriverManager` interface, which is part of the `java.sql` package.

The Java™ application first loads the JDBC driver by invoking the `Class.forName` method. After the application loads the driver, it connects to a database server by invoking the `DriverManager.getConnection` method.

For the DB2 Universal JDBC Driver, you load the driver by invoking the `Class.forName` method with the following argument:

```
com.ibm.db2.jcc.DB2Driver
```

For compatibility with previous JDBC drivers, you can use the following argument instead:

```
COM.ibm.db2os390.sqlj.jdbc.DB2SQLJDriver
```

The following code demonstrates loading the DB2 Universal JDBC Driver:

```
try {
  // Load the DB2® Universal JDBC Driver with DriverManager
  Class.forName("com.ibm.db2.jcc.DB2Driver");
} catch (ClassNotFoundException e) {
    e.printStackTrace();
}
```

The `catch` block is used to print an error if the driver is not found.

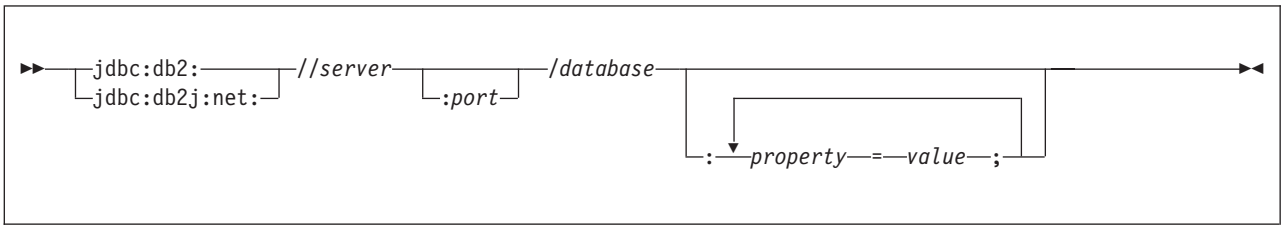
After you load the driver, you connect to the data source by invoking the `DriverManager.getConnection` method. You can use one of the following forms of `getConnection`:

```
getConnection(String url);
getConnection(String url, user, password);
getConnection(String url, java.util.Properties info);
```

The *url* argument represents a data source, and indicates what type of JDBC connectivity you are using.

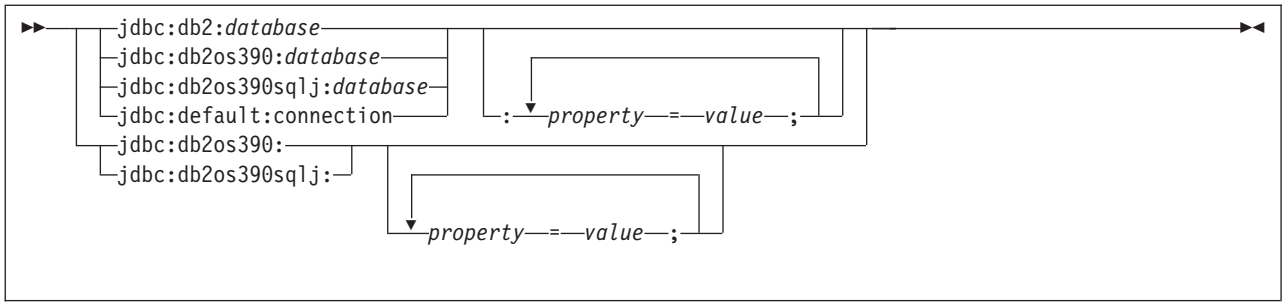
For DB2 Universal JDBC Driver type 4 connectivity, specify a URL of the following form:

Syntax for a URL for Universal Type 4 Connectivity:



For DB2 Universal JDBC Driver type 2 connectivity, specify a URL of one of the following forms:

Syntax for a URL for Universal Type 2 Connectivity:



The parts of the URL have the following meanings:

jdbc:db2: or jdbc:db2j:net:
The meanings of the initial portion of the URL are:

jdbc:db2:
Indicates that the connection is to a server in the DB2 UDB family.

jdbc:db2j:net:
Indicates that the connection is to a remote IBM® Cloudscape™ server.

server
The domain name or IP address of the database server.

port
The TCP/IP server port number that is assigned to the database server. This is an integer between 0 and 65535. The default is 446.

database
A name for the database server. This name depends on whether Universal Type 4 Connectivity or Universal Type 2 Connectivity is used.

For Universal Type 4 Connectivity:

- If the connection is to a DB2 UDB for z/OS server, *database* is the DB2 location name that is defined during installation. All characters in this value must be uppercase characters. You can determine the location name by executing the following SQL statement on the server:
  ```
  SELECT CURRENT SERVER FROM SYSIBM.SYSDUMMY1;
  ```
- If the connection is to a DB2 UDB for z/OS server or a DB2 UDB for iSeries server, all characters in *database* must be uppercase characters.
- If the connection is to a DB2 UDB for Linux, UNIX and Windows server, *database* is the database name that is defined during installation.
- If the connection is to an IBM Cloudscape server, the *database* is the fully-qualified name of the file that contains the database. This name must be enclosed in double quotation marks ("). For example:
  ```
  "c:/databases/testdb"
  ```

For Universal Type 2 Connectivity:

- *database* is the database name that is defined during installation, if the value of the serverName connection property is null. If the value of serverName property is not null, *database* is a database alias.
- If the connection is to a DB2 UDB for z/OS server or a DB2 UDB for iSeries server, all characters in *database* must be uppercase characters.

property=value;
A property for the JDBC connection. For the definitions of these properties, see Properties for the DB2 Universal JDBC Driver.

The *info* argument is an object of type `java.util.Properties` that contains a set of driver properties for the connection. Specifying the *info* argument is an alternative to specifying *property=value* strings in the URL. See Properties for the DB2 Universal JDBC Driver for the properties that you can specify.

Specifying a user ID and password for a connection: There are several ways to specify a user ID and password for a connection:

- Use the form of the `getConnection` method that specifies *url* with `property=value;` clauses, and include the user and password properties in the URL.
- Use the form of the `getConnection` method that specifies *user* and *password*.
- Use the form of the `getConnection` method that specifies *info*, after setting the user and password properties in a `java.util.Properties` object.

Example: Setting the user ID and password in a URL:

```
String url = "jdbc:db2://sysmvs1.stl.ibm.com:5021/san_jose:" +
  "user=db2adm;password=db2adm;";
                                        // Set URL for data source
Connection con = DriverManager.getConnection(url);
                                        // Create connection
```

Example: Setting the user ID and password in user and password parameters:

```
String url = "jdbc:db2://sysmvs1.stl.ibm.com:5021/san_jose";
                                        // Set URL for data source
String user = "db2adm";
String password = "db2adm";
Connection con = DriverManager.getConnection(url, user, password);
                                        // Create connection
```

Example: Setting the user ID and password in a `java.util.Properties` object:

```
Properties properties = new Properties(); // Create Properties object
properties.put("user", "db2adm");         // Set user ID for connection
properties.put("password", "db2adm");     // Set password for connection
String url = "jdbc:db2://sysmvs1.stl.ibm.com:5021/san_jose";
                                        // Set URL for data source
Connection con = DriverManager.getConnection(url, properties);
                                        // Create connection
```

Related concepts:

- "Security under the DB2 Universal JDBC Driver" on page 985

Related reference:

- "Properties for the DB2 Universal JDBC Driver" in the *Application Development Guide: Programming Client Applications*

Security and Routines

Routines in application development

A routine is a database object that can encapsulates programming and database logic related to a specific task. There are three types of routines: procedures, functions, and methods. Each type of routine provides a different interface for containing logic and database operations that can be used to extend the functionality of an SQL statement or a client application. You should consider the many benefits of creating and using routines when you are developing or updating a database application.

When faced with the task of developing new functionality that will interact with a database, there are two approaches you can choose from. You can add the new logic to a client application, or you can develop a routine, where the new logic will reside on the database server. There are a number of benefits in choosing the latter approach.

Benefits of using routines:

The following benefits can be gained by moving application logic into routines:

Encapsulate application logic
In an environment with numerous client computers, each running a variety of database applications, the effective use of routines can simplify code reuse, code standardization, and code maintenance. For example, if a particular aspect of application behavior needs to be changed in an environment where routines are used, only the affected routine that encapsulates that behavior, will require modification. If routines had not been used in this instance, application logic changes would have been required in each client application.

Enable controlled access to database objects
You can use routines to control access to database objects. A user might not have permission to generally issue a particular SQL statement, however the user can be given permission to invoke routines that contain specific implementations of these statements.

Reduce network traffic
When an application is running on a client computer, each SQL statement is sent separately from the client computer to the server computer and each result is returned separately. This can result in a high degree of network traffic. If a piece of work can be identified that involves heavy database activity and little user interaction, it makes sense to install this piece of work on the server. With this work running on the server, the quantity of network traffic between the client computer and the server computer is reduced. DB2 routines run on the database server in this manner. Using routines is an effective way of reducing network traffic and improving overall client application performance.

Alleviate the processing load on the client
In environments where the performance of a client computer is a concern, routines are a practical means of reducing the dependence on the client computer. After an application invokes a routine, the processing of the routine is done on the database server, thus allowing the application to exploit the power of the database server while relieving the client computer of the processing load.

Allow faster, more efficient execution
Routines are database objects and therefore have a closer relationship with the database manager than client applications do. For some types of routines the performance of SQL statements can be much better than the performance of SQL statements that are executed from a client application. For example, NOT FENCED routines run in the same process as the database manager using shared memory for communication. This makes the routines more proficient in transmitting SQL requests and data, than a client application could ever be that communicates using TCP/IP protocols.

Interoperability of logic implementations
Because code modules are often implemented by different programmers,

each with programming expertise in different programming languages, and because it is generally desirable to reuse code wherever possible to save on development time and costs, DB2® routines are highly interoperable.

- A client application in one programming language can invoke routines that are implemented in a different programming language. For example, C client applications can invoke .NET common language runtime routines.
- A routine can invoke another routine regardless of the routine type or the implementation language of the routine. For example a Java™ procedure (one type of routine) can invoke an SQL scalar function (another type of routine with a different implementation language).
- A routine created in a database server on one operating system can be invoked from a DB2 client running on a different operating system.

There are various kinds of routines that address particular functional needs and various routine implementations. The choice of routine type and implementation can impact the degree to which the above benefits are exhibited. In general, routines are a powerful way of encapsulating logic so that you can extend your SQL, and improve the structure, maintenance, and potentially the performance of your applications.

Related concepts:

- "Procedures" on page 991
- "Routine invocation" in the *Application Development Guide: Programming Server Applications*
- "Supported routine programming languages" in the *Application Development Guide: Programming Server Applications*
- "User-defined scalar functions" on page 992
- "Methods" on page 996
- "User-defined scalar functions" on page 995

Related tasks:

- "Building JDBC routines" in the *Application Development Guide: Building and Running Applications*
- "Building SQLJ routines" in the *Application Development Guide: Building and Running Applications*
- "Building UNIX C routines" in the *Application Development Guide: Building and Running Applications*
- "Building UNIX C++ routines" in the *Application Development Guide: Building and Running Applications*
- "Building IBM COBOL routines on AIX" in the *Application Development Guide: Building and Running Applications*
- "Building UNIX Micro Focus COBOL routines" in the *Application Development Guide: Building and Running Applications*
- "Building C/C++ routines on Windows" in the *Application Development Guide: Building and Running Applications*
- "Building IBM COBOL routines on Windows" in the *Application Development Guide: Building and Running Applications*
- "Building Micro Focus COBOL routines on Windows" in the *Application Development Guide: Building and Running Applications*

- "Writing routines" in the *Application Development Guide: Programming Server Applications*
- "Creating routines" in the *Application Development Guide: Programming Server Applications*
- "Debugging routines" in the *Application Development Guide: Programming Server Applications*

Procedures

A procedure, also called a stored procedure, is a database object created by executing the CREATE PROCEDURE statement that can encapsulates logic and SQL statements. Procedures are used as subroutine extensions to applications, and other database objects that can contain logic.

Features

- Enables the encapsulation of SQL statements, function invocations, and logic elements that formulate a particular subroutine module that can be reused.
- Procedures can be called from client applications, other routines, triggers and dynamic compound statements. Procedures are called using the CALL statement.
- Procedures can return multiple result sets.
- Procedures can contain SQL statements that read or modify table data in both single and multiple partition databases.
- When a procedure is invoked the SQL and logic within a procedure is executed on the server. Data is only transferred between the client and the database server in the procedure call and in the procedure return. If you have a series of SQL statements to execute within a client application, and the application does not need to do any processing in between the statements, then this series of statements would benefit from being included in a procedure.

  Note: If only one SQL statement is invoked in a procedure, the overhead of setting up this invocation outweighs the benefit in network traffic savings.

Limitations

- Procedures are not intended to be called from within elements of an SQL query. Procedures can only be invoked by using the CALL statement where it is supported. Functions can be used to express logic that transforms column values. Although procedures can return result sets, table functions can be used to return a table within the FROM clause of an SQL query.
- Output arguments of procedure calls cannot be directly used by another SQL statement.
- Procedures cannot preserve state between invocations.

Common uses

- To implement application sub-routines that specifically encapsulate the database logic associated with a particular task. For example, a business application for managing employee information could use a procedure to encapsulate the database operations involved in hiring an employee.

  Such a procedure could insert employee information into an employee table, a department table, and a benefits table, calculate the employee's

weekly pay amount based on an input parameter, and return the weekly pay value as one of the output parameters. Another procedure could contain a statistical analysis of data in the employee table and return result sets that contain the results of the analysis. This use of procedures effectively isolates database tasks from non-database tasks within an application.

- Standardize application logic. If multiple applications must similarly access or modify the database, a procedure can provide a single interface for that access or modification. The procedure is available to be used by all of the applications. If the interface must change to accommodate a change in business logic, only the single procedure must be modified.

Supported languages

- SQL
- C/C++
- Java™
- OLE
- COBOL
- .NET common language runtime languages

Note: SQL procedures are supported natively and do not require the installation of a compiler.

Related concepts:

- "Routines in application development" on page 988
- "Procedure parameter modes" in the *Application Development Guide: Programming Server Applications*
- "Procedure result sets" in the *Application Development Guide: Programming Server Applications*
- "SQL Procedural Language (SQL PL) in DB2" in the *Application Development Guide: Programming Server Applications*

Related tasks:

- "Setting up the application development environment" in the *Application Development Guide: Building and Running Applications*
- "Creating SQL procedures from the command line" in the *Application Development Guide: Programming Server Applications*
- "Calling procedures from triggers or SQL routines" in the *Application Development Guide: Programming Server Applications*
- "Calling procedures from applications or external routines" in the *Application Development Guide: Programming Server Applications*

Related reference:

- "CALL statement" in the *SQL Reference, Volume 2*
- "CREATE PROCEDURE" on page 588
- "DB2 supported development software" in the *Application Development Guide: Building and Running Applications*

User-defined scalar functions

Scalar user-defined functions (UDFs) enable you to extend and customize SQL statements. They can be invoked in the same manner as DB2® built-in functions

such as LENGTH and COUNT. That is, they can be referenced in SQL statements wherever an expression is valid. Scalar UDFs accept zero or more typed values as input arguments and return a single value upon each invocation.

SQL scalar user-defined functions:

SQL scalar UDFs enable you to encapsulate SQL statements, built-in functions and other routine references, and a subset of SQL PL statements that can be used to implement some basic database logic. SQL scalar functions can read and modify SQL data. SQL functions give the best performance when they make use of built-in functions and do not contain extremely complex logic. For extremely complex logic, consider implementing an external scalar UDF.

External scalar user-defined functions:

External scalar UDFs have their logic implemented in an external programming language. The logic of the function can access the filesystem, perform system calls or access a network. The execution of the external scalar UDF routine logic, like that of SQL scalar UDFs takes place on the server. External scalar UDFs can read SQL data, but cannot modify SQL data. An external scalar UDF can be repeatedly invoked for a single reference of the function and can maintain state between these invocations by using a scratchpad, which is a memory buffer. This can be powerful if a function requires some initial, but expensive, setup logic. The setup logic can be done on a first invocation that may make use of the scratchpad to store some values that can be accessed or updated in subsequent invocations of the scalar function.

Features of SQL and external scalar UDFs

- Can be referenced as part of an SQL statement anywhere an expression is supported.
- The output of a scalar UDF can be used directly by the invoking SQL statement.
- For external scalar user-defined functions, state can be maintained between the iterative invocations of the function by using a scratchpad.
- Can provide a performance advantage when used in predicates, because they are executed at the server. If a function can be applied to a candidate row at the server, it can often eliminate the row from consideration before transmitting it to the client machine, reducing the amount of data that must be passed from server to client.
- An excellent way to build scalar functions out of existing built-in functions. For example, you can create a complex mathematical formula by re-using the built-in scalar functions along with other logic.

Limitations

- Cannot do transaction management within a scalar UDF. That is, you cannot issue a COMMIT or a ROLLBACK within a scalar UDF.
- Cannot return result sets.
- Scalar UDF's are intended to return a single scalar value per set of inputs.
- External scalar UDF's are not intended to be used for a single invocation. They are designed such that for a single reference to the UDF and a given set of inputs, that the UDF be invoked once per input, and return a single scalar value. On the first invocation, scalar UDFs can be designed to do some setup work, or store some information that can be

accessed in subsequent invocations. SQl scalar UDFs are better suited to functionality that requires a single invocation.

- In a single partition database external scalar UDFs can contain SQL statements. These statements can read data from tables, but cannot modify data in tables. If the database has more than one partition then there must be no SQL statements in an external scalar UDF.

  In serial and in partitioned databases SQL scalar UDFs can contain SQL statements that read data from database tables

Common uses

- Extend the set of DB2 built-in functions.
- Perform logic inside an SQL statement that SQL cannot natively perform.
- Encapsulate a scalar query that is commonly reused as a subquery in SQL statements. For example, given a postal code, search a table for the city where the postal code is found.

Supported languages

- SQL
- C/C++
- Java™
- OLE
- .NET common language runtime languages

Notes:

1. There is a limited capability for creating aggregate functions. Also known as column functions, these functions receive a set of like values (a column of data) and return a single answer. A user-defined aggregate function can only be created if it is sourced upon a built-in aggregate function. For example, if a distinct type `SHOESIZE` exists that is defined with base type INTEGER, you could define a UDF, `AVG(SHOESIZE)`, as an aggregate function sourced on the existing built-in aggregate function, `AVG(INTEGER)`.
2. You can also create UDFs that return a row. These are known as row UDFs and can only be used as a transform function for structured types. The output of a row UDF is a single row.

Related concepts:

- "Routines in application development" on page 988
- "Scratchpads for UDFs and methods" in the *Application Development Guide: Programming Server Applications*

Related tasks:

- "Invoking scalar functions or methods" in the *Application Development Guide: Programming Server Applications*

Related reference:

- "CREATE FUNCTION" on page 574

User-defined scalar functions

Like scalar UDFs, a table UDF enables you to extend and customize SQL, but for the purpose of generating a table. Table UDFs can only be invoked in the FROM clause of an SQL statement. Table UDFs accept zero or more typed values as input arguments and return a table.

Table functions are powerful because they enable you to make almost any source of data appear as a DB2® table. A table function can be easily created by writing a program that collects the desired data, filters it according to some input parameters if so desired, and returns it to the DB2 one row at a time.

Features

- Can be referenced as part of an SQL statement FROM clause.
- External table-functions can make operating system calls, read data from files or even access data across a network in a single partitioned database.
- Results can be directly processed by the SQL statement that references the table function.
- SQL table functions can encapsulate SQL statements that modify SQL table data. (Only SQL table functions have this property)
- For a single table function reference, a table function can be invoked multiple times and maintain state between invocations by using a scratchpad.
- Provides a set of data for processing.

Limitations

- Cannot do transaction management. This means that you cannot execute COMMIT or ROLLBACK statements from within a table function.
- Cannot return result sets.
- Not designed for single invocations.
- Can only be used in a FROM clause.
- External table functions can read SQL data, but cannot modify SQL data. SQL table functions can be used to contain statements that modify SQL data.

Common uses

- Encapsulate a complex, but commonly used subquery.
- Provide a tabular interface to non-relational data. For example, read a spreadsheet and produce a table, which could then be inserted into a DB2® table.

Supported languages

- SQL
- C/C++
- Java™
- OLE
- OLE DB
- .NET common language runtime languages

Related concepts:

• "Routines in application development" on page 988

- "Scratchpads for UDFs and methods" in the *Application Development Guide: Programming Server Applications*
- "Table function processing model" in the *Application Development Guide: Programming Server Applications*

Related reference:
- "CREATE FUNCTION" on page 574

Methods

Methods enable you to define behaviors for structured types. They are like scalar UDFs, but can only be defined for structured types. Methods share all the features of scalar UDFs, in addition to the following features:

Features
- Strongly associated with the structured type.
- Can be sensitive to the dynamic type of the subject type.

Limitations
- Can only return a scalar value.
- Can only be used with structured types.
- Cannot be invoked against typed tables.

Common uses
- Providing operations on structured types.
- Encapsulating the structured type.

Supported languages
- SQL
- C/C++
- Java™
- OLE

Related concepts:
- "Routines in application development" on page 988
- "Scratchpads for UDFs and methods" in the *Application Development Guide: Programming Server Applications*

Related tasks:
- "Defining behavior for structured types" in the *Application Development Guide: Programming Server Applications*

Related reference:
- "CREATE TYPE (Structured) statement" in the *SQL Reference, Volume 2*
- "CREATE METHOD" on page 583

Security considerations for routines

Developing and deploying routines provides you with an opportunity to greatly improve the performance and effectiveness of your database applications. There can, however, be security risks if the deployment of routines is not managed correctly by the database administrator. The following sections describe security

risks and means by which you can mitigate these risks. The security risks are followed by a section on how to safely deploy routines whose security is unknown.

Security risks:

NOT FENCED routines can access database manager resources

NOT FENCED routines run in the same process as the database manager. Because of their close proximity to the database engine, NOT FENCED routines can accidentally or maliciously corrupt the database manager's shared memory, or damage the database control structures. Either form of damage will cause the database manager to fail. NOT FENCED routines can also corrupt databases and their tables.

To ensure the integrity of the database manager and its databases, you must thoroughly screen routines you intend to register as NOT FENCED. These routines must be fully tested, debugged, and exhibit no unexpected side-effects. In the examination of the routine, pay close attention to memory management and the use of static variables. The greatest potential for corruption arises when code does not properly manage memory or incorrectly uses static variables. These problems are prevalent in languages other than Java™ and .NET programming langauges.

In order to register a NOT FENCED routine, the CREATE_NOT_FENCED_ROUTINE authority is required. When granting the CREATE_NOT_FENCED_ROUTINE authority, be aware that the recipient can potentially gain unrestricted access to the database manager and all its resources.

Note: NOT FENCED routines are not supported in Common Criteria compliant configurations.

FENCED THREADSAFE routines can access memory in other FENCED THREADSAFE routines

FENCED THREADSAFE routines run as threads inside a shared process. Each of these routines are able to read the memory used by other routine threads in the same process. Therefore, it is possible for one threaded routine to collect sensitive data from other routines in the threaded process. Another risk inherent in the sharing of a single process, is that one routine thread with flawed memory management can corrupt other routine threads, or cause the entire threaded process to crash.

To ensure the integrity of other FENCED THREADSAFE routines, you must thoroughly screen routines you intend to register as FENCED THREADSAFE. These routines must be fully tested, debugged, and exhibit no unexpected side-effects. In the examination of the routine, pay close attention to memory management and the use of static variables. This is where the greatest potential for corruption lies, particularly in languages other than Java.

In order to register a FENCED THREADSAFE routine, the CREATE_EXTERNAL_ROUTINE authority is required. When granting the CREATE_EXTERNAL_ROUTINE authority, be aware that the recipient can potentially monitor or corrupt the memory of other FENCED THREADSAFE routines.

Write access to the database server by the owner of fenced processes can result in database manager corruption

The user ID under which fenced processes run is defined by the `db2icrt`

(create instance) or `db2iupdt` (update instance) system commands. This user ID must not have write access to the directory where routine libraries and classes are stored (in UNIX® environments, sqllib/function; in Windows® environments, sqllib\function). This user ID must also not have read or write access to any database, operating system, or otherwise critical files and directories on the database server.

If the owner of fenced processes does have write access to various critical resources on the database server, the potential for system corruption exists. For example, a database administrator registers a routine received from an unknown source as FENCED NOT THREADSAFE, thinking that any potential harm can be averted by isolating the routine in its own process. However, the user ID that owns fenced processes has write access to the sqllib/function directory. Users invoke this routine, and unbeknownst to them, it overwrites a library in sqllib/function with an alternate version of a routine body that is registered as NOT FENCED. This second routine has unrestricted access to the entire database manager, and can thereby distribute sensitive information from database tables, corrupt the databases, collect authentication information, or crash the database manager.

Ensure the user ID that owns fenced processes does not have write access to critical files or directories on the database server (especially sqllib/function and the database data directories).

Vulnerability of routine libraries and classes

If access to the directory where routine libraries and classes are stored is not controlled, routine libraries and classes can be deleted or overwritten. As discussed in the previous item, the replacement of a NOT FENCED routine body with a malicious (or poorly coded) routine can severely compromise the stability, integrity, and privacy of the database server and its resources.

To protect the integrity of routines, you must manage access to the directory containing the routine libraries and classes. Ensure that the fewest possible number of users can access this directory and its files. When assigning write access to this directory, be aware that this privilege can provide the owner of the user ID unrestricted access to the database manager and all its resources.

Deploying potentially insecure routines:

If you happen to acquire a routine from an unknown source, be sure you know exactly what it does before you build, register, and invoke it. It is recommend that you register it as FENCED and NOT THREADSAFE unless you have tested it thoroughly, and it exhibits no unexpected side-effects.

If you need to deploy a routine that does not meet the criteria for secure routines, register the routine as FENCED and NOT THREADSAFE. To ensure that database integrity is maintained, FENCED and NOT THREADSAFE routines:

- Run in a separate DB2® process, shared with no other routines. If they abnormally terminate, the database manager will be unaffected.
- Use memory that is distinct from memory used by the database. An inadvertent mistake in a value assignment will not affect the database manager.

Related concepts:

- "Routines in application development" on page 988

- "Performance considerations for developing routines" in the *Application Development Guide: Programming Server Applications*
- "Restrictions on using routines" in the *Application Development Guide: Programming Server Applications*
- "Library and class management considerations" on page 1000

Related reference:

- "CREATE FUNCTION" on page 574
- "CREATE PROCEDURE" on page 588
- "GRANT (Routine Privileges)" on page 708
- "REVOKE (Routine Privileges)" on page 742

Connection contexts in SQLJ routines

With the introduction of multithreaded routines in DB2® Universal Database, Version 8, it is important that SQLJ routines avoid the use of the default connection context. That is, each SQL statement must explicitly indicate the ConnectionContext object, and that context must be explicitly instantiated in the Java™ method. For instance, in previous releases of DB2, a SQLJ routine could be written as follows:

```
class myClass
{
  public static void myRoutine( short myInput )
  {
    DefaultContext ctx = DefaultContext.getDefaultContext();
    #sql { some SQL statement };
  }
}
```

This use of the default context causes all threads in a multithreaded environment to use the same connection context, which, in turn, will result in unexpected failures.

The SQLJ routine above must be changed as follows:

```
#context MyContext;

class myClass
{
  public static void myRoutine( short myInput )
  {
    MyContext ctx = new MyContext( "jdbc:default:connection", false );
    #sql [ctx] { some SQL statement };
    ctx.close();
  }
}
```

This way, each invocation of the routine will create its own unique ConnectionContext (and underlying JDBC connection), which avoids unexpected interference by concurrent threads.

Related concepts:

- "Java routines" in the *Application Development Guide: Programming Server Applications*
- "Basic steps in writing an SQLJ application" in the *Application Development Guide: Programming Client Applications*
- "SQL statements in an SQLJ application" in the *Application Development Guide: Programming Client Applications*

Library and class management considerations

When developing routines for DB2®, you have the option of using a variety of different programming languages, including SQL, Java™, C, C++, and .NET compatible languages. If you develop routines in a language other than SQL, they are known as external routines. The compiled source code for an external routine is referred to as a routine body.

Protecting routine bodies

The bodies of external routines reside in libraries and classes stored on the database server. These files are not backed up or protected in any way by DB2. The CREATE statement used to create a routine in the database adds routine definition information to the database catalogs including information about where the external code librarary associated with the routine resides. This is specified in the EXTERNAL clause in the CREATE statement. The routine library or class specified in the EXTERNAL clause is not stored in the database, but resides in the file system of the server. It is imperative for the successful invocation of your external routines that the library associated with a given routine exist in the location specified in the EXTERNAL clause. It is possible that the library can be moved or deleted. If this happens the routine can no longer be invoked successfully.

To preserve the integrity of the invoking clients and routines that depend on the routine, you must prevent the routine body from being inadvertently or intentionally deleted or replaced. This can be done by managing access to the directory containing the routine and by protecting the routine body itself.

Note: The bodies of SQL routines are considered to be part of the database, and as such, will be backed up with other database objects. However, like external routines, their bodies are prone to being altered, and therefore require the same protection.

The scope of routine bodies

For routines to be used in a database, they must be cataloged with that same database. If there are multiple databases in an instance, you can catalog external routines in one database using routine bodies that are already being used in another database. Hence, the scope of routine bodies is instance wide. While this affords the possibility of reusing code, library or class name conflicts can arise in situations where code is not being reused.

Specifically, library or class name conflicts can manifest themselves in a situation such as the following: there are multiple databases in a single instance and the routines in each database use their own libraries and classes of routine bodies. A conflict arises when the name of a library or class used by a routine in one database is identical to the name of a library or class used by a routine in another database (in the same instance). This is because routine bodies are normally stored in the sqllib/function directory, which is used by all the databases of an instance.

For non-Java routines library name conflicts can be resolved with the following steps:

1. Store the libraries with routine bodies in separate directories for each database.
2. Catalog the routines with the EXTERNAL NAME clause, specifying the full path of the given library.

For Java routines class name conflicts are not solved by moving the files in question into different directories, because the CLASSPATH environment variable is instance-wide. The first class encountered in the CLASSPATH is the one that is used. Therefore, if you have two different Java routines that reference a class with the same name, one of the routines will use the incorrect class. There are two possible solutions: either rename the affected classes, or create a separate instance for each database.

Updating a routine body

If you need to change the body of a routine, do not recompile and relink the routine to the same file (for example, sqllib/function/foo.a) the current routine is using while the database manager is running. If a current routine invocation is accessing a cached version of the routine process and the underlying library is replaced, this can cause the rotine invocation to fail. If it is necessary to change the body of a routine without stopping and restarting DB2, complete the following steps:

1. Create the new body for the routine with a different library or class name.
2. Bind the new routine body (if it contains embedded SQL) with the database.
3. Use the ALTER statement to change the routine's EXTERNAL NAME to reference the updated routine body.

Once the ALTER updates the routine's catalog entries, all subsequent invocations of the updated routine will point to the new routine body.

For updating Java routines that are built into JAR files, you must issue a CALL SQLJ.REFRESH_CLASSES() statement to force DB2 to load the new classes. If you do not issue the CALL SQLJ.REFRESH_CLASSES() statement after you update Java routine classes, DB2 continues to use the previous versions of the classes. DB2 refreshes the classes when a COMMIT or ROLLBACK occurs.

Note: If the routine body to be updated is used by routines cataloged in multiple databases, the actions prescribed in this section must be completed for each affected database.

Library management-related performance considerations

The DB2 library manager dynamically adjusts its library caching according to your workload. For optimal performance consider the following:

- Keep the number of routines in your libraries as small as possible. If you are including multiple routines in the same library, ensure that you group them based on whether they are invoked in the same time frame. Consider a scenario where in a number of applications a call to the procedure ProcA is followed by a call to the procedure ProcB. In this situation, it might be appropriate to include ProcA and ProcB in the same library. With a library caching scheme, it is better to have numerous smaller libraries than a few large libraries.
- The load cost for a library in the C process is paid only once for libraries that are consistently in use by C routines. After the routine's first invocation, all subsequent invocations, from the same thread in the process, do not need to load the routine's library.

Routine bodies in partitioned databases

When using external routines in partitioned databases, the library or class must be available on all partitions of the database.

On UNIX®, sqllib/function is a good location for routine bodies, because the sqllib directory is cross-mounted between all partitions of the database.

On Windows®, a good approach would be to create a shared directory accessible to all the partitions, and put the libraries or classes in this directory.

Related concepts:

- "Performance considerations for developing routines" in the *Application Development Guide: Programming Server Applications*
- "Security considerations for routines" on page 996
- "Restrictions on using routines" in the *Application Development Guide: Programming Server Applications*

Related reference:

- "CREATE FUNCTION" on page 574
- "CREATE PROCEDURE" on page 588
- "CREATE METHOD" on page 583
- "ALTER FUNCTION" on page 519
- "ALTER METHOD" on page 521
- "ALTER PROCEDURE" on page 522

Rebuilding DB2 routine shared libraries

DB2® will cache the shared libraries used for stored procedures and user-defined functions once loaded. If you are developing a routine, you might want to test loading the same shared library a number of times, and this caching can prevent you from picking up the latest version of a shared library. The way to avoid caching problems depends on the type of routine:

1. **Fenced, not threadsafe routines.** The database manager configuration keyword KEEPFENCED has a default value of YES. This keeps the fenced mode process alive. This default setting can interfere with reloading the library. It is best to change the value of this keyword to NO while developing fenced, not threadsafe routines, and then change it back to YES when you are ready to load the final version of your shared library. For more information, see "Updating the database manager configuration file" on page 1003.
2. **Trusted or threadsafe routines.** Except for SQL routines (including SQL procedures), the only way to ensure that an updated version of a DB2 routine library is picked up when that library is used for trusted, or threadsafe routines, is to recycle the DB2 instance by entering db2stop followed by db2start on the command line. This is not needed for an SQL routine because when it is recreated, the compiler uses a new unique library name to prevent possible conflicts.

For routines other than SQL routines, you can also avoid caching problems by creating the new version of the routine with a differently named library (for example foo.a becomes foo.1.a), and then using either the ALTER PROCEDURE or ALTER FUNCTION SQL statement with the new library.

Related tasks:

- "Updating the database manager configuration file" on page 1003

Related reference:

- "ALTER FUNCTION" on page 519
- "ALTER PROCEDURE" on page 522

Updating the database manager configuration file

This file contains important settings for application development.

The keyword KEEPFENCED has the default value YES. For fenced, not threadsafe routines (stored procedures and UDFs), this keeps the routine process alive. It is best to change the value of this keyword to NO while developing these routines, and then change it back to YES when you are ready to load the final version of your shared library. For more information, see "Rebuilding DB2 routine shared libraries" on page 1002.

Note: KEEPFENCED was known as KEEPDARI in previous versions of DB2.

For Java application development, you need to update the JDK_PATH keyword with the path where the Java Development Kit is installed.

Note: JDK_PATH was known as JDK11_PATH in previous versions of DB2.

Procedure:

To change these settings enter:

```
db2 update dbm cfg using <keyword> <value>
```

For example, to set the keyword KEEPFENCED to NO:

```
db2 update dbm cfg using KEEPFENCED NO
```

To set the JDK_PATH keyword to the directory /home/db2inst/jdk13:

```
db2 update dbm cfg using JDK_PATH /home/db2inst/jdk13
```

To view the current settings in the database manager configuration file, enter:

```
db2 get dbm cfg
```

Note: On Windows, you need to enter these commands in a DB2 command window.

Related concepts:

- "Rebuilding DB2 routine shared libraries" on page 1002
- "Database manager instances" in the *Application Development Guide: Building and Running Applications*

Related tasks:

- "Setting up the Java environment" in the *Application Development Guide: Building and Running Applications*

Related reference:

- "CREATE FUNCTION" on page 574
- "CREATE PROCEDURE" on page 588
- "GET DATABASE MANAGER CONFIGURATION" on page 281
- "RESET DATABASE MANAGER CONFIGURATION Command" in the *Command Reference*

SQLCA (SQL communications area)

An SQLCA is a collection of variables that is updated at the end of the execution of every SQL statement. A program that contains executable SQL statements and is precompiled with option LANGLEVEL SAA1 (the default) or MIA must provide exactly one SQLCA, though more than one SQLCA is possible by having one SQLCA per thread in a multi-threaded application.

When a program is precompiled with option LANGLEVEL SQL92E, an SQLCODE or SQLSTATE variable may be declared in the SQL declare section or an SQLCODE variable can be declared somewhere in the program.

An SQLCA should not be provided when using LANGLEVEL SQL92E. The SQL INCLUDE statement can be used to provide the declaration of the SQLCA in all languages but REXX. The SQLCA is automatically provided in REXX.

To display the SQLCA after each command executed through the command line processor, issue the command `db2 -a`. The SQLCA is then provided as part of the output for subsequent commands. The SQLCA is also dumped in the db2diag.log file.

SQLCA field descriptions

Table 75. Fields of the SQLCA. The field names shown are those present in an SQLCA that is obtained via an INCLUDE statement.

Name	Data Type	Field Values
sqlcaid	CHAR(8)	An "eye catcher" for storage dumps containing 'SQLCA'. The sixth byte is 'L' if line number information is returned from parsing an SQL procedure body.
sqlcabc	INTEGER	Contains the length of the SQLCA, 136.
sqlcode	INTEGER	Contains the SQL return code. **Code** **Means** **0** Successful execution (although one or more SQLWARN indicators may be set). **positive** Successful execution, but with a warning condition. **negative** Error condition.
sqlerrml	SMALLINT	Length indicator for *sqlerrmc*, in the range 0 through 70. 0 means that the value of *sqlerrmc* is not relevant.
sqlerrmc	VARCHAR (70)	Contains one or more tokens, separated by X'FF', which are substituted for variables in the descriptions of error conditions. This field is also used when a successful connection is completed. When a NOT ATOMIC compound SQL statement is issued, it may contain information on up to seven errors. The last token might be followed by X'FF'. The *sqlerrml* value will include any trailing X'FF'.

Table 75. Fields of the SQLCA (continued). The field names shown are those present in an SQLCA that is obtained via an INCLUDE statement.

Name	Data Type	Field Values
sqlerrp	CHAR(8)	Begins with a three-letter identifier indicating the product, followed by five digits indicating the version, release, and modification level of the product. For example, SQL08010 means DB2 Universal Database Version 8 Release 1 Modification level 0. If SQLCODE indicates an error condition, this field identifies the module that returned the error. This field is also used when a successful connection is completed.
sqlerrd	ARRAY	Six INTEGER variables that provide diagnostic information. These values are generally empty if there are no errors, except for sqlerrd(6) from a partitioned database.
sqlerrd(1)	INTEGER	If connection is invoked and successful, contains the maximum expected difference in length of mixed character data (CHAR data types) when converted to the database code page from the application code page. A value of 0 or 1 indicates no expansion; a value greater than 1 indicates a possible expansion in length; a negative value indicates a possible contraction. On successful return from an SQL procedure, contains the return status value from the SQL procedure.
sqlerrd(2)	INTEGER	If connection is invoked and successful, contains the maximum expected difference in length of mixed character data (CHAR data types) when converted to the application code page from the database code page. A value of 0 or 1 indicates no expansion; a value greater than 1 indicates a possible expansion in length; a negative value indicates a possible contraction. If the SQLCA results from a NOT ATOMIC compound SQL statement that encountered one or more errors, the value is set to the number of statements that failed.
sqlerrd(3)	INTEGER	If PREPARE is invoked and successful, contains an estimate of the number of rows that will be returned. After INSERT, UPDATE, DELETE, or MERGE, contains the actual number of rows that qualified for the operation. If compound SQL is invoked, contains an accumulation of all sub-statement rows. If CONNECT is invoked, contains 1 if the database can be updated, or 2 if the database is read only. If the OPEN statement is invoked, and the cursor contains SQL data change statements, this field contains the sum of the number of rows that qualified for the embedded insert, update, delete, or merge operations. If CREATE PROCEDURE for an SQL procedure is invoked, and an error is encountered when parsing the SQL procedure body, contains the line number where the error was encountered. The sixth byte of sqlcaid must be 'L' for this to be a valid line number.

Table 75. Fields of the SQLCA (continued). The field names shown are those present in an SQLCA that is obtained via an INCLUDE statement.

Name	Data Type	Field Values
sqlerrd(4)	INTEGER	If PREPARE is invoked and successful, contains a relative cost estimate of the resources required to process the statement. If compound SQL is invoked, contains a count of the number of successful sub-statements. If CONNECT is invoked, contains 0 for a one-phase commit from a down-level client; 1 for a one-phase commit; 2 for a one-phase, read-only commit; and 3 for a two-phase commit.
sqlerrd(5)	INTEGER	Contains the total number of rows deleted, inserted, or updated as a result of both: • The enforcement of constraints after a successful delete operation • The processing of triggered SQL statements from activated triggers If compound SQL is invoked, contains an accumulation of the number of such rows for all sub-statements. In some cases, when an error is encountered, this field contains a negative value that is an internal error pointer. If CONNECT is invoked, contains an authentication type value of 0 for server authentication; 1 for client authentication; 2 for authentication using DB2 Connect; 4 for SERVER_ENCRYPT authentication; 5 for authentication using DB2 Connect with encryption; 7 for KERBEROS authentication; 8 for KRB_SERVER_ENCRYPT authentication; 9 for GSSPLUGIN authentication; 10 for GSS_SERVER_ENCRYPT authentication; and 255 for unspecified authentication.
sqlerrd(6)	INTEGER	For a partitioned database, contains the partition number of the partition that encountered the error or warning. If no errors or warnings were encountered, this field contains the partition number of the coordinator node. The number in this field is the same as that specified for the partition in the db2nodes.cfg file.
sqlwarn	Array	A set of warning indicators, each containing a blank or W. If compound SQL is invoked, contains an accumulation of the warning indicators set for all sub-statements.
sqlwarn0	CHAR(1)	Blank if all other indicators are blank; contains W if at least one other indicator is not blank.
sqlwarn1	CHAR(1)	Contains W if the value of a string column was truncated when assigned to a host variable. Contains N if the null terminator was truncated. Contains A if the CONNECT or ATTACH is successful, and the authorization name for the connection is longer than 8 bytes. Contains P if the PREPARE statement relative cost estimate stored in sqlerrd(4) exceeded the value that could be stored in an INTEGER or was less than 1, and either the CURRENT EXPLAIN MODE or the CURRENT EXPLAIN SNAPSHOT special register is set to a value other than NO.
sqlwarn2	CHAR(1)	Contains W if null values were eliminated from the argument of a function. [a]

Table 75. Fields of the SQLCA (continued). The field names shown are those present in an SQLCA that is obtained via an INCLUDE statement.

Name	Data Type	Field Values
sqlwarn3	CHAR(1)	Contains W if the number of columns is not equal to the number of host variables. Contains Z if the number of result set locators specified on the ASSOCIATE LOCATORS statement is less than the number of result sets returned by a procedure.
sqlwarn4	CHAR(1)	Contains W if a prepared UPDATE or DELETE statement does not include a WHERE clause.
sqlwarn5	CHAR(1)	Reserved for future use.
sqlwarn6	CHAR(1)	Contains W if the result of a date calculation was adjusted to avoid an impossible date.
sqlwarn7	CHAR(1)	Reserved for future use. If CONNECT is invoked and successful, contains 'E' if the DYN_QUERY_MGMT database configuration parameter is enabled.
sqlwarn8	CHAR(1)	Contains W if a character that could not be converted was replaced with a substitution character.
sqlwarn9	CHAR(1)	Contains W if arithmetic expressions with errors were ignored during column function processing.
sqlwarn10	CHAR(1)	Contains W if there was a conversion error when converting a character data value in one of the fields in the SQLCA.
sqlstate	CHAR(5)	A return code that indicates the outcome of the most recently executed SQL statement.

[a] Some functions may not set SQLWARN2 to W, even though null values were eliminated, because the result was not dependent on the elimination of null values.

Error reporting

The order of error reporting is as follows:

1. Severe error conditions are always reported. When a severe error is reported, there are no additions to the SQLCA.
2. If no severe error occurs, a deadlock error takes precedence over other errors.
3. For all other errors, the SQLCA for the first negative SQL code is returned.
4. If no negative SQL codes are detected, the SQLCA for the first warning (that is, positive SQL code) is returned.

   In a partitioned database system, the exception to this rule occurs if a data manipulation operation is invoked against a table that is empty on one partition, but has data on other partitions. SQLCODE +100 is only returned to the application if agents from all partitions return SQL0100W, either because the table is empty on all partitions, or there are no more rows that satisfy the WHERE clause in an UPDATE statement.

SQLCA usage in partitioned database systems

In partitioned database systems, one SQL statement may be executed by a number of agents on different partitions, and each agent may return a different SQLCA for different errors or warnings. The coordinator agent also has its own SQLCA.

SQLCA usage in partitioned database systems

To provide a consistent view for applications, all SQLCA values are merged into one structure, and SQLCA fields indicate global counts, such that:

- For all errors and warnings, the *sqlwarn* field contains the warning flags received from all agents.
- Values in the *sqlerrd* fields indicating row counts are accumulations from all agents.

Note that SQLSTATE 09000 may not be returned every time an error occurs during the processing of a triggered SQL statement.

SQLDA (SQL descriptor area)

An SQLDA is a collection of variables that is required for execution of the SQL DESCRIBE statement. The SQLDA variables are options that can be used by the PREPARE, OPEN, FETCH, and EXECUTE statements. An SQLDA communicates with dynamic SQL; it can be used in a DESCRIBE statement, modified with the addresses of host variables, and then reused in a FETCH or EXECUTE statement.

SQLDAs are supported for all languages, but predefined declarations are provided only for C, REXX, FORTRAN, and COBOL.

The meaning of the information in an SQLDA depends on its use. In PREPARE and DESCRIBE, an SQLDA provides information to an application program about a prepared statement. In OPEN, EXECUTE, and FETCH, an SQLDA describes host variables.

In DESCRIBE and PREPARE, if any one of the columns being described is either a LOB type (LOB locators and file reference variables do not require doubled SQLDAs), reference type, or a user-defined type, the number of SQLVAR entries for the entire SQLDA will be doubled. For example:

- When describing a table with 3 VARCHAR columns and 1 INTEGER column, there will be 4 SQLVAR entries
- When describing a table with 2 VARCHAR columns, 1 CLOB column, and 1 integer column, there will be 8 SQLVAR entries

In EXECUTE, FETCH, and OPEN, if any one of the variables being described is a LOB type (LOB locators and file reference variables do not require doubled SQLDAs) or a structured type, the number of SQLVAR entries for the entire SQLDA must be doubled. (Distinct types and reference types are not relevant in these cases, because the additional information in the double entries is not required by the database.)

SQLDA field descriptions

An SQLDA consists of four variables followed by an arbitrary number of occurrences of a sequence of variables collectively named SQLVAR. In OPEN, FETCH, and EXECUTE, each occurrence of SQLVAR describes a host variable. In DESCRIBE and PREPARE, each occurrence of SQLVAR describes a column of a result table or a parameter marker. There are two types of SQLVAR entries:

- **Base SQLVARs:** These entries are always present. They contain the base information about the column, parameter marker, or host variable such as data type code, length attribute, column name, host variable address, and indicator variable address.

- **Secondary SQLVARs:** These entries are only present if the number of SQLVAR entries is doubled as per the rules outlined above. For user-defined types (distinct or structured), they contain the user-defined type name. For reference types, they contain the target type of the reference. For LOBs, they contain the length attribute of the host variable and a pointer to the buffer that contains the actual length. (The distinct type and LOB information does not overlap, so distinct types can be based on LOBs without forcing the number of SQLVAR entries on a DESCRIBE to be tripled.) If locators or file reference variables are used to represent LOBs, these entries are not necessary.

In SQLDAs that contain both types of entries, the base SQLVARs are in a block before the block of secondary SQLVARs. In each, the number of entries is equal to the value in SQLD (even though many of the secondary SQLVAR entries may be unused).

The circumstances under which the SQLVAR entries are set by DESCRIBE is detailed in “Effect of DESCRIBE on the SQLDA” on page 1012.

Fields in the SQLDA header

Table 76. Fields in the SQLDA Header

C Name	SQL Data Type	Usage in DESCRIBE and PREPARE (set by the database manager except for SQLN)	Usage in FETCH, OPEN, and EXECUTE (set by the application prior to executing the statement)
sqldaid	CHAR(8)	The seventh byte of this field is a flag byte named SQLDOUBLED. The database manager sets SQLDOUBLED to the character ’2’ if two SQLVAR entries have been created for each column; otherwise it is set to a blank (X'20' in ASCII, X'40' in EBCDIC). See “Effect of DESCRIBE on the SQLDA” on page 1012 for details on when SQLDOUBLED is set.	The seventh byte of this field is used when the number of SQLVARs is doubled. It is named SQLDOUBLED. If any of the host variables being described is a structured type, BLOB, CLOB, or DBCLOB, the seventh byte must be set to the character ’2’; otherwise it can be set to any character but the use of a blank is recommended.
sqldabc	INTEGER	For 32 bit, the length of the SQLDA, equal to SQLN*44+16. For 64 bit, the length of the SQLDA, equal to SQLN*56+16	For 32 bit, the length of the SQLDA, >= to SQLN*44+16. For 64 bit, the length of the SQLDA, >= to SQLN*56+16.
sqln	SMALLINT	Unchanged by the database manager. Must be set to a value greater than or equal to zero before the DESCRIBE statement is executed. Indicates the total number of occurrences of SQLVAR.	Total number of occurrences of SQLVAR provided in the SQLDA. SQLN must be set to a value greater than or equal to zero.
sqld	SMALLINT	Set by the database manager to the number of columns in the result table or to the number of parameter markers.	The number of host variables described by occurrences of SQLVAR.

Fields in an occurrence of a base SQLVAR

Table 77. Fields in a Base SQLVAR

Name	Data Type	Usage in DESCRIBE and PREPARE	Usage in FETCH, OPEN, and EXECUTE
sqltype	SMALLINT	Indicates the data type of the column or parameter marker, and whether it can contain nulls. (Parameter markers are always considered nullable.) Table 79 on page 1014 lists the allowable values and their meanings. Note that for a distinct or reference type, the data type of the base type is placed into this field. For a structured type, the data type of the result of the FROM SQL transform function of the transform group (based on the CURRENT DEFAULT TRANSFORM GROUP special register) for the type is placed into this field. There is no indication in the base SQLVAR that it is part of the description of a user-defined type or reference type.	Same for host variable. Host variables for datetime values must be character string variables. For FETCH, a datetime type code means a fixed-length character string. If sqltype is an even number value, the sqlind field is ignored.
sqllen	SMALLINT	The length attribute of the column or parameter marker. For datetime columns and parameter markers, the length of the string representation of the values. See Table 79 on page 1014. Note that the value is set to 0 for large object strings (even for those whose length attribute is small enough to fit into a two byte integer).	The length attribute of the host variable. See Table 79 on page 1014. Note that the value is ignored by the database manager for CLOB, DBCLOB, and BLOB columns. The len.sqllonglen field in the Secondary SQLVAR is used instead.
sqldata	pointer	For string SQLVARS, sqldata contains the code page. For character-string SQLVARs where the column is defined with the FOR BIT DATA attribute, sqldata contains 0. For other character-string SQLVARS, sqldata contains either the SBCS code page for SBCS data, or the SBCS code page associated with the composite MBCS code page for MBCS data. For Japanese EUC, Traditional Chinese EUC, and Unicode UTF-8 character-string SQLVARS, sqldata contains 954, 964, and 1208 respectively. For all other column types, sqldata is undefined.	Contains the address of the host variable (where the fetched data will be stored).
sqlind	pointer	For character-string SQLVARS, sqlind contains 0, except for MBCS data, when sqlind contains the DBCS code page associated with the composite MBCS code page. For all other types, sqlind is undefined.	Contains the address of an associated indicator variable, if there is one; otherwise, not used. If sqltype is an even number value, the sqlind field is ignored.

Table 77. Fields in a Base SQLVAR (continued)

Name	Data Type	Usage in DESCRIBE and PREPARE	Usage in FETCH, OPEN, and EXECUTE
sqlname	VARCHAR (30)	Contains the unqualified name of the column or parameter marker. For columns and parameter markers that have a system-generated name, the thirtieth byte is set to X'FF'. For column names specified by the AS clause, this byte is X'00'.	When using DB2 Connect to access the server, sqlname can be set to indicate a FOR BIT DATA string as follows: • the length of sqlname is 8 • the first four bytes of sqlname are X'00000000' • the remaining four bytes of sqlname are reserved (and currently ignored).

Fields in an occurrence of a secondary SQLVAR

Table 78. Fields in a Secondary SQLVAR

Name	Data Type	Usage in DESCRIBE and PREPARE	Usage in FETCH, OPEN, and EXECUTE
len.sqllonglen	INTEGER	The length attribute of a BLOB, CLOB, or DBCLOB column or parameter marker.	The length attribute of a BLOB, CLOB, or DBCLOB host variable. The database manager ignores the SQLLEN field in the Base SQLVAR for the data types. The length attribute stores the number of bytes for a BLOB or CLOB, and the number of characters for a DBCLOB.
reserve2	CHAR(3) for 32 bit, and CHAR(11) for 64 bit.	Not used.	Not used.
sqlflag4	CHAR(1)	The value is X'01' if the SQLVAR represents a reference type with a target type named in sqldatatype_name. The value is X'12' if the SQLVAR represents a structured type, with the user-defined type name in sqldatatype_name. Otherwise, the value is X'00'.	Set to X'01' if the SQLVAR represents a reference type with a target type named in sqldatatype_name. Set to X'12' if the SQLVAR represents a structured type, with the user-defined type name in sqldatatype_name. Otherwise, the value is X'00'.

Table 78. Fields in a Secondary SQLVAR (continued)

Name	Data Type	Usage in DESCRIBE and PREPARE	Usage in FETCH, OPEN, and EXECUTE
sqldatalen	pointer	Not used.	Used for BLOB, CLOB, and DBCLOB host variables only. If this field is NULL, then the actual length (in characters) should be stored in the 4 bytes immediately before the start of the data and SQLDATA should point to the first byte of the field length. If this field is not NULL, it contains a pointer to a 4 byte long buffer that contains the actual length *in bytes* (even for DBCLOB) of the data in the buffer pointed to from the SQLDATA field in the matching base SQLVAR. Note that, whether or not this field is used, the len.sqllonglen field must be set.
sqldatatype_name	VARCHAR(27)	For a user-defined type, the database manager sets this to the fully qualified user-defined type name.[1] For a reference type, the database manager sets this to the fully qualified type name of the target type of the reference.	For structured types, set to the fully qualified user-defined type name in the format indicated in the table note.[1]
reserved	CHAR(3)	Not used.	Not used.

[1] The first 8 bytes contain the schema name of the type (extended to the right with spaces, if necessary). Byte 9 contains a dot (.). Bytes 10 to 27 contain the low order portion of the type name, which is *not* extended to the right with spaces.

Note that, although the prime purpose of this field is for the name of user-defined types, the field is also set for IBM predefined data types. In this case, the schema name is SYSIBM, and the low order portion of the name is the name stored in the TYPENAME column of the DATATYPES catalog view. For example:

```
type name        length   sqldatatype_name
---------        ------   ----------------
A.B              10       A       .B
INTEGER          16       SYSIBM  .INTEGER
"Frank's".SMINT  13       Frank's .SMINT
MY."type  "      15       MY      .type
```

Effect of DESCRIBE on the SQLDA

For a DESCRIBE OUTPUT or PREPARE OUTPUT INTO statement, the database manager always sets SQLD to the number of columns in the result set, or the number of output parameter markers. For a DESCRIBE INPUT or PREPARE INPUT INTO statement, the database manager always sets SQLD to the number of input parameter markers in the statement. Note that a parameter marker that corresponds to an INOUT parameter in a CALL statement is described in both the input and output descriptors.

The SQLVARs in the SQLDA are set in the following cases:

- SQLN >= SQLD and no entry is either a LOB, user-defined type or reference type

  The first SQLD SQLVAR entries are set and SQLDOUBLED is set to blank.
- SQLN >= 2*SQLD and at least one entry is a LOB, user-defined type or reference type

  Two times SQLD SQLVAR entries are set, and SQLDOUBLED is set to '2'.
- SQLD <= SQLN < 2*SQLD and at least one entry is a distinct type or reference type, but there are no LOB entries or structured type entries

  The first SQLD SQLVAR entries are set and SQLDOUBLED is set to blank. If the SQLWARN bind option is YES, a warning SQLCODE +237 (SQLSTATE 01594) is issued.

The SQLVARs in the SQLDA are NOT set (requiring allocation of additional space and another DESCRIBE) in the following cases:

- SQLN < SQLD and no entry is either a LOB, user-defined type or reference type

  No SQLVAR entries are set and SQLDOUBLED is set to blank. If the SQLWARN bind option is YES, a warning SQLCODE +236 (SQLSTATE 01005) is issued.

  Allocate SQLD SQLVARs for a successful DESCRIBE.
- SQLN < SQLD and at least one entry is a distinct type or reference type, but there are no LOB entries or structured type entries

  No SQLVAR entries are set and SQLDOUBLED is set to blank. If the SQLWARN bind option is YES, a warning SQLCODE +239 (SQLSTATE 01005) is issued.

  Allocate 2*SQLD SQLVARs for a successful DESCRIBE including the names of the distinct types and target types of reference types.
- SQLN < 2*SQLD and at least one entry is a LOB or a structured type

  No SQLVAR entries are set and SQLDOUBLED is set to blank. A warning SQLCODE +238 (SQLSTATE 01005) is issued (regardless of the setting of the SQLWARN bind option).

  Allocate 2*SQLD SQLVARs for a successful DESCRIBE.

References in the above lists to LOB entries include distinct type entries whose source type is a LOB type.

The SQLWARN option of the BIND or PREP command is used to control whether the DESCRIBE (or PREPARE INTO) will return the warning SQLCODEs +236, +237, +239. It is recommended that your application code always consider that these SQLCODEs could be returned. The warning SQLCODE +238 is always returned when there are LOB or structured type entries in the select list and there are insufficient SQLVARs in the SQLDA. This is the only way the application can know that the number of SQLVARs must be doubled because of a LOB or structured type entry in the result set.

If a structured type entry is being described, but no FROM SQL transform is defined (either because no TRANSFORM GROUP was specified using the CURRENT DEFAULT TRANSFORM GROUP special register (SQLSTATE 42741), or because the name group does not have a FROM SQL transform function defined (SQLSTATE 42744), the DESCRIBE will return an error. This error is the same error returned for a DESCRIBE of a table with a structured type entry.

SQLTYPE and SQLLEN

Table 79 on page 1014 shows the values that may appear in the SQLTYPE and SQLLEN fields of the SQLDA. In DESCRIBE and PREPARE INTO, an even value

of SQLTYPE means that the column does not allow nulls, and an odd value means the column does allow nulls. In FETCH, OPEN, and EXECUTE, an even value of SQLTYPE means that no indicator variable is provided, and an odd value means that SQLIND contains the address of an indicator variable.

Table 79. SQLTYPE and SQLLEN values for DESCRIBE, FETCH, OPEN, and EXECUTE

	For DESCRIBE and PREPARE INTO		For FETCH, OPEN, and EXECUTE	
SQLTYPE	**Column Data Type**	**SQLLEN**	**Host Variable Data Type**	**SQLLEN**
384/385	date	10	fixed-length character string representation of a date	length attribute of the host variable
388/389	time	8	fixed-length character string representation of a time	length attribute of the host variable
392/393	timestamp	26	fixed-length character string representation of a timestamp	length attribute of the host variable
396/397	DATALINK	length attribute of the column	DATALINK	length attribute of the host variable
400/401	N/A	N/A	NUL-terminated graphic string	length attribute of the host variable
404/405	BLOB	0 *	BLOB	Not used. *
408/409	CLOB	0 *	CLOB	Not used. *
412/413	DBCLOB	0 *	DBCLOB	Not used. *
448/449	varying-length character string	length attribute of the column	varying-length character string	length attribute of the host variable
452/453	fixed-length character string	length attribute of the column	fixed-length character string	length attribute of the host variable
456/457	long varying-length character string	length attribute of the column	long varying-length character string	length attribute of the host variable
460/461	N/A	N/A	NUL-terminated character string	length attribute of the host variable
464/465	varying-length graphic string	length attribute of the column	varying-length graphic string	length attribute of the host variable
468/469	fixed-length graphic string	length attribute of the column	fixed-length graphic string	length attribute of the host variable
472/473	long varying-length graphic string	length attribute of the column	long graphic string	length attribute of the host variable
480/481	floating point	8 for double precision, 4 for single precision	floating point	8 for double precision, 4 for single precision
484/485	packed decimal	precision in byte 1; scale in byte 2	packed decimal	precision in byte 1; scale in byte 2
492/493	big integer	8	big integer	8
496/497	large integer	4	large integer	4
500/501	small integer	2	small integer	2
916/917	Not applicable	Not applicable	BLOB file reference variable.	267

Table 79. SQLTYPE and SQLLEN values for DESCRIBE, FETCH, OPEN, and EXECUTE (continued)

	For DESCRIBE and PREPARE INTO		For FETCH, OPEN, and EXECUTE	
SQLTYPE	**Column Data Type**	**SQLLEN**	**Host Variable Data Type**	**SQLLEN**
920/921	Not applicable	Not applicable	CLOB file reference variable.	267
924/925	Not applicable	Not applicable	DBCLOB file reference variable.	267
960/961	Not applicable	Not applicable	BLOB locator	4
964/965	Not applicable	Not applicable	CLOB locator	4
968/969	Not applicable	Not applicable	DBCLOB locator	4

Note:

- The len.sqllonglen field in the secondary SQLVAR contains the length attribute of the column.
- The SQLTYPE has changed from the previous version for portability in DB2. The values from the previous version (see previous version SQL Reference) will continue to be supported.

Unrecognized and unsupported SQLTYPEs

The values that appear in the SQLTYPE field of the SQLDA are dependent on the level of data type support available at the sender as well as at the receiver of the data. This is particularly important as new data types are added to the product.

New data types may or may not be supported by the sender or receiver of the data and may or may not even be recognized by the sender or receiver of the data. Depending on the situation, the new data type may be returned, or a compatible data type agreed upon by both the sender and receiver of the data may be returned or an error may result.

When the sender and receiver agree to use a compatible data type, the following indicates the mapping that will take place. This mapping will take place when at least one of the sender or the receiver does not support the data type provided. The unsupported data type can be provided by either the application or the database manager.

Data Type	Compatible Data Type
BIGINT	DECIMAL(19, 0)
ROWID[1]	VARCHAR(40) FOR BIT DATA

[1] ROWID is supported by DB2 Universal Database for z/OS and OS/390 Version 6.

Note that no indication is given in the SQLDA that the data type is substituted.

Packed decimal numbers

Packed decimal numbers are stored in a variation of Binary Coded Decimal (BCD) notation. In BCD, each nybble (four bits) represents one decimal digit. For example, 0001 0111 1001 represents 179. Therefore, read a packed decimal value nybble by nybble. Store the value in bytes and then read those bytes in hexadecimal representation to return to decimal. For example, 0001 0111 1001 becomes 00000001 01111001 in binary representation. By reading this number as hexadecimal, it becomes 0179.

The decimal point is determined by the scale. In the case of a DEC(12,5) column, for example, the rightmost 5 digits are to the right of the decimal point.

Sign is indicated by a nybble to the right of the nybbles representing the digits. A positive or negative sign is indicated as follows:

Table 80. Values for Sign Indicator of a Packed Decimal Number

	Representation		
Sign	**Binary**	**Decimal**	**Hexadecimal**
Positive (+)	1100	12	C
Negative (-)	1101	13	D

In summary:

- To store any value, allocate $p/2+1$ bytes, where p is precision.
- Assign the nybbles from left to right to represent the value. If a number has an even precision, a leading zero nybble is added. This assignment includes leading (insignificant) and trailing (significant) zero digits.
- The sign nybble will be the second nybble of the last byte.

For example:

Column	Value	Nybbles in Hexadecimal Grouped by Bytes
DEC(8,3)	6574.23	00 65 74 23 0C
DEC(6,2)	-334.02	00 33 40 2D
DEC(7,5)	5.2323	05 23 23 0C
DEC(5,2)	-23.5	02 35 0D

SQLLEN field for decimal

The SQLLEN field contains the precision (first byte) and scale (second byte) of the decimal column. If writing a portable application, the precision and scale bytes should be set individually, versus setting them together as a short integer. This will avoid integer byte reversal problems.

For example, in C:

```
((char *)&(sqlda->sqlvar[i].sqllen))[0] = precision;
((char *)&(sqlda->sqlvar[i].sqllen))[1] = scale;
```

Related reference:

- "CHAR scalar function" in the *SQL Reference, Volume 1*

SQL-AUTHORIZATIONS

This structure is used to return information after a call to the sqluadau API. The data type of all fields is SMALLINT. The first half of the following table contains authorities granted directly to a user. The second half of the table contains authorities granted to the groups to which a user belongs.

Table 81. Fields in the SQL-AUTHORIZATIONS Structure

Field Name	Description
SQL_AUTHORIZATIONS_LEN	Size of structure.
SQL_SYSADM_AUTH	SYSADM authority.
SQL_SYSCTRL_AUTH	SYSCTRL authority.
SQL_SYSMAINT_AUTH	SYSMAINT authority.

Table 81. Fields in the SQL-AUTHORIZATIONS Structure (continued)

Field Name	Description
SQL_DBADM_AUTH	DBADM authority.
SQL_CREATETAB_AUTH	CREATETAB authority.
SQL_CREATET_NOT_FENC_AUTH	CREATE_NOT_FENCED authority.
SQL_BINDADD_AUTH	BINDADD authority.
SQL_CONNECT_AUTH	CONNECT authority.
SQL_IMPLICIT_SCHEMA_AUTH	IMPLICIT_SCHEMA authority.
SQL_LOAD_AUTH	LOAD authority.
SQL_SYSADM_GRP_AUTH	User belongs to a group which holds SYSADM authority.
SQL_SYSCTRL_GRP_AUTH	User belongs to a group which holds SYSCTRL authority.
SQL_SYSMAINT_GRP_AUTH	User belongs to a group which holds SYSMAINT authority.
SQL_DBADM_GRP_AUTH	User belongs to a group which holds DBADM authority.
SQL_CREATETAB_GRP_AUTH	User belongs to a group which holds CREATETAB authority.
SQL_CREATE_NON_FENC_GRP_AUTH	User belongs to a group which holds CREATE_NOT_FENCED authority.
SQL_BINDADD_GRP_AUTH	User belongs to a group which holds BINDADD authority.
SQL_CONNECT_GRP_AUTH	User belongs to a group which holds CONNECT authority.
SQL_IMPLICIT_SCHEMA_GRP_AUTH	User belongs to a group which holds IMPLICIT_SCHEMA authority.
SQL_LOAD_GRP_AUTH	User belongs to a group which holds LOAD authority.
Note: SYSADM, SYSMAINT, and SYSCTRL are only indirect authorities and cannot be granted directly to the user. They are available only through the groups to which the user belongs.	

Language syntax:

C Structure

```
/* File: sqlutil.h */
/* Structure: SQL-AUTHORIZATIONS */
/* ... */
SQL_STRUCTURE sql_authorizations
{
  short           sql_authorizations_len;
  short           sql_sysadm_auth;
  short           sql_dbadm_auth;
  short           sql_createtab_auth;
  short           sql_bindadd_auth;
  short           sql_connect_auth;
  short           sql_sysadm_grp_auth;
  short           sql_dbadm_grp_auth;
  short           sql_createtab_grp_auth;
  short           sql_bindadd_grp_auth;
  short           sql_connect_grp_auth;
  short           sql_sysctrl_auth;
  short           sql_sysctrl_grp_auth;
  short           sql_sysmaint_auth;
  short           sql_sysmaint_grp_auth;
  short           sql_create_not_fenc_auth;
  short           sql_create_not_fenc_grp_auth;
```

```
  short           sql_implicit_schema_auth;
  short           sql_implicit_schema_grp_auth;
  short           sql_load_auth;
  short           sql_load_grp_auth;
};
/* ... */
```

COBOL Structure

```
* File: sqlutil.cbl
01 SQL-AUTHORIZATIONS.
    05 SQL-AUTHORIZATIONS-LEN PIC S9(4) COMP-5.
    05 SQL-SYSADM-AUTH        PIC S9(4) COMP-5.
    05 SQL-DBADM-AUTH         PIC S9(4) COMP-5.
    05 SQL-CREATETAB-AUTH     PIC S9(4) COMP-5.
    05 SQL-BINDADD-AUTH       PIC S9(4) COMP-5.
    05 SQL-CONNECT-AUTH       PIC S9(4) COMP-5.
    05 SQL-SYSADM-GRP-AUTH    PIC S9(4) COMP-5.
    05 SQL-DBADM-GRP-AUTH     PIC S9(4) COMP-5.
    05 SQL-CREATETAB-GRP-AUTH PIC S9(4) COMP-5.
    05 SQL-BINDADD-GRP-AUTH   PIC S9(4) COMP-5.
    05 SQL-CONNECT-GRP-AUTH   PIC S9(4) COMP-5.
    05 SQL-SYSCTRL-AUTH       PIC S9(4) COMP-5.
    05 SQL-SYSCTRL-GRP-AUTH   PIC S9(4) COMP-5.
    05 SQL-SYSMAINT-AUTH      PIC S9(4) COMP-5.
    05 SQL-SYSMAINT-GRP-AUTH  PIC S9(4) COMP-5.
    05 SQL-CREATE-NOT-FENC-AUTH PIC S9(4) COMP-5.
    05 SQL-CREATE-NOT-FENC-GRP-AUTH PIC S9(4) COMP-5.
    05 SQL-IMPLICIT-SCHEMA-AUTH PIC S9(4) COMP-5.
    05 SQL-IMPLICIT-SCHEMA-GRP-AUTH PIC S9(4) COMP-5.
    05 SQL-LOAD-AUTH PIC S9(4) COMP-5.
    05 SQL-LOAD-GRP-AUTH PIC S9(4) COMP-5.
*
```

Related reference:

- "sqluadau - Get Authorizations" on page 510

Part 3. Security Plug-Ins

Only the default IBM-supplied operating-system based authentication and group plug-ins are supported in Common Criteria compliant environments. User-written or third-party plug-ins are not supported. In addition, Kerberos-based authorization is not supported. The sections that follow are for informational purposes only.

© Copyright IBM Corp. 1993-2004

Chapter 28. Security plug-ins

Security plug-ins

Authentication in DB2® is done using *security plug-ins*. A security plug-in is a dynamically-loadable library that DB2 loads to provide the following functionality:

- Group retrieval plug-in: retrieves group membership information for a given user
- Client authentication plug-in: manages authentication on a DB2 client.
- Server authentication plug-in: manages authentication on a DB2 server.

DB2 supports two mechanisms for plug-in authentication:

- Authentication using a user ID and password, which is known as user ID/password authentication. The authentication types CLIENT, SERVER, SERVER_ENCRYPT, DATA_ENCRYPT, and DATA_ENCRYPT_CMP determine how and where authentication of a user occurs. The authentication type used depends on the authentication type specified by the *authentication* database manager configuration parameter. These authentication types are all implemented using user ID/password authentication plug-ins.
- Authentication using GSS-API, formally known as *Generic Security Service Application Program Interface, Version 2* (IETF RFC2743) and *Generic Security Service API Version 2: C-Bindings* (IETF RFC2744). Kerberos authentication is also implemented using GSS-API. The authentication types KERBEROS, GSSPLUGIN, KRB_SERVER_ENCRYPT, and GSS_SERVER_ENCRYPT use GSS-API authentication plug-ins. KRB_SERVER_ENCRYPT and GSS_SERVER_ENCRYPT support both GSS-API authentication and user ID/password authentication; however, GSS-API authentication is the preferred authentication type.

Each of the plug-ins can be used independently or in conjunction with one or more of the other plug-ins. For example, you might only use a server authentication plug-in and assume the DB2 defaults for client and group authentication. Alternatively, you might have only a group or client authentication plug-in. The only situation where both a client and server plug-in are required is for GSS-API authentication plug-ins.

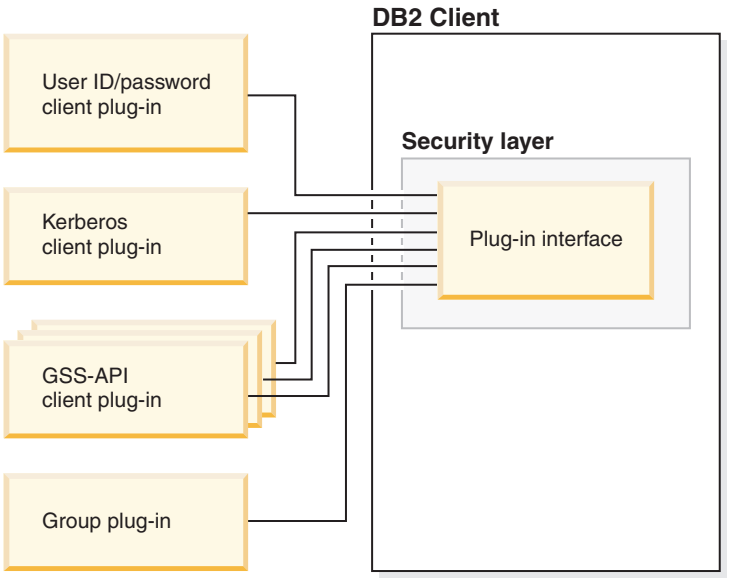
In DB2 Universal Database for Linux, UNIX, and Windows version 8.2, the default behavior is to use a user ID/password plug-in that implements an operating-system-level mechanism for authentication. In all previous releases of DB2, the default behavior is to directly use operating-system-level authentication without a plug-in implementation. In DB2 Universal Database for Linux, UNIX, and Windows version 8.2, client-side Kerberos support is available on Solaris, AIX®, Windows®, and IA32 Linux operating systems; however, it is only enabled by default on Windows.

© Copyright IBM Corp. 1993-2004

DB2 includes sets of plug-ins for group retrieval, user ID/password authentication, and for Kerberos authentication. With the security plug-in architecture you can customize DB2's authentication behavior by either developing your own plug-ins, or buying plug-ins from a third party.

Deployment of security plug-ins on DB2 clients:

DB2 clients can support one group plug-in, one user ID/password authentication plug-in, and will negotiate with the DB2 server for a particular GSS-API plug-in. This negotiation consists of a scan by the client of the DB2 server's list of implemented GSS-API plug-ins for the first authentication plug-in name that matches an authentication plug-in implemented on the client. The server's list of plug-ins is a user-specified database manager configuration parameter value that contains the names of all of the plug-ins that are implemented on the server. The following figure portrays the security plug-in infrastructure on a DB2 client.

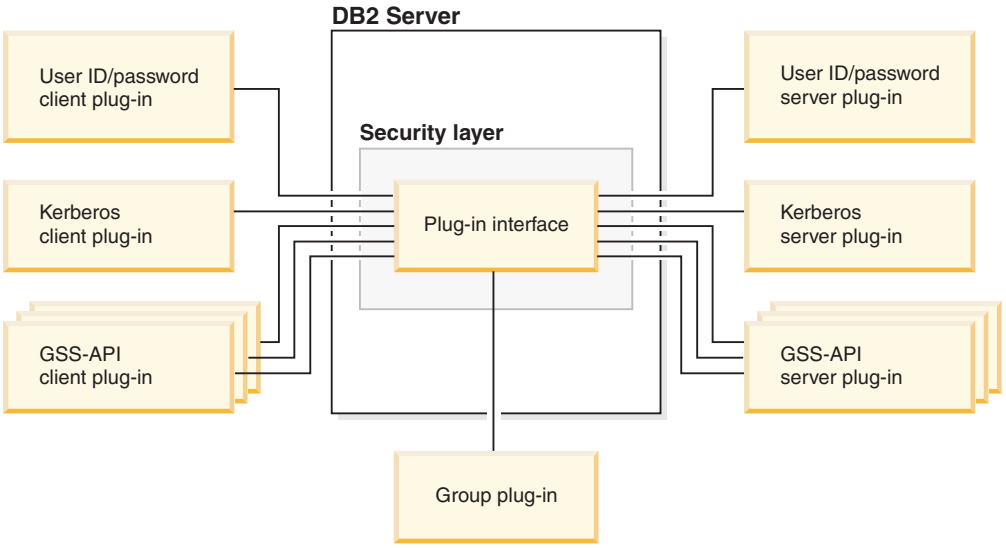


Deployment of security plug-ins on DB2 servers:

DB2 servers can support one group plug-in, one user ID/password authentication plug-in, and multiple GSS-API plug-ins. The multiple GSS-API plug-ins are named in a database manager configuration parameter value as a list. Only one GSS-API plug-in in this list can be a Kerberos plug-in.

In addition to server-side security plug-ins, you might also need to deploy client authorization plug-ins on your database server. When you run instance-level operations like `db2start` and `db2trc`, DB2 performs authorization checking for the operation using client authentication plug-ins. Therefore, you should install the client authentication plug-in that corresponds to the server plug-in that is specified by the *authentication* database manager configuration parameter. If you do not use client authentication plug-ins on the database server, instance level operations such as `db2start` will fail. For example, if the authentication type is SERVER and no user-supplied client plug-in is used, DB2 will use the IBM-shipped default client operating-system plug-in. The following figure portrays the security plug-in

infrastructure on a DB2 server.



Enabling security plug-ins:

The system administrator can specify the names of the plug-ins to use for each authentication mechanism by updating certain plug-in-related database manager configuration parameters. If these parameters are null, they will default to the DB2-supplied plug-ins for group retrieval, user ID/password management, or Kerberos (if authentication is set to Kerberos -- on the server). However, DB2 does not provide a default GSS-API plug-in. Therefore, if the system administrator specifies an authentication type of GSSPLUGIN in *authentication*, she must also specify a GSS-API authentication plug-in in *srvcon_gssplugin_list*.

How DB2 loads security plug-ins:

All of the supported plug-ins identified by the database manager configuration parameters are loaded when the database manager starts.

The DB2 client will load an appropriate plug-in based on the security mechanism negotiated with the server during connect or attach operations. It is possible that a client application can cause multiple security plug-ins to be concurrently loaded and used. This situation can occur, for example, in a threaded program that has concurrent connections to different databases from different instances.

For other actions that require authorization (such as updating the database manager configuration, starting and stopping the database manager, turning DB2 trace on and off), the DB2 client program will load a plug-in specified in another database manager configuration parameter. If *authentication* is set to GSSPLUGIN, DB2 will use the plug-in specified by *local_gssplugin*. If *authentication* is set to KERBEROS, DB2 will use the plug-in specified by *clnt_krb_plugin*. Otherwise, DB2 will use the plug-in specified by *clnt_pw_plugin*.

Developing security plug-ins:

If you are developing a security plug-in, you need to implement the standard authentication functions that DB2 will invoke. For the available types of plug-ins, the functionality you will need to implement is as follows:

Group retrieval
Gets the list of groups to which a user belongs.

User ID/password authentication
Identifies the default security context (client only), validates and optionally changes a password, determines if a given string represents a valid user (server only), modifies the user ID or password provided on the client before it is sent to the server (client only), returns the DB2 authorization ID associated with a given user.

GSS-API authentication
Implements the required GSS-API functions, identifies the default security context (client only), generates initial credentials based on a user ID and password and optionally changes password (client only), creates and accepts security tickets, and returns the DB2 authorization ID associated with a given GSS-API security context.

Related concepts:

- "Authentication methods for your server" on page 38
- "Security plug-in library locations" on page 1024
- "How DB2 loads security plug-ins" on page 1037
- "Security plug-in APIs" on page 1047

Related reference:

- "authentication - Authentication type" on page 783
- "srvcon_auth - Authentication type for incoming connections at the server" on page 1087
- "Security plug-in samples" in the *Application Development Guide: Building and Running Applications*

Security plug-in library locations

After you acquire your security plug-ins (by developing them yourself, or purchasing them from a third party), copy them to specific locations on your database server.

DB2® looks for client-side user authentication plug-ins in the following directory:

- UNIX® 32-bit: `$DB2PATH/security32/plugin/client`
- UNIX 64-bit: `$DB2PATH/security64/plugin/client`
- WINDOWS 32-bit and 64-bit: `$DB2PATH\security\plugin\`*`instance name`*`\client`

Note: On Windows®-based platforms, the subdirectories *instance name* and `client` are not created automatically. The instance owner has to manually create them.

DB2 will look for server-side user authentication plug-ins in the following directory:

- UNIX 32-bit: `$DB2PATH/security32/plugin/server`
- UNIX 64-bit: `$DB2PATH/security64/plugin/server`
- WINDOWS 32-bit and 64-bit: `$DB2PATH\security\plugin\`*`instance name`*`\server`

Note: On Windows-based platforms, the subdirectories *instance name* and `server` are not created automatically. The instance owner has to manually create them.

DB2 will look for group plug-ins in the following directory:

- UNIX 32-bit: `$DB2PATH/security32/plugin/group`
- UNIX 64-bit: `$DB2PATH/security64/plugin/group`
- WINDOWS 32-bit and 64-bit: `$DB2PATH\security\plugin\`*`instance name`*`\group`

Note: On Windows-based platforms, the subdirectories *instance name* and `group` are not created automatically. The instance owner has to manually create them.

Related concepts:

- "Security plug-ins" on page 1021
- "How DB2 loads security plug-ins" on page 1037

Related tasks:

- "Deploying a group retrieval plug-in" on page 1030
- "Deploying a user ID/password plug-in" on page 1031
- "Deploying a GSS-API plug-in" on page 1033
- "Deploying a Kerberos plug-in" on page 1034

Related reference:

- "Restrictions on security plug-in libraries" on page 1038

Security plug-in naming conventions

The security plug-in libraries must have the appropriate file name extension for each individual platform. By operating system these extensions are as follows:

- Windows®: `.DLL`
- AIX®: `.a`
- Linux, HP IPF and Solaris Operating Environment: `.so`
- HPUX on PA-RISC: `.sl`

For example, assume you have a security plug-in library called `MyPlugin`. For each supported operating system, the appropriate library file name follows:

- Windows 32-bit: MyPlugin.dll
- Windows 64-bit: MyPlugin64.dll
- AIX 32 or 64-bit: MyPlugin.a
- SUN 32 or 64-bit, Linux 32 or 64 bit, HP 32 or 64 bit on IPF: MyPlugin.so
- HP-UX 32 or 64-bit on PA-RISC: MyPlugin.sl

Note: The suffix "64" is only required on the library name for 64-bit Windows security plug-ins.

When you update the database manager configuration with the name of a security plug-in, use the full name of the library without the "64" suffix and omit both the file extension and any qualified path portion of the name. Regardless of the operating system, the security plug-in library called `MyPlugin` would be registered as follows:

```
UPDATE DBM CFG USING CLNT_PW_PLUGIN MyPlugin
```

The security plug-in name is case sensitive, and must exactly match the library name. DB2® uses the value from the relevant database manager configuration parameter to assemble the library path, and then uses the library path to load the security plug-in library.

To avoid security plug-in name conflicts, you should name the plug-in using the authentication method used, and an identifying symbol of the firm that wrote the plug-in. For instance, if the company `Foo, Inc.` wrote a plug-in implementing the authentication method `somemethod`, the plug-in could have a name like `FOOsomemethod.DLL`.

The maximum length of a plug-in name (not including the file extension and the ″64″ suffix) is limited to 32 bytes. There is no maximum number of plug-ins supported by the database server, but the maximum length of the comma-separated list of plug-ins in the database manager configuration is 255 bytes. Two defines located in the include file `sqlenv.h` establish these two limits:

```
#define SQL_PLUGIN_NAME_SZ     32     /* plug-in name */
#define SQL_SRVCON_GSSPLUGIN_LIST_SZ 255 /* GSS API plug-in list */
```

The security plug-in library files must have the following file permissions:

- Owned by the instance owner.
- Readable by all users on the system.
- Executable by all users on the system.

Related concepts:

- “Configuration parameters” on page 769
- “Security plug-ins” on page 1021
- “Security plug-in library locations” on page 1024

Related tasks:

- “Configuring DB2 with configuration parameters” on page 779
- “Deploying a group retrieval plug-in” on page 1030
- “Deploying a user ID/password plug-in” on page 1031
- “Deploying a GSS-API plug-in” on page 1033
- “Deploying a Kerberos plug-in” on page 1034

Related reference:

- “UPDATE DATABASE MANAGER CONFIGURATION” on page 384
- “clnt_krb_plugin - Client Kerberos plug-in” on page 1085
- “clnt_pw_plugin - Client userid-password plug-in” on page 1085

Security plug-in support for two-part user IDs

DB2® UDB for Linux, UNIX®, and Windows® supports the use of two-part user IDs, and the mapping of two-part user IDs to two-part authorization IDs.

For example, consider a Windows operating system two-part user ID composed of a domain and user ID such as: `MEDWAY\pieter`. In this example, `MEDWAY` is a domain and `pieter` is the user name. In DB2, you can specify whether this two-part user ID should be mapped to either a one-part authorization ID or a two-part authorization ID.

In DB2, prior to version 8.2, you could only have a one-part user ID that mapped to a one-part authorization ID. In DB2 Version 8.2, by default, one-part user IDs map to one-part authorization IDs and two-part user IDs map to one-part authorization IDs. The mapping of a two-part user ID to a two-part authorization ID is supported, but is not the default behavior.

The default mapping of a two-part user ID to a one-part user ID allows a user to connect to the database using:

```
db2 connect to db user MEDWAY\pieter using pw
```

In this situation, if the default behavior is used, the user ID `MEDWAY\pieter` is resolved to the authorization ID `PETER`. If the support for mapping a two-part user ID to a two-part authorization ID is enabled, the authorization ID would be `MEDWAY\PETER`.

To enable DB2 to map two-part user IDs to two-part authorization IDs, specify the appropriate plug-in when updating the database manager configuration. (The relevant database manager configuration parameters are described below.)

DB2 supplies two sets of authentication plug-ins. One set is exclusively for mapping user IDs to one-part authorization IDs; that is for mapping a one-part user ID to a one-part authorization ID and mapping a two-part user ID to a one-part authorization ID. The second set maps a one-part user ID to a one-part authorization ID, and a two-part user ID to a two-part authorization ID.

If a user name in your work environment can be mapped to multiple accounts defined in different locations (such as local account, domain account, and trusted domain accounts), you may want to specify the plug-ins that enable two-part authorization ID mapping.

It is important to note that a one-part authorization ID, such as, `PETER` and a two-part authorization ID that combines a domain and a user ID like `MEDWAY\peter` are functionally distinct authorization IDs. The set of privileges associated with one of these authorization IDs is completely distinct from the set of privileges associated with the other authorization ID. Care should be taken when working with one-part and two-part authorization IDs.

The following table identifies the kinds of plug-ins supplied by DB2, and the plug-in names for the specific authentication implementations.

Table 82. DB2 security plug-ins

Authentication type	Name of one-part user ID plug-in	Name of two-part user ID plug-in
User ID/password (client)	IBMOSauthclient	IBMOSauthclientTwoPart
User ID/password (server)	IBMOSauthserver	IBMOSauthserverTwoPart
Kerberos	IBMkrb5	IBMkrb5TwoPart

Note: On Windows 64-bit platforms, the characters ″64″ are appended to the plug-in names listed here.

To map a two-part user ID to a two-part authorization ID, you must specify that the two-part plug-in, which is the non-default plug-in, be used. Security plug-ins are specified at the instance level by setting the security related database manager configuration parameters as follows:

For server authentication that maps two-part user IDs to two-part authorization IDs, you must set:

- `srvcon_pw_plugin` to `IBMOSauthserverTwoPart`
- `clnt_pw_plugin` to `IBMOSauthclientTwoPart`

For client authentication that maps two-part user IDs to two-part authorization IDs, you must set:

- `srvcon_pw_plugin` to `IBMOSauthserverTwoPart`
- `clnt_pw_plugin` to `IBMOSauthclientTwoPart`

For Kerberos authentication that maps two-part user IDs to two-part authorization IDs, you must set:

- `srvcon_gssplugin_list` to `IBMOSkrb5TwoPart`
- `clnt_krb_plugin` to `IBMkrb5TwoPart`

The security plug-in libraries accept two-part user IDs specified in a Microsoft® Windows Security Account Manager compatible format. For example, in the format: *domain\user ID*. Both the domain and user ID information will be used by the DB2 authentication and authorization processes at connection time.

When you specify an authentication type that requires a user ID/password or Kerberos plug-in, the plug-ins that are listed in the ″Name of one-part user ID plug-in″ column in the previous table are used by default.

You should consider implementing the two-part plug-ins when creating new databases to avoid conflicts with one-part authorization IDs in existing databases. New databases that use two-part authorization IDs authentication must be created in a separate instance from databases that use single-part authorization IDs.

Related concepts:

- “DB2 for Windows NT and Windows NT security introduction” in the *Administration Guide: Implementation*

Related tasks:

- “DB2 for Windows NT authentication with groups and domain security” in the *Administration Guide: Implementation*

Related reference:

- “clnt_pw_plugin - Client userid-password plug-in” on page 1085
- “srvcon_pw_plugin - Userid-password plug-in for incoming connections at the server” on page 1088

32-bit and 64-bit considerations for security plug-ins

In general, a 32-bit DB2® instance will use the 32-bit security plug-in and 64-bit DB2 instance will use the 64-bit security plug-in. However, on a 64-bit instance, DB2 supports 32-bit applications, which will require the 32-bit plug-in library.

A database instance where both the 32-bit and the 64-bit applications can run is known as a hybrid instance. If you have a hybrid instance and intend to run 32-bit applications, ensure that the required 32-bit security plug-ins are available in the 32-bit plug-in directory. For hybrid DB2 instances on a UNIX® operating system, the directories `security32` and `security64` appear. For a Windows® 64-bit hybrid

instance, both 32-bit and 64-bit security plug-ins are located in the same directory, but 64-bit plug-in names have a suffix, "64".

If you want to migrate from a 32-bit instance to a 64-bit instance, you should obtain versions of your security plug-ins that are recompiled for 64-bit.

If you acquired your security plug-ins from a vendor that does not supply 64-bit plug-in libraries, you can implement a 64-bit stub that executes a 32-bit application. In this situation, the security plug-in is an external program rather than a library.

Related concepts:

- "Security plug-ins" on page 1021
- "Security plug-in library locations" on page 1024

Related tasks:

- "Migrating applications from 32-bit to 64-bit environments" in the *Application Development Guide: Building and Running Applications*

Security plug-in problem determination

Problems with security plug-ins are reported in two ways: through SQL errors and through the administrative log.

Following are the SQLCODE values related to security plug-ins:

- SQLCODE -1365 is returned when a plug-in error occurs during `db2start` or `db2stop`.
- SQLCODE -1366 is returned whenever there is a local authorization problem.
- SQLCODE -30082 is returned for all connection-related plug-in errors.

The administrative log is a good resource for debugging and administrating security plug-ins. To see the administrative log on UNIX®, check `sqllib/db2dump/`*`instance name`*`.nfy`. To see the administrative log on Windows operating systems, use the Event Viewer tool. The Event Viewer tool can be found by navigating from the Windows operating system "Start" button to `Settings -> Control Panel -> Administrative Tools -l> Event Viewer`. Following are the administration log values related to security plug-ins:

- 13000 indicates that a call to a GSS-API security plug-in API failed with an error, and returned an optional error message.

  ```
  SQLT_ADMIN_GSS_API_ERROR (13000)
  Plug-in "plug-in name" received error code "error code" from
  GSS API  "gss api name" with the error message "error message"
  ```

- 13001 indicates that a call to a DB2® security plug-in API failed with an error, and returned an optional error message.

  ```
  SQLT_ADMIN_PLUGIN_API_ERROR(13001)
  Plug-in "plug-in name" received error code "error code" from DB2
  security plug-in API "gss api name" with the error message
  "error message"
  ```

- 13002 indicates that DB2 failed to unload a plug-in.

  ```
  SQLT_ADMIN_PLUGIN_UNLOAD_ERROR (13002)
  Unable to unload plug-in "plug-in name". No further action required.
  ```

- 13003 indicates a bad principal name.

```
SQLT_ADMIN_INVALID_PRIN_NAME (13003)
The principal name "principal name" used for "plug-in name"
is invalid. Fix the principal name.
```

- 13004 indicates that the plug-in name is not valid. Path separators (On UNIX "/" and on Windows® "\") are not allowed in the plug-in name.

  ```
  SQLT_ADMIN_INVALID_PLGN_NAME (13004)
  The plug-in name "plug-in name" is invalid. Fix the plug-in name.
  ```

- 13005 indicates that the security plug-in failed to load. Ensure the plug-in is in the correct directory and that the appropriate database manager configuration parameters are updated.

  ```
  SQLT_ADMIN_PLUGIN_LOAD_ERROR (13005)
  Unable to load plug-in "plug-in name". Verify the plug-in existence and
  directory where it is located is correct.
  ```

- 13006 indicates that an unexpected error was encountered by a security plug-in. Gather all the `db2support` information, if possible capture a `db2trc`, and then call IBM® support for further assistance.

  ```
  SQLT_ADMIN_PLUGIN_UNEXP_ERROR (13006)
  Plug-in encountered unexpected error. Contact IBM Support for further assistance.
  ```

Note: If you are using security plug-ins on a Windows 64-bit database server and are seeing a load error for a security plug-in, see the topics "32-bit and 64-bit considerations for security plug-ins" and "Security plug-in naming conventions". The 64-bit plug-in library requires the suffix "64" on the library name, but the entry in the security plug-in database manager configuration parameters should not indicate this suffix.

Related concepts:

- "Event monitors" in the *System Monitor Guide and Reference*
- "Error Information in the SQLCODE, SQLSTATE, and SQLWARN Fields" in the *Application Development Guide: Programming Client Applications*
- "SQLSTATE and SQLCODE Variables in C and C++" in the *Application Development Guide: Programming Client Applications*
- "SQLCODE and SQLSTATE Differences among IBM Relational Database Systems" in the *Application Development Guide: Programming Client Applications*
- "DB2 trace (db2trc)" in the *Troubleshooting Guide*
- "Security plug-ins" on page 1021
- "32-bit and 64-bit considerations for security plug-ins" on page 1028
- "Security plug-in APIs" on page 1047

Related reference:

- "CREATE EVENT MONITOR statement" in the *SQL Reference, Volume 2*
- "db2trc - Trace Command" in the *Command Reference*
- "Error messages for security plug-ins" on page 1042
- "Required APIs and Definitions for GSS-API authentication plug-ins" on page 1080
- "APIs for group retrieval plug-ins" on page 1048
- "APIs for user ID/password authentication plug-ins" on page 1057

Deploying a group retrieval plug-in

If you want to customize the DB2 security system's group retrieval behavior, you can develop your own group retrieval plug-in or buy one from a third party.

After you acquire a group retrieval plug-in that is suitable for your database management system, you can deploy it.

Procedure:

To deploy a group retrieval plug-in on the database server, perform the following steps:

1. Place the group retrieval plug-in library in the server's group plug-in directory.
2. Update the database manager configuration parameter *group_plugin* with the name of the plug-in.

To deploy a group retrieval plug-in on database clients, perform the following steps:

1. Place the group retrieval plug-in library in the client's group plug-in directory.
2. On the database client, update the database manager configuration parameter *group_plugin* with the name of the plug-in.

Related concepts:

- "Security plug-ins" on page 1021
- "Security plug-in library locations" on page 1024
- "Security plug-in naming conventions" on page 1025

Related tasks:

- "Deploying a user ID/password plug-in" on page 1031
- "Deploying a GSS-API plug-in" on page 1033
- "Deploying a Kerberos plug-in" on page 1034

Related reference:

- "group_plugin - Group plug-in" on page 1086

Deploying a user ID/password plug-in

If you want to customize the DB2 security system's user ID/password authentication behavior, you can develop your own user ID/password authentication plug-ins or buy one from a third party.

After you acquire user ID/password authentication plug-ins that are suitable for your database management system, you can deploy them.

Depending on their intended usage, all user ID-password based authentication plug-ins must be placed in either the client plug-in directory or the server plug-in directory. If a plug-in is placed in the client plug-in directory, it will be used both for local authorization checking and for validating the client when it attempts to connect with the server. If the plug-in is placed in the server plug-in directory, it will be used for handling incoming connections to the server and for checking whether an authid exists and is valid whenever the GRANT statement is issued without specifying either the keyword USER or GROUP.

In most situations, user ID/password authentication requires only a server-side plug-in. It is possible, though generally deemed less useful, to have only a client user ID/password plug-in. It is possible, though quite unusual to require matching user ID/password plug-ins on both the client and the server.

Procedure:

To deploy a user ID/password authentication plug-in on the database server, perform the following steps:

1. Place the user ID/password authentication plug-in library in the server's plug-in directory.
2. Update the database manager configuration parameter *srvcon_pw_plugin* with the name of the server plug-in.

   This plug-in is used by the server when it is handling CONNECT and ATTACH requests.
3. Either:
   - Set the database manager configuration parameter *srvcon_auth* to the CLIENT, SERVER, SERVER_ENCRYPT, DATA_ENCRYPT, or DATA_ENCRYPT_CMP authentication type. Or:
   - Set the database manager configuration parameter *srvcon_auth* to NOT_SPECIFIED and set *authentication* to CLIENT, SERVER, SERVER_ENCRYPT, DATA_ENCRYPT, or DATA_ENCRYPT_CMP authentication type.

To deploy a user ID/password authentication plug-in on database clients, perform the following steps:

1. Place the user ID/password authentication plug-in library in the client plug-in directory on the client.
2. Update the database manager configuration parameter *clnt_pw_plugin* with the name of the client plug-in.

   This plug-in is loaded and called regardless of where the authentication is being done, that is, not only when the database configuration parameter, *authentication* is set to CLIENT.

For local authorization on a client, server, or gateway, using a user ID/password authentication plug-in, perform the following steps:

1. Place the user ID/password authentication plug-in library in the client plug-in directory on the client, server, or gateway.
2. Update the database manager configuration parameter *clnt_pw_plugin* with the name of the plug-in.
3. Set the *authentication* database manager configuration parameter to CLIENT, SERVER, SERVER_ENCRYPT, DATA_ENCRYPT, or DATA_ENCRYPT_CMP.

Related concepts:

- "Security plug-ins" on page 1021
- "Security plug-in library locations" on page 1024

Related tasks:

- "Deploying a group retrieval plug-in" on page 1030
- "Deploying a GSS-API plug-in" on page 1033

Related reference:

- "authentication - Authentication type" on page 783
- "GRANT (Database Authorities)" on page 700
- "clnt_pw_plugin - Client userid-password plug-in" on page 1085

- "srvcon_auth - Authentication type for incoming connections at the server" on page 1087
- "srvcon_pw_plugin - Userid-password plug-in for incoming connections at the server" on page 1088

Deploying a GSS-API plug-in

If you want to customize the DB2 security system's authentication behavior, you can develop your own authentication plug-ins using the GSS-API, or buy one from a third party.

After you acquire GSS-API authentication plug-ins that are suitable for your database management system, you can deploy them.

When using GSS-API or Kerberos plug-ins, you must have matching plug-in types on the client and the server. The plug-ins on the client and server need not be from the same vendor, but they must generate and consume compatible GSS-API tokens. For example, any combination of Kerberos plug-ins deployed on the client and the server is supported because Kerberos plug-ins are standardized; however, different implementations of less standardized GSS-API mechanisms, such as *x.509* certificates might not be completely compatible.

Depending on the intended usage, all GSS-API authentication plug-ins must be placed in either the client plug-in directory or the server plug-in directory. If a plug-in is placed in the client plug-in directory, it will be used for local authorization checking and when a client attempts to connect with the server. If the plug-in is placed in the server plug-in directory, it will be used for handling incoming connections to the server and for checking whether an authid exists and is valid whenever the GRANT statement is issued without specifying either the keyword USER or GROUP.

Procedure:

To deploy a GSS-API authentication plug-in on the database server, perform the following steps:

1. Place the GSS-API authentication plug-in library in the server plug-in directory on the server. You can copy numerous GSS-API plug-ins into this directory.
2. Update the database manager configuration parameter *srvcon_gssplugin_list* with an ordered, comma-delimited list of the names of the plug-ins installed in the GSS-API plug-in directory.
3. Either:
   - Set the database manager configuration parameter *srvcon_auth* to GSSPLUGIN. Or:
   - Set the database manager configuration parameter *srvcon_auth* to NOT_SPECIFIED and set *authentication* to GSSPLUGIN.

To deploy a GSS-API authentication plug-in on database clients, perform the following steps:

1. Place the GSS-API authentication plug-in library in the client plug-in directory on the client. You can copy numerous GSS-API plug-ins into this directory. The client selects the appropriate GSS-API plug-in for authentication during a CONNECT or ATTACH operation by picking the first GSS-API plug-in contained in the server's plug-in list that is available on the client.

2. Optional: Catalog the databases that the client will access, indicating that the client will only accept a GSS-API authentication plug-in as the authentication mechanism. For example:

   ```
   CATALOG DB testdb AT NODE testnode AUTHENTICATION GSSPLUGIN
   ```

For local authorization on a client, server, or gateway using a GSS-API authentication plug-in, perform the following steps:

1. Place the GSS-API authentication plug-in library in the client plug-in directory on the client, server, or gateway.
2. Update the database manager configuration parameter *local_gssplugin* with the name of the plug-in.
3. Set the *authentication* database manager configuration parameter to GSSPLUGIN, or GSS_SERVER_ENCRYPT.

Related concepts:

- "Security plug-ins" on page 1021
- "Security plug-in library locations" on page 1024

Related reference:

- "authentication - Authentication type" on page 783
- "CATALOG DATABASE" on page 249
- "local_gssplugin - GSS API plug-in used for local instance level authorization" on page 1086
- "srvcon_auth - Authentication type for incoming connections at the server" on page 1087
- "srvcon_gssplugin_list - List of GSS API plug-ins for incoming connections at the server" on page 1087
- "Security plug-in samples" in the *Application Development Guide: Building and Running Applications*

Deploying a Kerberos plug-in

If you want to customize the DB2 security system's Kerberos authentication behavior, you can develop your own Kerberos authentication plug-ins or buy one from a third party.

After you acquire Kerberos authentication plug-ins that are suitable for your database management system, you can deploy them.

Procedure:

To deploy a Kerberos authentication plug-in on the database server, perform the following steps:

1. Place the Kerberos authentication plug-in library in the plug-in directory on the server.
2. Update the database manager configuration parameter *srvcon_gssplugin_list*, which is presented as an ordered, comma delimited list, to include the Kerberos server plug-in name. Only one plug-in in this list can be a Kerberos plug-in.

   If this list is blank and *authentication* is set to KERBEROS or KRB_SVR_ENCRYPT, the default DB2 Kerberos plug-in: IBMkrb5 will be used.
3. Either:

- Set the database manager configuration parameter *srvcon_auth* to the KERBEROS or KRB_SERVER_ENCRYPT authentication type. Or:
- Set the database manager configuration parameter *srvcon_auth* to NOT_SPECIFIED and set *authentication* to KERBEROS or KRB_SERVER_ENCRYPT authentication type.

To deploy a Kerberos authentication plug-in on database clients, perform the following steps:

1. Place the Kerberos authentication plug-in library in the client plug-in directory on the client.
2. Update the database manager configuration parameter *clnt_krb_plugin* with the name of the Kerberos plug-in.

   If *clnt_krb_plugin* is blank, DB2 assumes that the client cannot use Kerberos authentication. This setting is only appropriate when the server cannot support plug-ins. See the limitations on the use of security plug-ins for more information. If both the server and the client support security plug-ins, the client will not use the value of *clnt_krb_plugin* because the server has a GSS-API plug-in with the name *IBMkrb5* listed.

   For local authorization on a client, server, or gateway using a Kerberos authentication plug-in, perform the following steps:

   a. Place the Kerberos authentication plug-in library in the client plug-in directory on the client, server, or gateway.
   b. Update the database manager configuration parameter *clnt_krb_plugin* with the name of the plug-in.
   c. Set the *authentication* database manager configuration parameter to KERBEROS, or KRB_SERVER_ENCRYPT.

   The Kerberos plug-in provided by DB2 is named `IBMkrb5`.
3. Optional: Catalog the databases that the client will access, indicating that the client will only use a Kerberos authentication plug-in. For example:

```
CATALOG DB testdb AT NODE testnode AUTHENTICATION KERBEROS
        TARGET PRINCIPAL service/host@REALM
```

Note: For platforms supporting Kerberos, the `IBMkrb5` library will be present in the client plug-in directory. DB2 will recognize this library as a valid GSS-API plug-in, because Kerberos plug-ins are implemented using GSS-API plug-in.

Related concepts:

- "Security plug-ins" on page 1021
- "Security plug-in library locations" on page 1024

Related tasks:

- "Deploying a group retrieval plug-in" on page 1030
- "Deploying a user ID/password plug-in" on page 1031
- "Deploying a GSS-API plug-in" on page 1033

Related reference:

- "authentication - Authentication type" on page 783
- "CATALOG DATABASE" on page 249
- "clnt_krb_plugin - Client Kerberos plug-in" on page 1085
- "srvcon_auth - Authentication type for incoming connections at the server" on page 1087

- "srvcon_gssplugin_list - List of GSS API plug-ins for incoming connections at the server" on page 1087

Chapter 29. Developing security plug-ins

How DB2 loads security plug-ins

Each plug-in library must contain an initialization function with a specific name determined by the plug-in type:

- Server side authentication plug-in: db2secServerAuthPluginInit()
- Client side authentication plug-in: db2secClientAuthPluginInit()
- Group plug-in: db2secGroupPluginInit()

This function is known as the plug-in initialization function. The plug-in initialization function initializes the specified plug-in and provides DB2® with information that it requires to call the plug-in's functions. The plug-in initialization function accepts the following parameters:

- The highest version number of the functions pointer structure that DB2 can support
- A pointer to a structure containing pointers to all the APIs requiring implementation
- A pointer to a function that adds log messages to the db2diag.log file
- A pointer to an error message string
- The Length of the error message

The following is a function signature for the initialization function of a group retrieval plug-in:

```
SQL_API_RC SQL_API_FN db2secGroupPluginInit(
  db2int32 version,
  void *group_fns,
  db2secLogMessage *logMessage_fn,
  char **errormsg,
  db2int32  *errormsglen);
```

Note: Plug-in libraries can only be implemented in C or C++. If the plug-in library is compiled as C++, all functions must be declared with: `extern "C"`. DB2 relies on the underlying operating system dynamic loader to handle the C++ constructors and destructors used inside of a C++ user-written plug-in library.

The initialization function is the only function in the plug-in library that uses a prescribed function name. The other plug-in functions are referenced through function pointers returned from the initialization function. Server plug-ins are loaded when the DB2 server starts. Client plug-ins are loaded when required on the client. Immediately after DB2 loads the plug-in library, it will resolve the location of this function and call it. The specific task of this function is as follows:

- Cast the functions pointer to a pointer to an appropriate functions structure
- Fill in the pointers to the other functions in the library
- Fill in the version number of the function pointer structure being returned

© Copyright IBM Corp. 1993-2004

DB2 can potentially call the plug-in initialization function more than once. This situation can occur when an application dynamically loads the DB2 client library, unloads it, and reloads it again, then performs authentication functions from a plug-in both before and after reloading. In this situation, the plug-in library might not be unloaded and then re-loaded; however, this behavior varies depending on the operating system.

Another example of DB2 issuing multiple calls to a plug-in initialization function occurs during the execution of stored procedures or federated system calls, where the database server can itself act as a client. If the client and server plug-ins on the database server are in the same file, DB2 could call the plug-in initialization function twice.

If the plug-in detects that `db2secGroupPluginInit` is called more than once, it should handle this event as if it was directed to terminate and reinitialize the plug-in library. As such, the plug-in initialization function should do the entire cleanup that a call to `db2secPluginTerm` would do before returning the set of function pointers again.

On a DB2 server running on a UNIX®-based operating system, DB2 can potentially load and initialize plug-in libraries more than once in different processes.

Related concepts:

- "Security plug-ins" on page 1021
- "Security plug-in library locations" on page 1024

Related reference:

- "Restrictions on security plug-in libraries" on page 1038
- "Return codes for security plug-ins" on page 1040
- "Calling sequences for the security plug-in APIs" on page 1043
- "db2secGroupPluginInit - Initialize group plug-in function" on page 1050
- "db2secPluginTerm - Clean up group plug-in resources function" on page 1051
- "db2secClientAuthPluginInit - Initialize client authentication plug-in" on page 1064
- "db2secServerAuthPluginInit - Initialize server authentication plug-in function" on page 1075

Restrictions on security plug-in libraries

Following are restrictions for developing plug-in libraries.

C-linkage
Plug-in libraries must be linked with C-linkage. Header files providing the prototypes, data structures needed to implement the plug-ins, and error code definitions are provided for C/C++ only. Functions that DB2 will resolve at load time must be declared with extern ″C″ if the plug-in library is compiled as C++.

.NET common language runtime is not supported
The .NET common language runtime (CLR) is not supported for compiling and linking source code for plug-in libraries.

Signal handlers
The plug-in libraries must not install signal handlers or change the signal mask, because this will interfere with DB2's signal handlers. Interfering

with the DB2 signal handlers could seriously interfere with DB2's ability to report and recover from errors, including traps in the plug-in code itself. Plug-in libraries should also never throw C++ exceptions, as this can also interfere with DB2's error handling.

Thread-safe
Plug-in libraries must be thread-safe and re-entrant. The plug-in initialization function is the only API that is not required to be re-entrant. The plug-in initialization function could potentially be called multiple times from different processes; in which case, the plug-in will cleanup all used resources and reinitialize itself.

Exit handlers and overriding standard C library and operating system calls
Plug-in libraries should not override standard C library or operating system calls. Plug-in libraries should also not install exit handlers or `pthread_atfork` handlers. The use of exit handlers is not recommended because they may be unloaded before the program exits.

Library dependencies
On Linux or UNIX the processes that load the plug-in libraries can be `setuid` or `setgid`, which means that they will not be able to rely on the `$LD_LIBRARY_PATH`, `$SHLIB_PATH`, or `$LIBPATH` environment variables to find dependent libraries. Therefore, plug-in libraries should not depend on other libraries, unless any dependant libraries are accessible through other methods, such as the following:

- By being in `/lib` or `/usr/lib`
- By having the directories they reside in being specified OS-wide (such as in the `ld.so.conf` file on Linux)
- By being specified in the RPATH in the plug-in library itself

This restriction is not applicable to Windows operating systems.

Symbol collisions
When possible, plug-in libraries should be compiled and linked with any available options that reduce the likelihood of symbol collisions, such as those that reduce unbound external symbolic references. For example, use of the "-Bsymbolic" linker option on HP, Sun Solaris, and Linux can help prevent problems related to symbol collisions. However, for a plug-in written on AIX platform, do not use `"-brtl"` linker option explicitly or implicitly.

32-bit and 64-bit applications
32-bit applications must use 32-bit plug-ins. 64-bit applications must use 64-bit plug-ins. Please refer to the topic 32-bit and 64-bit considerations for security plug-ins for more details.

Text strings
Input text strings are not guaranteed to be null-terminated, and output strings are not required to be null-terminated. Instead, integer lengths are given for all input strings, and pointers to integers are given for lengths to be returned.

Passing authid parameters
An authid parameter that DB2 passes into a plug-in (an input authid parameter) will contain an upper-case authid, with padded blanks removed. An authid parameter that a plug-in returns to DB2 (an output authid parameter) does not require any special treatment, but DB2 will take the authid and fold it to upper-case, and pad it with blanks according to the internal DB2 standard.

Size limits for parameters

The plug-in APIs use the following as length limits for parameters:

```
#define DB2SEC_MAX_AUTHID_LENGTH 255
#define DB2SEC_MAX_USERID_LENGTH 255
#define DB2SEC_MAX_USERNAMESPACE_LENGTH 255
#define DB2SEC_MAX_PASSWORD_LENGTH 255
#define DB2SEC_MAX_DBNAME_LENGTH 128
```

A particular plug-in implementation may require or enforce smaller maximum lengths for the authorization IDs, user IDs, and passwords. In particular, the operating system authentication plug-ins supplied with DB2 UDB are restricted to the maximum user, group and namespace length limits enforced by the operating system for cases where the operating system limits are lower than those stated above.

Related concepts:

- "Security plug-ins" on page 1021
- "Security plug-in library locations" on page 1024

Return codes for security plug-ins

All security plug-in APIs must return an integer value to indicate the success or failure of the execution of the API. A return code value of 0 indicates that the API ran successfully. All negative return codes, with the exception of -3, -4, and -5, indicate that the API encountered an error.

All negative return codes returned from the security-plug-in APIs are mapped to SQLCODE -1365, SQLCODE -1366, or SQLCODE -30082, with the exception of return codes with the -3, -4, or -5. The values -3, -4, and -5 are used to indicate whether or not an AUTHID represents a valid user or group.

All the security plug-in API return codes are defined in db2secPlugin.h, which can be found in DB2's include directory: SQLLIB/include.

Details regarding all of the security plug-in return codes are presented in the following table:

Table 83. Security plug-in return codes

Return code	Define value	Meaning	Applicable APIs
0	DB2SEC_PLUGIN_OK	The plug-in API executed successfully.	All
-1	DB2SEC_PLUGIN _UNKNOWNERROR	The plug-in API encountered an unexpected error.	All
-2	DB2SEC_PLUGIN_BADUSER	The user ID passed in as input is not defined.	db2secGenerateInitialCred db2secValidatePassword db2secRemapUserid db2secGetGroupsForUser
-3	DB2SEC_PLUGIN _INVALIDUSERORGROUP	No such user or group.	db2secDoesAuthIDExist db2secDoesGroupExist
-4	DB2SEC_PLUGIN _USERSTATUSNOTKNOWN	Unknown user status. This is not treated as an error by DB2; it is used by a GRANT statement to determine if an authid represents a user or an operating system group.	db2secDoesAuthIDExist

Table 83. Security plug-in return codes (continued)

Return code	Define value	Meaning	Applicable APIs
-5	DB2SEC_PLUGIN _GROUPSTATUSNOTKNOWN	Unknown group status. This is not treated as an error by DB2; it is used by a GRANT statement to determine if an authid represents a user or an operating system group.	db2secDoesGroupExist
-6	DB2SEC_PLUGIN_UID_EXPIRED	User ID expired.	db2secValidatePassword db2GetGroupsForUser db2secGenerateInitialCred
-7	DB2SEC_PLUGIN_PWD_EXPIRED	Password expired.	db2secValidatePassword db2GetGroupsForUser db2secGenerateInitialCred
-8	DB2SEC_PLUGIN_USER_REVOKED	User revoked.	db2secValidatePassword db2GetGroupsForUser
-9	DB2SEC_PLUGIN _USER_SUSPENDED	User suspended.	db2secValidatePassword db2GetGroupsForUser
-10	DB2SEC_PLUGIN_BADPWD	Bad password.	db2secValidatePassword db2secRemapUserid db2secGenerateInitialCred
-11	DB2SEC_PLUGIN _BAD_NEWPASSWORD	Bad new password.	db2secValidatePassword db2secRemapUserid
-12	DB2SEC_PLUGIN _CHANGEPASSWORD _NOTSUPPORTED	Change password not supported.	db2secValidatePassword db2secRemapUserid db2secGenerateInitialCred
-13	DB2SEC_PLUGIN_NOMEM	Plug-in attempt to allocate memory failed due to insufficient memory.	All
-14	DB2SEC_PLUGIN_DISKERROR	Plug-in encountered a disk error.	All
-15	DB2SEC_PLUGIN_NOPERM	Plug-in attempt to access a file failed because of wrong permissions on the file.	All
-16	DB2SEC_PLUGIN_NETWORKERROR	Plug-in encountered a network error.	All
-17	DB2SEC_PLUGIN _CANTLOADLIBRARY	Plug-in is unable to load a required library.	db2secGroupPluginInit db2secClientAuthPluginInit db2secServerAuthPluginInit
-18	DB2SEC_PLUGIN_CANT _OPEN_FILE	Plug-in is unable to open and read a file for a reason other than missing file or inadequate file permissions.	All
-19	DB2SEC_PLUGIN_FILENOTFOUND	Plug-in is unable to open and read a file, because the file is missing from the file system.	All
-20	DB2SEC_PLUGIN _CONNECTION_DISALLOWED	The plug-in is refusing the connection because of the restriction on which database is allowed to connect, or the TCP/IP address cannot connect to a specific database.	All server-side plug-in APIs.
-21	DB2SEC_PLUGIN_NO_CRED	GSS API plug-in only: initial client credential is missing.	db2secGetDefaultLoginContext db2secServerAuthPluginInit
-22	DB2SEC_PLUGIN_CRED_EXPIRED	GSS API plug-in only: client credential has expired.	db2secGetDefaultLoginContext db2secServerAuthPluginInit
-23	DB2SEC_PLUGIN _BAD_PRINCIPAL_NAME	GSS API plug-in only: the principal name is invalid.	db2secProcessServerPrincipalName
-24	DB2SEC_PLUGIN _NO_CON_DETAILS	This return code is returned by the `db2secGetConDetails` callback (for example, from DB2 to the plug-in) to indicate that DB2 is unable to determine the client's TCP/IP address.	db2secGetConDetails

Table 83. Security plug-in return codes (continued)

Return code	Define value	Meaning	Applicable APIs
-25	DB2SEC_PLUGIN _BAD_INPUT_PARAMETERS	Some parameters are not valid or are missing when plug-in API is called.	All
-26	DB2SEC_PLUGIN _INCOMPATIBLE_VER	The version of the APIs reported by the plug-in is not compatible with DB2.	db2secGroupPluginInit db2secClientAuthPluginInit db2secServerAuthPluginInit
-27	DB2SEC_PLUGIN_PROCESS_LIMIT	Insufficient resources are available for the plug-in to create a new process.	All
-28	DB2SEC_PLUGIN_NO_LICENSES	The plug-in encountered a user license problem. A possibility exists that the underlying mechanism license has reached the limit.	All

Related concepts:

- "Security plug-ins" on page 1021
- "Security plug-in problem determination" on page 1029

Error messages for security plug-ins

When an error occurs in a security plug-in API, the API can return an ASCII text string in the `errormsg` field to provide a more specific description of the problem than the return code. For instance, the `errormsg` string can contain `"File /home/db2inst1/mypasswd.txt does not exist."` DB2 will write this entire string into the DB2 administration notification log, and will also include a truncated version as a token in some SQL messages. Because tokens in SQL messages can only be of limited length, these messages should be kept short, and important variable portions of these messages should appear at the front of the string. To aid in debugging, consider adding the name of the security plug-in to the error message.

For non-urgent errors, such as password expired errors, the `errormsg` string will only be dumped when the DIAGLEVEL database manager configuration parameter is set at 4.

The memory for these error messages must be allocated by the security plug-in. Therefore, the plug-ins must also provide an API to free this memory: `db2secFreeErrormsg`.

The `errormsg` field will only be checked by DB2 if an API returns a non-zero value. Therefore, the plug-in should not allocate memory for this returned error message if there is no error.

At initialization time a message logging function pointer, `logMessage_fn`, is passed to the group, client, and server plug-ins. The plug-ins can use the function to log any debugging information to db2diag.log. For example:

```
// Log an message indicate init successful
(*(logMessage_fn))(DB2SEC_LOG_CRITICAL,
                   "db2secGroupPluginInit successful",
                    strlen("db2secGroupPluginInit successful"));
```

For more details about each parameter for the db2secLogMessage function, please refer to the initialization API for each of the plug-in types.

Related concepts:

- "Security plug-ins" on page 1021
- "Security plug-in problem determination" on page 1029
- "Security plug-in APIs" on page 1047
- "Security plug-in support for two-part user IDs" on page 1026

Related reference:
- "Return codes for security plug-ins" on page 1040

Calling sequences for the security plug-in APIs

There are five main scenarios in which DB2 will call security plug-in APIs:
- On a client for a database connection
- On a client, server, or gateway for local authorization
- On a server for a database connection
- On a server for a grant statement
- On a server to get a list of groups that an authid belongs to

Note: The DB2 server treats database actions requiring local authorizations, such as `db2start`, `db2stop`, and `db2trc`, like client applications.

For each of these operations, the sequence with which DB2 calls security plug-in APIs is appropriately different. Following are the sequences of APIs called by DB2 for each of these scenarios.

On a client for a database connection

When the user-configured authentication type is CLIENT, the DB2 client application will call the following security plug-in APIs:
- `db2secGetDefaultLoginContext();`
- `db2secValidatePassword();`
- `db2secFreetoken();`

For an implicit authentication, that is, when you connect without specifying a particular user ID or password, the `db2secValidatePassword` API is called if you are using a user ID/password plug-in. This API permits plug-in developers to prohibit implicit authentication if necessary.

On an implicit authentication, if the *authentication* database manager configuration parameter is set to anything other than CLIENT (implying authentication at the server), the application will call the following security plug-in APIs for the user ID/password authentication mechanism:
- `db2secGetDefaultLoginContext();`
- `db2secFreeToken();`

On an implicit authentication, if *authentication* is set to anything other than CLIENT (implying authentication at the server), the application will call the following security plug-in APIs for GSS-API plug-ins. (The call to `gss_init_sec_context()` will use GSS_C_NO_CREDENTIAL as the input credential.)
- `db2secGetDefaultLoginContext();`
- `db2secProcessServerPrincipalName();`
- `gss_init_sec_context();`
- `gss_release_buffer();`

- `gss_release_name();`
- `gss_delete_sec_context();`
- `db2secFreeToken();`

The API *gss_init_sec_context()* may be called twice if a mutual authentication token is returned from the server.

On an explicit authentication, if *authentication* is set to CLIENT the DB2 client application will call the following security plug-in APIs:

- `db2secRemapUserid();`
- `db2secValidatePassword();`
- `db2secFreeToken();`

On an explicit authentication, if *authentication* is set to anything other than CLIENT, the application will call the following security plug-in APIs for the user ID/password authentication mechanism:

- `db2secRemapUserid();`

If the negotiated authentication type is GSS-API or Kerberos, the client application will call the following security plug-in APIs for GSS-API plug-ins in the following sequence. These APIs are used for both implicit and explicit authentication (a connection to a database in which both the user ID and password are specified) unless otherwise stated.

- `db2secProcessServerPrincipalName();`
- `db2secGenerateInitialCred();` (For explicit authentication only)
- `gss_init_sec_context();`
- `gss_release_buffer ();`
- `gss_release_name();`
- `gss_release_cred();`
- `db2secFreeInitInfo();`
- `gss_delete_sec_context();`
- `db2secFreeToken();`

The API gss_init_sec_context() may be called twice if a mutual authentication token is returned from the server.

On a client, server, or gateway for local authorization

For a local authorization, the DB2 command being used will call the following security plug-in APIs:

- `db2secGetDefaultLoginContext();`
- `db2secGetGroupsForUser();`
- `db2secFreeToken();`
- `db2secFreeGroupList();`

These APIs will be called for both user ID/password and GSS-API authentication mechanisms.

On a server for a database connection

For a database connection on the database server, the DB2 agent process or thread will call the following security plug-in APIs for the user ID/password authentication mechanism:

- `db2secValidatePassword();` Only if the *authentication* database configuration parameter is not CLIENT
- `db2secGetAuthIDs();`
- `db2secGetGroupsForUser();`
- `db2secFreeToken();`
- `db2secFreeGroupList();`

For a CONNECT to a database, the DB2 agent process or thread will call the following security plug-in APIs for the GSS-API authentication mechanism:

- `gss_accept_sec_context();`
- `gss_release_buffer();`
- `db2secGetAuthIDs();`
- `db2secGetGroupsForUser();`
- `gss_delete_sec_context();`
- `db2secFreeToken();`
- `db2secFreeGroupList();`

On a server for a GRANT statement

For a GRANT statement that does not specify the USER or GROUP keyword, (for example, "`GRANT CONNECT ON DATABASE TO user1`"), DB2 must be able to determine if `user1` is a user, a group, or both. Therefore, DB2 will call the following security plug-in APIs:

- `db2secDoesGroupExist();`
- `db2secDoesAuthIDExist();`

On a server to get a list of groups to which an authid belongs

From your database server, when you need to get a list of groups to which an authid belongs, DB2 will call the following security plug-in API with only the authid as input:

- `db2secGetGroupsForUser();`

There will be no token from other security plug-ins.

Related concepts:

- "Security plug-ins" on page 1021
- "Security plug-in APIs" on page 1047

Chapter 30. Security plug-in APIs

Security plug-in APIs

To enable you to customize DB2®'s authorization behavior, DB2 provides APIs that you can use to modify existing plug-ins or build new security plug-ins.

When you develop a security plug-in, you need to implement the standard authentication functions that DB2 will invoke. For the three available types of plug-ins, the functionality you need to implement is as follows:

Group retrieval
Retrieves group membership information for a given user and determines if a given string represents a valid group name.

User ID/password authentication
Authentication that identifies the default security context (client only), validates and optionally changes a password, determines if a given string represents a valid user (server only), modifies the user ID or password provided on the client before it is sent to the server (client only), returns the DB2 authorization ID associated with a given user.

GSS-API authentication
Authentication that implements the required GSS-API functions, identifies the default security context (client side only), generates initial credentials based on user ID and password, and optionally changes password (client side only), creates and accepts security tickets, and returns the DB2 authorization ID associated with a given GSS-API security context.

© Copyright IBM Corp. 1993-2004

The following are definitions for terminology used in the descriptions of the plug-in APIs.

Plug-in
A dynamically loadable library that DB2 will load to access user-written authentication functions.

Implicit authentication
A connection to a database without specifying a user ID or a password.

Explicit authentication
A connection to a database in which both the user ID and password are specified.

Authid
An internal ID representing an individual or group to which authorities and privileges within the database are granted. Internally, a DB2 authid is folded to upper-case and is a minimum of 8 characters (blank padded to 8 characters). Currently, DB2 requires authids, user IDs, passwords, group names, namespaces, and domain names that can be represented in 7-bit ASCII. The maximum length of an authid is 30 characters.

Local authorization
Authorization that is local to the server or client that implements it, that checks if a user is authorized to perform an action (other than connecting to the database), such as starting and stopping the database manager, turning DB2 trace on and off, or updating the database manager configuration.

Namespace
A collection or grouping of users within which individual user identifiers must be unique. Common examples include Windows® domains and Kerberos Realms. For example, within the Windows domain ″usa.company.com″ all user names must be unique. For example, ″user1@usa.company.com″. The same user ID in another domain, as in the case of ″user1@canada.company.com″, however refers to a different person. A fully qualified user identifier includes a user ID and namespace pair; for example, ″user@domain.name″ or ″domain\user″.

Related concepts:

• “Security plug-ins” on page 1021

Group plug-in APIs

APIs for group retrieval plug-ins

For the group retrieval plug-in library, you will need to implement the following APIs:

```
SQL_API_RC SQL_API_FN db2secGroupPluginInit(
 db2int32 version,
 void *group_fns,
 db2secLogMessage *logMessage_fn,
 char **errormsg,
 db2int32  *errormsglen);
```

Note: The above function takes as input a pointer to a function, `*logMessage_fn`, with the following prototype:

```
      SQL_API_RC (SQL_API_FN db2secLogMessage) (
        db2int32 level,
       void *data,
       db2int32 length);
SQL_API_RC SQL_API_FN db2secPluginTerm(char **errormsg,
      db2int32  *errormsglen);
SQL_API_RC SQL_API_FN db2secGetGroupsForUser(
 const char *authid,
 db2int32  authidlen,
 const char *userid,
 db2int32  useridlen,
 const char *usernamespace,
 db2int32  usernamespacelen,
 db2int32  usernamespacetype,
 const char *dbname,
 db2int32  dbnamelen,
 const void *token,
 db2int32 tokentype,
 db2int32 location,
 const char *authpluginname,
 db2int32  authpluginname,
 char **grouplist,
 db2int32  *numgroups,
 char **errormsg,
 db2int32  *errormsglen);
SQL_API_RC SQL_API_FN db2secDoesGroupExist(
  const char *groupname,
  db2int32  groupnamelen
  char **errormsg,
  db2int32  *errormsglen);
SQL_API_RC SQL_API_FN db2secFreeGroupListMemory(
  char *ptr,
  char **errormsg,
  db2int32  *errormsglen);
SQL_API_RC SQL_API_FN db2secFreeErrormsg(char *msgtobefree);
```

The only API that must be resolvable externally is `db2secGroupPluginInit()`. This function will take a `void *` parameter, which should be cast to the type:

```
typedef struct db2secGroupFunctions_1
{
  db2int32 version;
  db2int32 plugintype;
  SQL_API_RC (SQL_API_FN * db2secGetGroupsForUser) (
    const char *authid,
    db2int32  authidlen,
    const char *userid,
    db2int32  useridlen,
    const char *usernamespace,
    db2int32  usernamespacelen,
    db2int32  usernamespacetype,
    const char *dbname,
    db2int32  dbnamelen,
    const void *token,
    db2int32 tokentype,
    db2int32 location,
    const char *authpluginname,
    db2int32  authpluginnamelen,
    void **grouplist,
    db2int32  *numgroups,
    char **errormsg,
    db2int32  *errormsglen);

 SQL_API_RC (SQL_API_FN * db2secDoesGroupExist)(
    const char *groupname,
```

```
    db2int32  groupnamelen,
    char **errormsg,
    db2int32  *errormsglen);

  SQL_API_RC (SQL_API_FN * db2secFreeGroupListMemory)(
    void *ptr,
    char **errormsg,
    db2int32  *errormsglen);

  SQL_API_RC (SQL_API_FN * db2secFreeErrormsg)(
    char *msgtobefree);

 SQL_API_RC (SQL_API_FN * db2secPluginTerm)(
    char **errormsg,
    db2int32  *errormsglen);

} db2secGroupFunctions_1;
```

`db2secGroupPluginInit()` will assign the addresses for the rest of the externally available functions.

Note: The _1 indicates that this is the structure corresponding to version 1 of the API. Subsequent interface versions will have the extension _2, _3, and so on.

Related concepts:

- "Security plug-ins" on page 1021
- "Security plug-in APIs" on page 1047

Related tasks:

- "Deploying a group retrieval plug-in" on page 1030

Related reference:

- "db2secGroupPluginInit - Initialize group plug-in function" on page 1050
- "db2secPluginTerm - Clean up group plug-in resources function" on page 1051
- "db2secGetGroupsForUser - Get list of groups for user function" on page 1052
- "db2secDoesGroupExist - Check if group exists function" on page 1055
- "db2secFreeGroupListMemory - Free group list memory function" on page 1056
- "db2secFreeErrormsg - Free error message memory function" on page 1057

db2secGroupPluginInit - Initialize group plug-in function

The initialization function for the library that DB2 will call immediately after loading the plug-in library. The `functions` pointer should be cast to the appropriate `group_functions` structure for the interface version.

C API syntax:

```
SQL_API_RC SQL_API_FN db2secGroupPluginInit(
  db2int32 version,
  void *group_fns,
  db2secLogMessage *logMessage_fn,
  char **errormsg,
  db2int32  *errormsglen);
```

Input:

db2int32 version
The highest version number of the API that DB2 will currently support.

*db2secLogMessage *logMessage_fn*
A pointer to a function provided by DB2. The plug-in can call this function to log additional error strings to db2diag.log for either debugging or informational purposes. The first parameter should use the define in db2secPlugin.h, and the last two parameters are the message string and its length. The defines to be used in the first parameter are:

```
#define DB2SEC_LOG_NONE       0   -  No logging
#define DB2SEC_LOG_CRITICAL   1   -  Severe Error encountered
#define DB2SEC_LOG_ERROR      2   -  Error encountered
#define DB2SEC_LOG_WARNING    3   -  Warning
#define DB2SEC_LOG_INFO       4   -  Informational
```

If you use the DB2SEC_LOG_INFO define, the message text is only written to the db2diag.log if the *diaglevel* database manager configuration parameter is set to 4.

Output:

*void *group_fns*
A pointer to memory provided by DB2 for a db2secGroupFunction_1 structure. In future versions of DB2, there may be different versions of the APIs, so this should be cast to a pointer to the db2secGroupFunction_1 structure corresponding to the version of the API that the plug-in uses.

*char **errormsg*
A pointer to the address of an ASCII string allocated by the plug-in that can be returned in this parameter if the API is not successful.

*db2int32 *errormsglen*
A pointer to an integer that indicates the length of the error message string in *char **errormsg*.

Related concepts:

- "Security plug-ins" on page 1021
- "Security plug-in APIs" on page 1047

Related reference:

- "APIs for group retrieval plug-ins" on page 1048

db2secPluginTerm - Clean up group plug-in resources function

This function is called by DB2 just before DB2 unloads the plug-in. The function should do a proper cleanup of any resources the plug-in library holds, for instance, free any memory allocated by the plug-in, close files that are still open, and close network connections. The plug-in is responsible for keeping track of these resources so that they can later be freed.

C API syntax:

```
SQL_API_RC SQL_API_FN db2secPluginTerm(char **errormsg,
           db2int32  *errormsglen);
```

Output:

*char **errormsg*
A pointer to the address of an ASCII string allocated by the plug-in that can be returned in this parameter if the API is not successful.

*db2int32 *errormsglen*
A pointer to an integer that indicates the length of the error message string in *char **errormsg*.

Related concepts:

- "Security plug-ins" on page 1021
- "Security plug-in APIs" on page 1047

Related reference:

- "diaglevel - Diagnostic error capture level configuration parameter" in the *Administration Guide: Performance*
- "APIs for group retrieval plug-ins" on page 1048

db2secGetGroupsForUser - Get list of groups for user function

This function will be called by DB2 to get the list of groups to which a user belongs.

C API syntax:

```
SQL_API_RC SQL_API_FN db2secGetGroupsForUser(
  const char *authid,
  db2int32  authidlen,
  const char *userid,
  db2int32  useridlen,
  const char *usernamespace,
  db2int32  usernamespacelen,
  db2int32  usernamespacetype,
  const char *dbname,
  db2inst32 dbnamelen,
  const void *token,
  db2int32 tokentype,
  db2int32 location,
  const char *authpluginname,
  db2int32  authpluginnamelen,
  void **grouplist,
  db2int32  *numgroups,
  char **errormsg,
  db2int32  *errormsglen);
```

Input:

*const char *authid*
The only input field that is provided by DB2. This field value is an SQL authid, which means that DB2 folds it to an uppercase character string with no trailing blanks. The plug-in must be able to return a list of groups to which the authid belongs without depending on the other input parameters. It is permissible to return a shortened or empty list if group membership cannot be determined.

If a user does not exist, the function should return `DB2SEC_PLUGIN_BADUSER`. DB2 does not treat the case of a user not existing as an error, because it is permissible for an authid to not have any groups associated with it. For example, the `db2secGetAuthids` function can return an authid that does not exist on the operating system. The authid is not associated with any groups; however, it can still be directly assigned privileges.

If the plug-in cannot return a complete list of groups from only the authid, some restrictions apply to the SQL functions related to group support.

Refer to the note in this topic, titled ″Problems that may occur if an incomplete group list is returned″ for more information.

db2int32 authidlen
Length of the authid.

*const char *userid*
The user ID corresponding to the authid. When this API is called on the server in a non-connect scenario, this will not be filled.

db2int32 useridlen
Length of the user ID.

*const void *token*
A pointer to data provided by the authentication plug-in. It is not seen by DB2. It provides the ability to the plug-in writer for coordinating user and group information, if desired. This may not be given in all cases, in which case it will be NULL. If the authentication plug-in used is GSS-API based, the token will be set to the GSS-API context handle (gss_ctx_id_t).

db2int32 tokentype
Indicates the type of data provided by the authentication plug-in. If the authentication plug-in used is GSS-API based, the token will be set to the GSS-API context handle (gss_ctx_id_t). If the authentication plug-in used is user ID/password based, it will be a generic type. See the following defines in `db2secPlugin.h`:

- #define DB2SEC_GENERIC 0 -- Indicates that the token is from a user ID/password based plug-in.
- #define DB2SEC_GSSAPI_CTX_HANDLE 1 -- Indicates that the token is from a GSS-API (including Kerberos) based plug-in.

db2int32 location
Indicates whether DB2 is to call the plug-in on the client side or server side. See the following defines in `db2secPlugin.h`:

- #define DB2SEC_SERVER_SIDE 0 -- The group plug-in is being called on the database server.
- #define DB2SEC_CLIENT_SIDE 1 -- The group plug-in is being called on a client.

*const char *usernamespace*
The namespace from which the user ID was obtained. When the user ID is not available, this will not be filled.

db2int32 usernamespacelen
Length of the namespace field.

db2int32 usernamespacetype
The type of namespace. Possible values are:
DB2SEC_NAMESPACE_SAM_COMPATIBLE (corresponding to a username style like torolab\myname″), or DB2SEC_NAMESPACE_USER_PRINCIPAL (corresponding to a username style like myname@torolab.ibm.com).
Currently DB2 only supports DB2SEC_NAMESPACE_SAM_COMPATIBLE. When the user ID is not available, this will be filled with DB2SEC_USER_NAMESPACE_UNDEFINED. All the defines are located in `db2secPlugin.h`.

*const char *dbname*
The name of the database being connected to.

db2int32 dbnamelen
Length of the database name specified by *dbname*.

*const char *authpluginname*
The name of the authentication plug-in that provided the data in the token. The plug-in may use this information in determining the correct group memberships. This may not be given if the authid is not authenticated (for instance, if the authid does not match the current connected user).

db2int32 authpluginnamelen
Length of the authpluginname.

Output:

*void **grouplist*
The list of groups must be returned as a pointer to a section of memory allocated by the plug-in containing concatenated `varchars` (a `varchar` is a character array in which the first byte indicates the number of bytes following it). The length is an unsigned char and that limits the maximum length of a groupname to 255 characters. In other words, because an unsigned char (1 byte) indicates the length of the group name, the maximum length is 255. For example:

```
"\006GROUP1\007MYGROUP\008MYGROUP3"
```

Each group name should be a valid DB2 authid. The memory for this array must be allocated by the plug-in. The plug-in must therefore provide a function, such as the `db2secFreeGroupListMemory()` plug-in function that DB2 will call to free the memory.

*db2int32 *numgroups*
The number of groups contained in the grouplist.

*char **errormsg*
A pointer to the address of an ASCII string allocated by the plug-in that can be returned in this parameter if the API is not successful.

*db2int32 *errormsglen*
A pointer to an integer that indicates the length of the error message string in *char **errormsg*.

Problems that may occur if an incomplete group list is returned from the API:

The following problems can occur if an incomplete group list is returned from this API to DB2 UDB:

- Embedded SQL application with DYNAMICRULES BIND (or DEFINEDBIND or INVOKEDBIND if the packages are running as a standalone application). DB2 checks for SYSADM membership and the application will fail if it is dependent on the implicit DBADM authority granted by being a member of the SYSADM group.
- Alternate authorization is provided in CREATE SCHEMA statement. Group lookup will be performed against the AUTHORIZATION NAME parameter if there are nested CREATE statements in the CREATE SCHEMA statement.
- Embedded SQL applications with DYNAMICRULES DEFINERUN/DEFINEBIND and the packages are running in a routine context. DB2 checks for SYSADM membership of the routine definer and the application will fail if it is dependent on the implicit DBADM authority granted by being a member of the SYSADM group.

- Processing a jar file in a partitioned database environment. In a partitioned database environment, the jar processing request is sent from the coordinator partition with the session authid. The catalog partition received the requests and process the jar files based on the privilege of the session authid (the user executing the jar processing requests).
  - Install jar file. The session authid needs to have one of the following rights: SYSADM, DBADM, or CREATEIN (implicit or explicit on the jar schema). The operation will fail if the above rights are granted to a group containing the session authid, but not explicitly to the session authid or if only SYSADM is held, because SYSADM membership is determined by membership in the group defined by a database configuration parameter.
  - Remove jar file. The session authid needs to have one of the following rights: SYSADM, DBADM, or DROPIN (implicit or explicit on the jar schema), or is the definer of the jar file. The operation will fail if the above rights are granted to a group containing the session authid, but not explicitly to the session authid, and if the session authid is not the definer of the jar file or if only SYSADM is held, because SYSADM membership is determined by membership in the group defined by a database configuration parameter.
  - Replace jar file. This is same as removing the jar file, followed by installing the jar file. Both of the above apply.
- Regenerate views. This is triggered by the ALTER TABLE, ALTER COLUMN, SET DATA TYPE VARCHAR/VARGRAPHIC statements, or during migration. DB2 checks for SYSADM membership of the view definer. The application will fail if it is dependent on the implicit DBADM authority granted by being a member of the SYSADM group.
- When SET SESSION_USER statement is issued. Subsequent DB2 operations are run under the context of the authid specified by this statement. These operations will fail if the privileges required are owned by one of the SESSION_USER's groups, but are not explicitly granted to the SESSION_USER authid.

Related concepts:

- "Security plug-ins" on page 1021
- "Security plug-in APIs" on page 1047

Related reference:

- "APIs for group retrieval plug-ins" on page 1048

db2secDoesGroupExist - Check if group exists function

This function will be used to determine if an authid represents a group. If the groupname exists, The function should return DB2SEC_PLUGIN_OK, to indicate success. It should return DB2SEC_PLUGIN_INVALIDUSERORGROUP if the group name is not valid. It is also permissible for the API to return DB2SEC_PLUGIN_GROUPSTATUSNOTKNOWN if it is impossible to determine if the input is a valid group. If an invalid group or group not known value is returned, DB2 might not be able to determine whether the authid is a group or user when issuing the GRANT statement without the keywords USER and GROUP, which would result in the error SQLCODE -569, SQLSTATE 56092 being returned to the user.

C API syntax:

```
SQL_API_RC SQL_API_FN db2secDoesGroupExist(
 const char *groupname,
 db2int32  groupnamelen,
 char **errormsg,
 db2int32  *errormsglen
);
```

Input:

*const char *groupname*
An authid, upper-cased, with no trailing blanks.

db2int32 groupnamelen
Length of the groupname.

Output:

*char **errormsg*
A pointer to the address of an ASCII string allocated by the plug-in that can be returned in this parameter if the API is not successful.

*db2int32 *errormsglen*
A pointer to an integer that indicates the length of the error message string in *char **errormsg*.

Related concepts:

- "Security plug-ins" on page 1021
- "Security plug-in APIs" on page 1047

Related reference:

- "APIs for group retrieval plug-ins" on page 1048

db2secFreeGroupListMemory - Free group list memory function

This function tells the plug-in library that the memory pointed to by `ptr` is no longer needed by DB2. The plug-in needs to free this memory.

C API syntax:

```
SQL_API_RC SQL_API_FN db2secFreeGroupListMemory(
 void *ptr
 char **errormsg,
 db2int32  *errormsglen);
```

Input:

*void *ptr*
Pointer to the memory to be freed.

Output:

*char **errormsg*
A pointer to the address of an ASCII string allocated by the plug-in that can be returned in this parameter if the API is not successful.

*db2int32 *errormsglen*
A pointer to an integer that indicates the length of the error message string in *char **errormsg*.

Related concepts:

- "Security plug-ins" on page 1021
- "Security plug-in APIs" on page 1047

Related reference:

- "APIs for group retrieval plug-ins" on page 1048

db2secFreeErrormsg - Free error message memory function

This function will be called by DB2 to free the memory used to hold an error message from a previous call to a plug-in API. This is the only API that does not also return an error message. If this function returns an error, DB2 will log it and continue.

C API syntax:

```
SQL_API_RC SQL_API_FN db2secFreeErrormsg(char *msgtobefree);
```

Input:

*char *msgtobefree*
A pointer to the error message allocated from a previous API call.

Related concepts:

- "Security plug-ins" on page 1021
- "Security plug-in APIs" on page 1047

Related reference:

- "APIs for group retrieval plug-ins" on page 1048

User authentication plug-in APIs

APIs for user ID/password authentication plug-ins

For the user ID/password plug-in library, you will need to implement the following client-side APIs:

```
SQL_API_RC SQL_API_FN db2secClientAuthPluginInit(
  db2int32 version,
  void *client_fns,
  db2secLogMessage *logMessage_fn,
  char **errormsg,
  db2int32  *errormsglen);
```

Note: The above function takes as input a pointer to a function, `*logMessage_fn`, with the following prototype:

```
SQL_API_RC (SQL_API_FN db2secLogMessage) (
  db2int32 level,
  void *data,
  db2int32 length);
```

```
SQL_API_RC SQL_API_FN db2secClientAuthPluginTerm(
  char **errormsg,
  db2int32  *errormsglen);
```

```
/* Only used for gssapi: */
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);

/* Optional */
SQL_API_RC SQL_API_FN db2secRemapUserid(
  char userid[DB2SEC_MAX_USERID_LENGTH],
  db2int32  *useridlen,
  db2int32  useridtype,
  char usernamespace[DB2SEC_MAX_USERNAMESPACE_LENGTH],
  db2int32  *usernamespacelen,
  db2int32  *usernamespacetype,
  char password[DB2SEC_MAX_PASSWORD_LENGTH],
  db2int32  *passwordlen,
  char newpassword[DB2SEC_MAX_PASSWORD_LENGTH],
  db2int32  *newpasswordlen,
  const char *dbname,
  db2int32  dbnamelen,
  char **errormsg,
  db2int32  *errormsglen);
SQL_API_RC SQL_API_FN db2secGetDefaultLoginContext(
  char authid[DB2SEC_MAX_AUTHID_LENGTH],
  db2int32  *authidlen,
  char userid[DB2SEC_MAX_USERID_LENGTH],
  db2int32  *useridlen,
  db2int32 useridtype,
  char usernamespace[DB2SEC_MAX_USERNAMESPACE_LENGTH],
  db2int32  *usernamespacelen,
  db2int32  *usernamespacetype,
  const char *dbname,
  db2int32  dbnamelen,
  void **token,
  char **errormsg,
  db2int32  *errormsglen);
SQL_API_RC SQL_API_FN db2secValidatePassword(
  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,
  db2Uint32   connection_details,
  void **token,
  char **errormsg,
  db2int32  *errormsglen);
```

```
/* This is only for GSS-API */
SQL_API_RC SQL_API_FN db2secProcessServerPrincipalName(
  const void *name,
  db2int32 nameLen,
  gss_name_t *gssName,
  char **errormsg,
  db2int32  *errormsglen);

/* Functions to free memory held by the DLL */
SQL_API_RC SQL_API_FN db2secFreeToken(
  void *token,
  char **errormsg,
  db2int32  *errormsglen);

SQL_API_RC SQL_API_FN db2secFreeErrormsg(char *errormsg);

SQL_API_RC SQL_API_FN db2secFreeInitInfo(
  void *initInfo,
  char **errormsg,
  db2int32  *errormsglen);
```

The only API that must be resolvable externally is `db2secClientAuthPluginInit()`. This function will take a `void *` parameter, which should be cast to either:

```
typedef struct db2secUseridPasswordClientAuthFunctions_1
{
   db2int32 version;
   db2int32 plugintype;

SQL_API_RC (SQL_API_FN * db2secGetDefaultLoginContext)(
  char authid[DB2SEC_MAX_AUTHID_LENGTH],
  db2int32  *authidlen,
  char userid[DB2SEC_MAX_USERID_LENGTH],
  db2int32  *useridlen,
  db2int32 useridtype,
  char usernamespace[DB2SEC_MAX_USERNAMESPACE_LENGTH],
  db2int32  *usernamespacelen,
  db2int32  *usernamespacetype,
  const char *dbname,
  db2int32  dbnamelen,
  void **token,
  char **errormsg,
  db2int32  *errormsglen);

SQL_API_RC (SQL_API_FN * db2secRemapUserid)(  // Optional
  char userid[DB2SEC_MAX_USERID_LENGTH],
  db2int32  *useridlen,
  char usernamespace[DB2SEC_MAX_USERNAMESPACE_LENGTH],
  db2int32  *usernamespacelen,
  db2int32  *usernamespacetype,
  char password[DB2SEC_MAX_PASSWORD_LENGTH],
  db2int32  *passwordlen,
  char newpassword[DB2SEC_MAX_PASSWORD_LENGTH],
  db2int32  *newpasswordlen,
  const char *dbname,
  db2int32  dbnamelen,
  char **errormsg,
  db2int32  *errormsglen);

SQL_API_RC (SQL_API_FN * db2secValidatePassword)(
  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,
  db2Uint32   connection_details,
  void **token,
  char **errormsg,
  db2int32  *errormsglen);

SQL_API_RC (SQL_API_FN * db2secFreeToken)(
  void **token,
  char **errormsg,
  db2int32  *errormsglen);

SQL_API_RC (SQL_API_FN * db2secFreeErrormsg)(char *errormsg);


SQL_API_RC (SQL_API_FN * db2secClientAuthPluginTerm)(
  char **errormsg,
  db2int32  *errormsglen);
}
```

or

```
typedef struct db2secGssapiClientAuthFunctions_1
{
  db2int32 version;
  db2int32 plugintype;

SQL_API_RC (SQL_API_FN * db2secGetDefaultLoginContext) (
  char authid[DB2SEC_MAX_AUTHID_LENGTH],
  db2int32  *authidlen,
  char userid[DB2SEC_MAX_USERID_LENGTH],
  db2int32  *useridlen,
  db2int32 useridtype,
  char usernamespace[DB2SEC_MAX_USERNAMESPACE_LENGTH],
  db2int32  *usernamespacelen,
  db2int32  *usernamespacetype,
  const char *dbname,
  db2int32  dbnamelen,
  void **token,
  char **errormsg,
  db2int32  *errormsglen);

SQL_API_RC (SQL_API_FN * db2secProcessServerPrincipalName) (
  const void *data,
  gss_name_t *gssName,
  char **errormsg,
  db2int32  *errormsglen);

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);

SQL_API_RC (SQL_API_FN * db2secFreeToken)(
  void *token,
```

```
  char **errormsg,
  db2int32  *errormsglen);

SQL_API_RC (SQL_API_FN * db2secFreeErrormsg)(char *errormsg);


SQL_API_RC (SQL_API_FN * db2secFreeInitInfo) (
  void *initInfo,
  char **errormsg,
  db2int32  *errormsglen);

SQL_API_RC (SQL_API_FN * db2secClientAuthPluginTerm) (
  char **errormsg,
  db2int32  *errormsglen);

/* GSS-API specific functions -- refer to db2secPlugin.h
   for parameter list*/

  OM_uint32 (SQL_API_FN * gss_init_sec_context )(<parameter list>);
  OM_uint32 (SQL_API_FN * gss_delete_sec_context )(<parameter list>);
  OM_uint32 (SQL_API_FN * gss_display_status )(<parameter list>);
  OM_uint32 (SQL_API_FN * gss_release_buffer )(<parameter list>);
  OM_uint32 (SQL_API_FN * gss_release_cred )(<parameter list>);
  OM_uint32 (SQL_API_FN * gss_release_name )(<parameter list>);
  }
```

You should use `db2secUseridPasswordClientAuthFunctions_1` if you are writing an user ID/password plug-in. If you are writing a GSS-API (including Kerberos) plug-in, you should use `db2secGssapiClientAuthFunctions_1`.

For the user ID/password plug-in library, you will need to implement the following server-side APIs:

```
db2secServerAuthPluginInit(
  db2int32 version,
  void *server_fns,
  db2secGetConDetails *getConDetails_fn,
  db2secLogMessage *logMessage_fn,
  char **errormsg,
  db2int32  *errormsglen);
```

The above function takes as input a pointer to a function, `*logMessage_fn`, and a function, `*getConDetails_fn`, with the following prototypes:

```
SQL_API_RC (SQL_API_FN db2secLogMessage) (
  db2int32 level,
  void *data,
  db2int32 length);

SQL_API_RC (SQL_API_FN db2secGetConDetails)(
  db2int32 conDetailsVersion,
  const void *pConDetails);
```

This function in turn, takes as its second parameter, `pConDetails`, a pointer to a structure defined as follows:

```
typedef struct db2sec_con_details_1
{
  db2int32 clientProtocol;
  db2Uint32 clientIPAddress;
  db2Uint32 connect_info_bitmap;
  db2int32 dbnameLen;
  char dbname[DB2SEC_MAX_DBNAME_LENGTH + 1];
} db2sec_con_details_1;
```

See the detailed description section for an explanation of this function and structure.

```
db2secServerAuthPluginTerm(
  char **errormsg,
  db2int32  *errormsglen);

SQL_API_RC SQL_API_FN db2secValidatePassword(
  const char *userid,
  db2int32  useridlen,
  const char *usernamespace,
  db2int32  usernamespacelen,
  db2int32  usernamespacetype,
  const char *password,
  db2int32  passwordlen,
  const char *newpasswd,
  db2int32  newpasswdlen,
  const char *dbname,
  db2int32  dbnamelen,
  db2Uint32  connection_details,
  void **token,
  char **errormsg,
  db2int32  *errormsglen);

SQL_API_RC SQL_API_FN db2secGetAuthIDs(
  const char *userid,
  db2int32  useridlen,
  const char *usernamespace,
  db2int32  usernamespacelen,
  db2int32  usernamespacetype,
  const char *dbname,
  db2int32  dbnamelen,
  void **token,
  char SystemAuthid[DB2SEC_MAX_AUTHID_LENGTH],
  db2int32  SystemAuthidlen,
  char InitialSessionAuthID[DB2SEC_MAX_AUTHID_LENGTH],
  db2int32  *InitialSessionAuthIdlen,
  char username[DB2SEC_MAX_USERID_LENGTH],
  db2int32  *usernamelen,
  db2int32 *initsessionidtype,
  char **errormsg,
  db2int32  *errormsglen);

SQL_API_RC SQL_API_FN db2secDoesAuthIDExist(
  const char *authid,
  db2int32 authidlen,
  const char *errormsg,
  db2int32  *errormsglen);

SQL_API_RC SQL_API_FN db2secFreeToken(
  void *token,
  char **errormsg,
  db2int32  *errormsglen);

SQL_API_RC SQL_API_FN db2secFreeErrormsg(char *errormsg);
```

The only API that must be resolvable externally is `db2secServerAuthPluginInit()`. This function will take a `void *` parameter, which should be cast to either:

```
typedef struct db2secUseridPasswordServerAuthFunctions_1
{
  db2int32 version;
  db2int32 plugintype;

  /* parameter lists left blank for readability
    see above for parameters */
 SQL_API_RC (SQL_API_FN * db2secValidatePassword)(<parameter list>);
  SQL_API_RC (SQL_API_FN * db2secGetAuthIDs)(<parameter list);
  SQL_API_RC (SQL_API_FN * db2secDoesAuthIDExist)(<parameter list>);
```

```
  SQL_API_RC (SQL_API_FN * db2secFreeToken)(<parameter list>);
  SQL_API_RC (SQL_API_FN * db2secFreeErrormsg)(<parameter list>);
 SQL_API_RC (SQL_API_FN * db2secServerAuthPluginTerm)();
} userid_password_server_auth_functions;
```

or

```
typedef struct db2secGssapiServerAuthFunctions_1
{
 db2int32 version;
  db2int32 plugintype;
  gss_buffer_desc serverPrincipalName;
  gss_cred_id_t ServerCredHandle;
  SQL_API_RC (SQL_API_FN * db2secGetAuthIDs)(<parameter list);
  SQL_API_RC (SQL_API_FN * db2secDoesAuthIDExist)(<parameter list>);
  SQL_API_RC (SQL_API_FN * db2secFreeToken)(<parameter list>);
  SQL_API_RC (SQL_API_FN * db2secFreeErrormsg)(<parameter list>);
  SQL_API_RC (SQL_API_FN * db2secServerAuthPluginTerm)();

 /* GSS-API specific functions
 refer to db2secPlugin.h for parameter list*/
 OM_uint32 (SQL_API_FN * gss_accept_sec_context )(<parameter list>);
 OM_uint32 (SQL_API_FN * gss_display_name )(<parameter list>);
 OM_uint32 (SQL_API_FN * gss_delete_sec_context )(<parameter list>);
 OM_uint32 (SQL_API_FN * gss_display_status )(<parameter list>);
 OM_uint32 (SQL_API_FN * gss_release_buffer )(<parameter list>);
 OM_uint32 (SQL_API_FN * gss_release_cred )(<parameter list>);
 OM_uint32 (SQL_API_FN * gss_release_name )(<parameter list>);

} gssapi_server_auth_functions;
```

You should use `db2secUseridPasswordServerAuthFunctions_1` if you are writing an user ID/password plug-in. If you are writing a GSS-API (including Kerberos) plug-in, you should use `db2secGssapiServerAuthFunctions_1`.

Related concepts:

- "Security plug-ins" on page 1021
- "Security plug-in APIs" on page 1047

Related tasks:

- "Deploying a user ID/password plug-in" on page 1031

Related reference:

- "db2secGetGroupsForUser - Get list of groups for user function" on page 1052
- "db2secClientAuthPluginInit - Initialize client authentication plug-in" on page 1064
- "db2secClientAuthPluginTerm - Clean up client authentication plug-in resources function" on page 1065
- "db2secRemapUserid - Remap user ID and password function" on page 1065
- "db2secGetDefaultLoginContext - Get default login context function" on page 1067
- "db2secGenerateInitialCred - Generate initial credentials function" on page 1069
- "db2secValidatePassword - Validate password function" on page 1070
- "db2secProcessServerPrincipalName - Process service principal name returned from server function" on page 1073
- "db2secFreeToken - Free memory held by token function" on page 1073
- "db2secFreeInitInfo - Clean up resources held by db2secGenerateInitialCred() function" on page 1074

- "db2secServerAuthPluginInit - Initialize server authentication plug-in function" on page 1075
- "db2secServerAuthPluginTerm - Clean up server authentication plug-in resources function" on page 1077
- "db2secGetAuthIDs - Get authentication IDs function" on page 1077
- "db2secDoesAuthIDExist - Check if authentication ID exists function" on page 1079

db2secClientAuthPluginInit - Initialize client authentication plug-in

This is the initialization function for the plug-in library that DB2 calls immediately after loading the library. The `functions` pointer should be cast to the appropriate `client_auth_functions` structure for the interface version.

C API syntax:

```
SQL_API_RC SQL_API_FN db2secClientAuthPluginInit(
 db2int32 version,
 void *client_fns,
 db2secLogMessage *logMessage_fn,
 char **errormsg,
 db2int32  *errormsglen);
```

Input:

db2int32 version
The highest version number of the API that DB2 will currently support.

*db2secLogMessage *logMessage_fn*
A pointer to a function provided by DB2. The plug-in can call this function to log additional error strings to `db2diag.log` for debugging or informational purposes. The first parameter should use the define in `db2secPlugin.h` and the last two parameters are the message string and its length. The defines to be used in the first parameter are:

```
#define DB2SEC_LOG_NONE       0   -  No logging
#define DB2SEC_LOG_CRITICAL   1   -  Severe Error encountered
#define DB2SEC_LOG_ERROR      2   -  Error encountered
#define DB2SEC_LOG_WARNING    3   -  Warning
#define DB2SEC_LOG_INFO       4   -  Informational
```

If you use the DB2SEC_LOG_INFO define, the message text is only written to the `db2diag.log` if the *diaglevel* database manager configuration parameter is set to 4.

Output:

*void *client_fns*
A pointer to memory provided by DB2 for a `client_auth_functions` structure. In future versions of DB2, there can be different versions of the APIs, so this should be cast to a pointer to the `client_auth_functions` structure corresponding to the version of the API that the plug-in implements.

*char **errormsg*
A pointer to the address of an ASCII string allocated by the plug-in that can be returned in this parameter if the API is not successful.

*db2int32 *errormsglen*
A pointer to an integer that indicates the length of the error message string in *char **errormsg*.

Related concepts:

- "Security plug-ins" on page 1021
- "Security plug-in APIs" on page 1047

Related reference:

- "diaglevel - Diagnostic error capture level configuration parameter" in the *Administration Guide: Performance*
- "APIs for user ID/password authentication plug-ins" on page 1057

db2secClientAuthPluginTerm - Clean up client authentication plug-in resources function

This function will be called by DB2 just before DB2 unloads the plug-in. This function should do a proper cleanup of any resources the plug-in library holds, for instance, free any memory allocated by the plug-in, close files that are still open, and close network connections. The plug-in is responsible for keeping track of these resources so that they can later be freed.

C API syntax:

```
SQL_API_RC SQL_API_FN db2secClientAuthPluginTerm(
 char **errormsg
 db2int32  *errormsglen);
```

Output:

*char **errormsg*
A pointer to the address of an ASCII string allocated by the plug-in that can be returned in this parameter if the API is not successful.

*db2int32 *errormsglen*
A pointer to an integer that indicates the length of the error message string in *char **errormsg*.

Related concepts:

- "Security plug-ins" on page 1021
- "Security plug-in APIs" on page 1047

Related reference:

- "APIs for user ID/password authentication plug-ins" on page 1057

db2secRemapUserid - Remap user ID and password function

This function will be called on the client side to provide the ability to remap a given user ID and password (and possibly new password and usernamespace) to different values from those given at connect time. DB2 will only call this function if both a user ID and a password are supplied at connect time. The user ID and password are both required to prevent a plug-in from remapping a user ID by itself to a user ID/password pair. This function is optional and is not called if it is not provided.

C API syntax:

```
SQL_API_RC SQL_API_FN db2secRemapUserid(
  char userid[DB2SEC_MAX_USERID_LENGTH],
  db2int32  *useridlen,
  char usernamespace[DB2SEC_MAX_USERNAMESPACE_LENGTH],
  db2int32  *usernamespacelen,
  db2int32  *usernamespacetype,
  char password[DB2SEC_MAX_PASSWORD_LENGTH],
  db2int32  *passwordlen,
  char newpassword[DB2SEC_MAX_PASSWORD_LENGTH],
  db2int32  *newpasswordlen,
  const char *dbname,
  db2int32  dbnamelen,
  char **errormsg,
  db2int32  *errormsglen);
```

Input:

*const char *dbname*
The name of the database to which the client is connecting.

db2int32 dbnamelen
Length of the dbname.

Input/output:

char userid[DB2SEC_MAX_USERID_LENGTH]
The user ID to be remapped. If there is an input user ID value, then there must be an output user ID value that can be the same or different from the input user ID value. If there is no input user ID value, the plug-in should not return an output user ID value.

*db2int32 *useridlen*
Length of the user ID returned in the userid parameter.

char usernamespace[DB2SEC_MAX_USERNAMESPACE_LENGTH]
The namespace the user ID came from. It is optional to remap this. If usernamespace was not provided as input to this function, and it does provide a value as output, the usernamespace will only be used by DB2 for CLIENT authentication and disregarded for other authentication types.

*db2int32 *usernamespacelen*
Old and new length of the usernamespace. Under the limitation that the usernamespacetype must be DB2SEC_NAMESPACE_SAM_COMPATIBLE, the maximum length supported is 15 bytes.

*db2int32 *usernamespacetype*
Old and new new namespacetype. The only supported namespace type is DB2SEC_NAMESPACE_SAM_COMPATIBLE.

char password[DB2SEC_MAX_PASSWORD_LENGTH]
The password to be remapped. If a password was passed as input, there must an output password and can be a different password. If there is no password passed in as input, the plug-in should not return an output password.

*db2int32 *passwordlen*
Length of the password.

char newpassword[DB2SEC_MAX_PASSWORD_LENGTH]
A new password if the password is to be changed.

Note: This is the new password that will be passed by DB2 into the newpassword field of the db2secValidatePassword function on the client or the server (depending on the value of the *authentication* database manager configuration parameter). If a new password was passed as input, there must an output new password and can be a different new password. If there is no new password passed in as input, the plug-in should not return a new output password.

*db2int32 *newpasswordlen*
Length of the new password.

Output:

*char **errormsg*
A pointer to the address of an ASCII string allocated by the plug-in that can be returned in this parameter if the API is not successful.

*db2int32 *errormsglen*
A pointer to an integer that indicates the length of the error message string in *char **errormsg*.

Related concepts:

- "Security plug-ins" on page 1021
- "Security plug-in APIs" on page 1047

Related reference:

- "APIs for user ID/password authentication plug-ins" on page 1057

db2secGetDefaultLoginContext - Get default login context function

This function is called by DB2 to determine the user associated with the default login context, in other words, to determine the DB2 authid of the user invoking a DB2 command without explicitly specifying a user ID (either an implicit authentication to a database, or a local authorization). This function must return both an authid and a user ID.

C API syntax:

```
db2secGetDefaultLoginContext(
 char authid[DB2SEC_MAX_AUTHID_LENGTH],
 db2int32  *authidlen,
 char userid[DB2SEC_MAX_USERID_LENGTH],
 db2int32  *useridlen,
 db2int32  useridtype,
 char usernamespace[DB2SEC_MAX_USERNAMESPACE_LENGTH],
 db2int32  *usernamespacelen,
 db2int32  *usernamespacetype,
 const char *dbname,
 db2int32  dbnamelen,
 void **token,
 char **errormsg,
 db2int32  *errormsglen);
```

Input:

*const char *dbname*
This contains the name of the database being connected to if this call is being used in the context of a database connection. For local authorization actions or instance attachments, this parameter is NULL.

db2int32 dbnamelen
Length of the dbname.

db2int32 useridtype
Specifies if DB2 wants the real or effective user of the process.

Output:

char authid[DB2SEC_MAX_AUTHID_LENGTH]
The field in which the authid should be returned. The returned value must conform to DB2 authid naming questions, or the user will not be authorized to perform the requested action.

*db2int32 *authidlen*
Length of the authid returned.

char userid[DB2SEC_MAX_USERID_LENGTH]
The field in which the user ID should be returned.

*db2int32 *useridlen*
Length of the user ID returned.

*void **token*
A pointer to data allocated by the plug-in that the plug-in will later want to pass to subsequent authentication calls in the plug-in, or possibly to the group retrieval plug-in. The structure of this data is determined by the plug-in writer.

char usernamespace[DB2SEC_MAX_USERNAMESPACE_LENGTH]
Length of the returned namespace. Under the limitation that the usernamespacetype must be DB2SEC_NAMESPACE_SAM_COMPATIBLE, the maximum length supported is 15 bytes.

*db2int32 *usernamespacelen*
Length of the namespace returned. Under the limitation that the usernamespacetype must be DB2SEC_NAMESPACE_SAM_COMPATIBLE, the maximum length supported is 15 bytes.

*db2int32 *usernamespacetype*
As specified above. The only supported namespace type is DB2SEC_NAMESPACE_SAM_COMPATIBLE.

*char **errormsg*
A pointer to the address of an ASCII string allocated by the plug-in that can be returned in this parameter if the API is not successful.

*db2int32 *errormsglen*
A pointer to an integer that indicates the length of the error message string in *char **errormsg*.

Related concepts:

- "Security plug-ins" on page 1021
- "Security plug-in APIs" on page 1047

Related reference:

- "authentication - Authentication type" on page 783
- "APIs for user ID/password authentication plug-ins" on page 1057

db2secGenerateInitialCred - Generate initial credentials function

This function obtains the initial GSS-API credentials based on the user ID and password that are passed in. For Kerberos this will be the TGT. The credential handle that is returned in `pGSSCredHandle` is the handle that is used with `gss_init_sec_context()`, and must be either an INITIATE or BOTH credential. This function is only called when a user ID, and possibly a password are supplied. Otherwise, DB2 will specify `GSS_C_NO_CREDENTIAL` when calling `gss_init_sec_context()` to signify that the default credential obtained from the current login context is to be used.

C API syntax:

```
db2secGenerateInitialCred(
 const char *userid,
 db2int32  useridlen,
 const char *usernamespace,
 db2int32  usernamspacelen,
 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);
```

Input:

*const char *userid*
The user ID whose password is to be verified.

db2int32 useridlen
Length of the user ID.

*const char *usernamespace*
The namespace from which the user ID was obtained.

db2int32 usernamespacelen
Length of the namespace field.

db2int32 usernamespacetype
The type of namespace.

*const char *password*
The password to be verified. This will be unencrypted by DB2 before being passed to the plug-in.

db2int32 passwordlen
Length of the newpassword.

*const char *newpassword*
A new password if the password is to be changed. If no change is requested, this is NULL. If this is non-NULL, the function should validate the old password before changing to the new password. The plug-in does not have to honour a request to change the password, but if it does not, it should immediately return DB2SEC_PLUGIN_CHANGEPASSWORD_NOTSUPPORTED without validating the old password.

db2int32 newpasswordlen
Length of the newpassword.

*const char *dbname*
The name of the database being connected to. This function is free to ignore this, or this function can return DB2SEC_PLUGIN_CONNECTION_DISALLOWED if the plug-in writer wants to restrict access to certain databases to users who otherwise have valid passwords.

db2int32 dbnamelen
Length of the dbname.

Output:

*gss_cred_id_t *pGSSCredHandle*
Pointer to the GSS-API credential handle.

*void **initInfo*
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. DB2 will call `db2secFreeInitInfo()` at the end of authentication, at which point the plug-in can then free these resources. If the plug-in does not need to maintain such a list, it should return NULL.

*char **errormsg*
A pointer to the address of an ASCII string allocated by the plug-in that can be returned in this parameter if the API 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.

*db2int32 *errormsglen*
A pointer to an integer that indicates the length of the error message string in *char **errormsg*.

Related concepts:

- "Security plug-ins" on page 1021
- "Security plug-in APIs" on page 1047

Related reference:

- "APIs for user ID/password authentication plug-ins" on page 1057

db2secValidatePassword - Validate password function

This function provides a user ID-and-password-style authentication method during a database connect operation.

Note: The plug-in code is run with the privileges of the client application. The plug-in writer should take this into consideration if authentication of the user requires special privileges (such as root).

This API should return DB2SEC_PLUGIN_OK (success) if the password is valid, or an error code such as DB2SEC_PLUGIN_BADPWD if the password is invalid.

This API is only called on the client side if *authentication* is set to CLIENT.

C API syntax:

```
SQL_API_RC SQL_API_FN db2secValidatePassword(
 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,
  db2Uint32   connection_details,
  void **token,
  char **errormsg,
  db2int32  *errormsglen);
```

Input:

*const char *userid*
: The user ID whose password is to be verified.

db2int32 useridlen
: Length of the user ID.

*const char *password*
: The password to be verified. This will be unencrypted by DB2 before being passed in.

db2int32 passwordlen
: Length of the password given.

*const char *newpassword*
: A new password, if the password is to be changed. If no change is requested, this parameter is NULL. If this parameter is not NULL, the function should validate the old password before changing to the new password. The plug-in does not have to fulfill a request to change the password, but if it does not, it should immediately return DB2SEC_PLUGIN_CHANGEPASSWORD_NOTSUPPORTED without validating the old password.

db2int32 newpasswordlen
: Length of the newpassword.

*const char *dbname*
: The name of the database being connected to. The function is free to ignore this, or it can return DB2SEC_PLUGIN_CONNECTIONREFUSED if the plug-in writer wants to restrict access to certain databases to users who otherwise have valid passwords.

db2int32 dbnamelen
: Length of the dbname.

db2int32 usernamespace
: The namespace from which the user ID was obtained.

db2int32 usernamespacelen
: Length of the namespace field.

db2int32 usernamespacetype
: The type of namespace. Possible values are:
DB2SEC_NAMESPACE_SAM_COMPATIBLE (corresponding to a username style like torolab\myname"), or DB2SEC_NAMESPACE_USER_PRINCIPAL

(corresponding to a username style like myname@torolab.ibm.com). Currently DB2 only supports DB2SEC_NAMESPACE_SAM_COMPATIBLE. When the user ID is not available, this will be filled with DB2SEC_USER_NAMESPACE_UNDEFINED. All the defines are located in `db2secPlugin.h`.

db2Uint32 connection_details
A bit field consisting of 2 fields:

- 1 bit indicates whether the connection is local (using IPC or connect from one of the nodes in the `db2nodes.cfg` in the partitioned database environment), or remote (through network or loopback). This gives the plug-in the ability to decide whether clients on the same machine can connect to the DB2 server without a password. Currently, DB2 always allows local connections without a password from clients on the same machine (assuming the client has CONNECT authority to the database).
- 1 bit indicates whether the source of the user ID is the default from `db2secGetDefaultLoginContext`, or was explicitly provided during the connect. The bit values are defined in `db2secPlugin.h`:

  ```
  #define DB2SEC_USERID_FROM_OS 0x00000001
  ```

  `DB2SEC_USERID_FROM_OS` indicates user ID is obtained from the operating system and is not explicitly given on the connect statement.

  ```
  #define DB2SEC_CONNECTION_ISLOCAL 0x00000002
  ```

  `DB2SEC_CONNECTION_ISLOCAL` indicates a local connection.

  ```
  #define DB2SEC_VALIDATING_ON_SERVER_SIDE 0x0000004
  ```

  `DB2SEC_VALIDATING_ON_SERVER_SIDE` indicates whether DB2 is calling from the server side for validating the password.

  DB2's default behavior for an implicit authentication is to allow the connection without any password validation. However, plug-in writers can disallow implicit authentication by returning a `DB2SEC_PLUGIN_BADPASSWORD` error.

*void **token*
A pointer to data that can be passed into subsequent plug-in API calls (`db2secGetAuthIDs`, `db2secGetGroupsForUser`) during this connection.

Output:

*char **errormsg*
A pointer to the address of an ASCII string allocated by the plug-in that can be returned in this parameter if the API is not successful.

*db2int32 *errormsglen*
A pointer to an integer that indicates the length of the error message string in *char **errormsg*.

Related concepts:

- "Security plug-ins" on page 1021
- "Security plug-in APIs" on page 1047

Related reference:

- "APIs for user ID/password authentication plug-ins" on page 1057

db2secProcessServerPrincipalName - Process service principal name returned from server function

This function processes the service principal name returned from the server and returns the principal name in the `gss_name_t` internal format to be used with `gss_init_sec_context()`. This function is also called to process the service principal name cataloged with the database directory when Kerberos authentication is used. Ordinarily, this conversion uses the `gss_import_name()` API. After the context is established, the `gss_name_t` object is freed through the call to `gss_release_name()`. The function returns `DB2SEC_PLUGIN_OK` if `gssName` points to a valid GSS name; a `DB2SEC_PLUGIN_BAD_PRINCIPAL_NAME` error code is returned if the principal name is invalid.

C API syntax:

```
SQL_API_RC SQL_API_FN db2secProcessServerPrincipalName(
 const void *name,
 db2int32 nameLen,
 gss_name_t *gssName,
 char **errormsg,
 db2int32  *errormsglen);
```

Input:

*const void *name*
: Text name of the service principal in GSS_C_NT_USER_NAME format, for example, service/host@REALM.

db2int32 nameLen
: Length of the text service principal name.

Output:

*gss_name_t *gssName*
: Pointer to the output service principal name in the GSS-API internal format

*char **errormsg*
: A pointer to the address of an ASCII string allocated by the plug-in that can be returned in this parameter if the API is not successful.

*db2int32 *errormsglen*
: A pointer to an integer that indicates the length of the error message string in *char **errormsg*.

Related concepts:

- "Security plug-ins" on page 1021
- "Security plug-in APIs" on page 1047

Related reference:

- "APIs for user ID/password authentication plug-ins" on page 1057

db2secFreeToken - Free memory held by token function

This function is called by DB2 when DB2 no longer needs the memory held by `token`. The plug-in must free the memory.

C API syntax:

```
SQL_API_RC SQL_API_FN db2secFreeToken(
 void *token
 char **errormsg,
 db2int32  *errormsglen);
```

Input:

*void *token*
Pointer to the memory to be freed.

Output:

*char **errormsg*
A pointer to the address of an ASCII string allocated by the plug-in that can be returned in this parameter if the API is not successful.

*db2int32 *errormsglen*
A pointer to an integer that indicates the length of the error message string in *char **errormsg*.

Related concepts:

- "Security plug-ins" on page 1021
- "Security plug-in APIs" on page 1047

Related reference:

- "APIs for user ID/password authentication plug-ins" on page 1057

db2secFreeInitInfo - Clean up resources held by db2secGenerateInitialCred() function

This function frees all resources allocated by `db2secGenerateInitialCred()`. Resources include, for example, handles to underlying mechanism contexts or a credential cache created for the GSS-API credential cache.

C API syntax:

```
SQL_API_RC SQL_API_FN db2secFreeInitInfo(
 void *initinfo,
 char **errormsg,
 db2int32  *errormsglen);
```

Input:

*void *initinfo*
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. These resources are freed by calling this API.

Output:

*char **errormsg*
A pointer to the address of an ASCII string allocated by the plug-in that can be returned in this parameter if the API is not successful.

*db2int32 *errormsglen*
A pointer to an integer that indicates the length of the error message string in *char **errormsg*.

Related concepts:

- "Security plug-ins" on page 1021
- "Security plug-in APIs" on page 1047

Related reference:

- "APIs for user ID/password authentication plug-ins" on page 1057

db2secServerAuthPluginInit - Initialize server authentication plug-in function

This is the initialization function for the library that DB2 calls immediately after loading the library. The `functions` pointer should be cast to the appropriate `server_auth_functions` structure for the interface version. For GSS-API, the plug-in is responsible for filling in the server's principal name in the `serverPrincipalName` variable inside the `gssapi_server_auth_functions` structure at initialization time, and for providing the server's credential handle in the *serverCredHandle* variable. The freeing of the memory that is allocated to hold the principal name and the credential handle must be done by the `db2secServerAuthPluginTerm()` cleanup function.

C API syntax:

```
SQL_API_RC SQL_API_FN db2secServerAuthPluginInit(
 db2int32 version,
  void *server_fns,
  db2secGetConDetails *getConDetails_fn,
  db2secLogMessage *logMessage_fn,
  char **errormsg,
  db2int32  *errormsglen);
```

Input:

db2int32 version
The highest version number of the API that DB2 currently supports.

*db2secGetConDetails *getConDetails_fn*
A pointer to a function provided by DB2. The plug-in can call this function in any one of the other authentication APIs to obtain details related to the database connection. These details will include information about the communication mechanism associated with the connection (such as the IP address, if TCP/IP is used), which the plug-in writer might need to reference when making authentication decisions. For instance, the plug-in could disallow a connection for a particular user unless that user is connecting from a particular IP address. The use of this callback is optional.

If the callback is called in a situation not involving a database connection, this function returns `DB2SEC_PLUGIN_NO_CON_DETAILS`; otherwise, this function returns 0 on success.

The parameter `getConDetails_fn` takes two input parameters, a pointer to the `db2sec_con_details` structure, and a version number indicating which `db2sec_con_details` structure to use. The current version number is 1. Upon a successful return, the `db2sec_con_details` structure is filled out with the following information:

- The protocol used for the connection to the server. The listing of protocol definitions can be found in file `sqlenv.h` (SQL_PROTOCOL_*).

- The TCP/IP address of the inbound connect to the server if the protocol is TCP/IP.
- The database name the client is attempting to connect to. The database name is not set for instance attachments.
- A connection information bit-map that contains the same details as documented in the `connection_details` parameter of the `db2secValidatePassword()` API.

*db2secLogMessage *logMessage_fn*
A pointer to a function provided by DB2. The plug-in can call this function to log additional error strings to `db2diag.log` for debugging or informational purposes. The first parameter should use the define in `db2secPlugin.h` and the last two parameters are the message string and its length. The defines to be used in the first parameter are:

```
#define DB2SEC_LOG_NONE       0   -  No logging
#define DB2SEC_LOG_CRITICAL   1   -  Severe Error encountered
#define DB2SEC_LOG_ERROR      2   -  Error encountered
#define DB2SEC_LOG_WARNING    3   -  Warning
#define DB2SEC_LOG_INFO       4   -  Informational
```

If you use the DB2SEC_LOG_INFO define, the message text is only written to the `db2diag.log` if the *diaglevel* database manager configuration parameter is set to 4.

Output:

*void *server_fns*
A pointer to memory provided by DB2 for a `server_auth_functions` structure. In future versions of DB2, there can be different versions of the APIs, so this should be cast to a pointer to the `server_auth_functions` structure corresponding to the version of the API that the plug-in implements.

Inside the `server_auth_functions`, the `plugintype` variable should be set to one of DB2SEC_PLUGIN_TYPE_USERID_PASSWORD, DB2SEC_PLUGIN_TYPE_GSSAPI, or DB2SEC_PLUGIN_TYPE_KERBEROS. Other values can be defined in future versions of the API.

*char **errormsg*
A pointer to the address of an ASCII string allocated by the plug-in that can be returned in this parameter if the API is not successful.

*db2int32 *errormsglen*
A pointer to an integer that indicates the length of the error message string in *char **errormsg*.

Related concepts:

- "Security plug-ins" on page 1021
- "Security plug-in APIs" on page 1047

Related reference:

- "diaglevel - Diagnostic error capture level configuration parameter" in the *Administration Guide: Performance*
- "APIs for user ID/password authentication plug-ins" on page 1057

db2secServerAuthPluginTerm - Clean up server authentication plug-in resources function

This function is called by DB2 just before DB2 unloads the plug-in. This function should do a proper cleanup of any resources the plug-in library holds, for instance, free any memory allocated by the plug-in, close files that are still open, and close network connections. The plug-in is responsible for keeping track of these resources so that they can later be freed.

C API syntax:

```
SQL_API_RC SQL_API_FN db2secServerAuthPluginTerm(char **errormsg,
 db2int32  *errormsglen);
```

Output:

*char **errormsg*
A pointer to the address of an ASCII string allocated by the plug-in that can be returned in this parameter if the API is not successful.

*db2int32 *errormsglen*
A pointer to an integer that indicates the length of the error message string in *char **errormsg*.

Related concepts:

- "Security plug-ins" on page 1021
- "Security plug-in APIs" on page 1047

Related reference:

- "APIs for user ID/password authentication plug-ins" on page 1057

db2secGetAuthIDs - Get authentication IDs function

This function returns an SQL authid for an authenticated user. This function will be called during database connections for both user ID/password and GSS-API authentication methods.

C API syntax:

```
SQL_API_RC SQL_API_FN db2secGetAuthIDs(
 const char *userid,
 db2int32  useridlen,
 const char *usernamespace,
 db2int32  usernamespacelen,
 db2int32  usernamespacetype,
 const char *dbname,
 db2int32  dbnamelen,
 void **token,
 char SystemAuthID[DB2SEC_MAX_AUTHID_LENGTH],
 db2int32  *SystemAuthIDlen,
 char InitialSessionAuthID[DB2SEC_MAX_AUTHID_LENGTH],
 db2int32  *InitialSessionAuthIDlen,
 char username[DB2SEC_MAX_USERID_LENGTH],
 db2int32  *usernamelen,
 db2int32 *initsessionidtype,
 char **errormsg,
 db2int32  *errormsglen);
```

Input:

*const char * userid*
The authenticated user. This is blank for GSS-API.

db2int32 useridlen
Length of the user ID.

*void **token*
Data that the plug-in might pass to the `db2secGetGroupsForUser` call. For GSS-API, this is a context handle (`gss_ctx_id_t`). Ordinarily, this is an input-only parameter and its value is taken from `db2secValidatePassword`. This value can also be an output parameter when authentication is done on the client (and `db2secValidatePassword`) is not called.

*const char *dbname*
The name of the database being connected to. The plug-in can ignore this, or it can return differing authids when the same user connects to different databases.

db2int32 dbnamelen
Length of the dbname.

*const char *usernamespace*
The namespace from which the user ID was obtained.

db2int32 usernamespacelen
Length of the namespace field.

db2int32 usernamespacetype
As specified above. The only supported namespace type is DB2SEC_NAMESPACE_SAM_COMPATIBLE.

Output:

char SystemAuthID[DB2SEC_MAX_AUTHID_LENGTH]
The system authid corresponds to the ID of the authenticated user. The size is 255, but DB2 uses up to 30.

*db2int32 *SystemAuthiIDlen*
Length of the `SystemAuthId` returned.

char InitialSessionAuthid[DB2SEC_MAX_AUTHID_LENGTH]
This is the authid used for this connection session. The authid is usually the same as the `SystemAuthID`, but can be different in some situations, for instance, when issuing a SET SESSION AUTHORIZATION statement. Size is 255, but DB2 only uses up to 30.

*db2int32 *InitialSessionAuthidlen*
Length of the `InitialSessionAuthID` returned.

char username[DB2SEC_MAX_USERID_LENGTH]
A username corresponding to the authenticated user and authid. This will only be used for auditing and is logged in the ″User ID″ field. If the plug-in does not fill in this field, DB2 copies it from the userid.

*db2int32 *usernamelen*
Length of the user ID returned.

*db2int32 *initsessionidtype*
Session authid type indicating whether or not the `InitialSessionAuthid` is a role or an authid. The plug-in should return the one of the following (defined in `db2secPlugin.h`): `DB2SEC_ID_TYPE_AUTHID` (0) or `DB2SEC_ID_TYPE_ROLE` (1). Currently, DB2 only supports authid (DB2SEC_ID_TYPE_AUTHID).

*char **errormsg*
A pointer to the address of an ASCII string allocated by the plug-in that can be returned in this parameter if the API is not successful.

*db2int32 *errormsglen*
A pointer to an integer that indicates the length of the error message string in *char **errormsg*.

Related concepts:

- "Security plug-ins" on page 1021
- "Security plug-in APIs" on page 1047

Related reference:

- "APIs for user ID/password authentication plug-ins" on page 1057

db2secDoesAuthIDExist - Check if authentication ID exists function

This function determines if the authid represents an individual user (for instance, whether the function can map the authid to an external user id). This function should return DB2SEC_PLUGIN_OK if it is successful - the authid is valid, DB2SEC_PLUGIN_INVALID_USERORGROUP if it is not valid, or DB2SEC_PLUGIN_USERSTATUSNOTKNOWN if the authid cannot be determined to exist.

C API syntax:

```
SQL_API_RC SQL_API_FN db2secDoesAuthIDExist(
 const char *authid,
 db2int32  authidlen,
 const char *errormsg,
 db2int32  *errormsglen);
```

Input:

*const char *authid*
The authid to validate. This value is folded to upper case, with no trailing blanks.

db2int32 authidlen
Length of the authid.

Output:

*char **errormsg*
A pointer to the address of an ASCII string allocated by the plug-in that can be returned in this parameter if the API is not successful.

*db2int32 *errormsglen*
A pointer to an integer that indicates the length of the error message string in *char **errormsg*.

Related concepts:

- "Security plug-ins" on page 1021
- "Security plug-in APIs" on page 1047

Related reference:

- "APIs for user ID/password authentication plug-ins" on page 1057

GSS-API plug-in APIs

Required APIs and Definitions for GSS-API authentication plug-ins

Following is a complete list of GSS-APIs required for the DB2 security plug-in interface. The supported APIs follow these specifications: *Generic Security Service Application Program Interface, Version 2* (IETF RFC2743) and *Generic Security Service API Version 2: C-Bindings* (IETF RFC2744). Before implementing a GSS-API based plug-in, you should have a complete understanding of these specifications.

Table 84. Required APIs and Definitions for GSS-API authentication plug-ins

Name		Description
Client-side APIs		
	`gss_init_sec_context`	Initiate a security context with a peer application.
Server-side APIs		
	`gss_accept_sec_context`	Accept a security context initiated by a peer application.
	`gss_display_name`	Convert an internal format name to text.
Common APIs		
	`gss_delete_sec_context`	Delete an established security context.
	`gss_display_status`	Obtain the text error message associated with a GSS-API status code.
	`gss_release_buffer`	Delete a buffer.
	`gss_release_cred`	Release local data structures associated with a GSS-API credential.
	`gss_release_name`	Delete internal format name.
Required definitions		
	`GSS_C_DELEG_FLAG`	Requests delegation.
	`GSS_C_EMPTY_BUFFER`	Signifies that the `gss_buffer_desc` does not contain any data.
	`GSS_C_GSS_CODE`	Indicates a GSS major status code.
	`GSS_C_INDEFINITE`	Indicates that the mechanism does not support context expiration.
	`GSS_C_MECH_CODE`	Indicates a GSS minor status code.
	`GSS_C_MUTUAL_FLAG`	Mutual authentication requested.
	`GSS_C_NO_BUFFER`	Signifies that the `gss_buffer_t` variable does not point to a valid `gss_buffer_desc` structure.
	`GSS_C_NO_CHANNEL_BINDINGS`	No communication channel bindings.
	`GSS_C_NO_CONTEXT`	Signifies that the `gss_ctx_id_t` variable does not point to a valid context.
	`GSS_C_NO_CREDENTIAL`	Signifies that `gss_cred_id_t` variable does not point to a valid credential handle.
	`GSS_C_NO_NAME`	Signifies that the `gss_name_t` variable does not point to a valid internal name.
	`GSS_C_NO_OID`	Use default authentication mechanism.
	`GSS_C_NULL_OID_SET`	Use default mechanism.

Table 84. Required APIs and Definitions for GSS-API authentication plug-ins (continued)

Name		Description
	GSS_S_COMPLETE	API completed successfully.
	GSS_S_CONTINUE_NEEDED	Processing is not complete and the API must be called again with the reply token received from the peer.

Related concepts:

- "Security plug-ins" on page 1021
- "Security plug-in APIs" on page 1047

Related reference:

- "Restrictions for GSS-API authentication plug-ins" on page 1081

Restrictions for GSS-API authentication plug-ins

The following is a list of restrictions for GSS-API authentication plug-ins.

- The default security mechanism is always assumed; therefore, there is no OID consideration.
- The only GSS services requested in `gss_init_sec_context()` are mutual authentication and delegation. DB2 always requests a ticket for delegation, but does not use that ticket to generate a new ticket.
- Only the default context time is requested.
- Context tokens from `gss_delete_sec_context()` are not sent from the client to the server and vice-versa.
- Anonymity is not supported.
- Channel binding is not supported
- If the initial credentials expire, DB2 does not automatically renew them.
- The GSS-API specification stipulates that even if `gss_init_sec_context()` or `gss_accept_sec_context()` fail, either function must return a token to send to the peer. However, because of DRDA limitations, DB2 only sends a token if `gss_init_sec_context()` fails and generates a token on the first call.

Related concepts:

- "Security plug-ins" on page 1021
- "Security plug-in APIs" on page 1047

Related reference:

- "Required APIs and Definitions for GSS-API authentication plug-ins" on page 1080

Security plug-in API versioning

Because it is possible that future releases of DB2® will need different versions of the security plug-in APIs, DB2 supports version numbering of the APIs. These version numbers are integers starting with 1 for DB2 UDB version 8.2. The version number that DB2 passes to the security plug-in APIs are the highest version number of the API that DB2 can support, and correspond to the version number of the structure. If the plug-in can support a higher API version, it must return function pointers for the version that DB2 has requested. If the plug-in only supports a lower version of the API, the plug-in should fill in function pointers for

the lower version. In either situation, the security plug-in APIs should return the version number for the API it is supporting in the version field of the functions structure.

For DB2, the version numbers of the security plug-ins will only change when necessary (for example, when there are changes to the parameters of the APIs). Version numbers will not automatically change with DB2 release numbers.

Related concepts:

- "Security plug-ins" on page 1021
- "Security plug-in APIs" on page 1047

Chapter 31. Security plug-in deployment limitations

The following are limitations on the use of security plug-ins:

DB2 Universal JDBC Driver support limitations:

The DB2® Universal JDBC Driver does not support the client side plug-in authentication model. You cannot use a GSS-API authentication plug-in to connect to a DB2 Universal Database (DB2 UDB) for Linux, UNIX®, and Windows® server from a DB2 Universal JDBC Driver client. A DB2 Universal JDBC Driver client can only use the supported operating system level authentication mechanism or the Kerberos authentication method. This limitation applies to both Type 2 and Type 4 connectivity.

Specifically, the server's database manager configuration parameter *srvcon_auth* cannot be set to GSSPLUGIN if the database manager configuration parameter *srvcon_gssplugin_list* value does not contain the name of a Kerberos based GSS-API plug-in.

The *srvcon_auth* parameter can however be set to any of: CLIENT, SERVER, SERVER_ENCRYPT, KERBEROS, KRB_SERVER_ENCRYPT, GSS_SERVER_ENCRYPT, DATA_ENCRYPT, or DATA_ENCRYPT_CMP.

DB2 UDB family support limitations:

You cannot use a GSS-API plug-in to authenticate connections between DB2 UDB clients on Linux, UNIX, and Windows and another DB2 UDB family server. You also cannot authenticate connections from another DB2 UDB family server, acting as a client, to a DB2 UDB server on Linux, UNIX, or Windows.

If you use a DB2 UDB client on Linux, UNIX, or Windows to connect to other DB2 UDB family servers, you can use client-side user ID/password plug-ins (such as the IBM®-shipped operating system authentication plug-in), or you can write your own user ID/password plug-in. You can also use the built-in Kerberos plug-ins, or implement your own.

With a DB2 UDB client on Linux, UNIX, or Windows, you should not catalog a database using the GSSPLUGIN authentication type.

DB2 Information Integrator support limitations:

DB2 II does not support the use of delegated credentials from a GSS_API plug-in to establish outbound connections to data sources. Connections to data sources must continue to use the CREATE USER MAPPING command.

Database Administration Server support limitations:

The DB2 Administration Server (DAS) does not support security plug-ins. The DAS only supports the operating system authentication mechanism.

© Copyright IBM Corp. 1993-2004

Chapter 32. Security Plug-In Configuration Parameters

clnt_krb_plugin - Client Kerberos plug-in

Configuration Type Database manager

Applies to

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type Configurable

Default [Range] Null or IBMkrb5 [any valid string]

This parameter specifies the name of the default Kerberos plug-in library to be used for client-side authentication and local authorization. By default, the value is null on UNIX-based systems, and `IBMkrb5` on Windows operating systems. This plug-in is used when the client is authenticated using KERBEROS authentication.

Related reference:

- "GET DATABASE MANAGER CONFIGURATION" on page 281
- "RESET DATABASE MANAGER CONFIGURATION Command" in the *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION" on page 384

clnt_pw_plugin - Client userid-password plug-in

Configuration Type Database manager

Applies to

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type Configurable

Default [Range] Null [any valid string]

This parameter specifies the name of the userid-password plug-in library to be used for client-side authentication and local authorization. By default, the value is null and the DB2-supplied userid-password plug-in library is used. The plug-in is used when the client is authenticated using CLIENT, SERVER, or SERVER_ENCRYPT authentication.

Related reference:

- "GET DATABASE MANAGER CONFIGURATION" on page 281
- "RESET DATABASE MANAGER CONFIGURATION Command" in the *Command Reference*

© Copyright IBM Corp. 1993-2004

- "UPDATE DATABASE MANAGER CONFIGURATION" on page 384

group_plugin - Group plug-in

Configuration Type Database manager

Applies to

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type Configurable

Default [Range] Null [any valid string]

This parameter specifies the name of the group plug-in library. By default, this value is null, and DB2 uses the operating system group lookup. The plug-in will be used for all group lookups.

Related reference:

- "GET DATABASE MANAGER CONFIGURATION" on page 281
- "RESET DATABASE MANAGER CONFIGURATION Command" in the *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION" on page 384

local_gssplugin - GSS API plug-in used for local instance level authorization

Configuration Type Database manager

Applies to

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type Configurable

Default [Range] Null [any valid string]

This parameter specifies the name of the default GSS API plug-in library to be used for instance level local authorization when the value of the *authentication* database manager configuration parameter is set to GSSPLUGIN or GSS_SERVER_ENCRYPT.

Related reference:

- "GET DATABASE MANAGER CONFIGURATION" on page 281
- "RESET DATABASE MANAGER CONFIGURATION Command" in the *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION" on page 384

srvcon_auth - Authentication type for incoming connections at the server

Configuration Type	Database manager
Applies to	• Database server with local and remote clients • Database server with local clients • Partitioned database server with local and remote clients
Parameter Type	Configurable
Default [Range]	Null [CLIENT; SERVER; SERVER_ENCRYPT; KERBEROS; KRB_SERVER_ENCRYPT; GSSPLUGIN; GSS_SERVER_ENCRYPT]

This parameter specifies how and where user authentication is to take place when handling incoming connections at the server; it is used to override the current authentication type. If a value is not specified, DB2 uses the value of the *authentication* database manager configuration parameter.

For a description of each authentication type, see "authentication - Authentication type" on page 783 .

Related reference:

- "GET DATABASE MANAGER CONFIGURATION" on page 281
- "RESET DATABASE MANAGER CONFIGURATION Command" in the *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION" on page 384

srvcon_gssplugin_list - List of GSS API plug-ins for incoming connections at the server

Configuration Type	Database manager
Applies to	• Database server with local and remote clients • Database server with local clients • Partitioned database server with local and remote clients
Parameter Type	Configurable
Default [Range]	Null [any valid string]

This parameter specifies the GSS API plug-in libraries that are supported by the database server. By default, the value is null. If the authentication type is GSSPLUGIN and this parameter is NULL, an error is returned. If the authentication type is KERBEROS and this parameter is NULL, the DB2-supplied kerberos module or library is used. This parameter is not used if another authentication type is used.

When the authentication type is KERBEROS and the value of this parameter is not NULL, the list must contain exactly one Kerberos plug-in, and that plug-in is used

for authentication (all other GSS plug-ins in the list are ignored). If there is more than one Kerberos plug-in, an error is returned.

Each GSS API plug-in name must be separated by a comma (,) with no space either before or after the comma. Plug-in names should be listed in the order of preference. This parameter handles incoming connections at the server when the *srvcon_auth* parameter is specified as KERBEROS, KRB_SERVER_ENCRYPT, GSSPLUGIN or GSS_SERVER_ENCRYPT, or when *srvcon_auth* is not specified, and authentication is specified as KERBEROS, KRB_SERVER_ENCRYPT, GSSPLUGIN or GSS_SERVER_ENCRYPT.

Related reference:

- "GET DATABASE MANAGER CONFIGURATION" on page 281
- "RESET DATABASE MANAGER CONFIGURATION Command" in the *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION" on page 384

srvcon_pw_plugin - Userid-password plug-in for incoming connections at the server

Configuration Type Database manager

Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type Configurable

Default [Range] Null [any valid string]

This parameter specifies the name of the default userid-password plug-in library to be used for server-side authentication. By default, the value is null and the DB2-supplied userid-password plug-in library is used.

The parameter handles incoming connections at the server when the *srvcon_auth* parameter is specified as SERVER or SERVER_ENCRYPT, or when *srvcon_auth* is not specified, and *authentication* is specified as CLIENT, SERVER, or SERVER_ENCRYPT.

Related reference:

- "GET DATABASE MANAGER CONFIGURATION" on page 281
- "RESET DATABASE MANAGER CONFIGURATION Command" in the *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION" on page 384

srv_plugin_mode - Server plug-in mode

Configuration Type Database manager

Applies to

- Database server with local and remote clients
- Database server with local clients

	• Partitioned database server with local and remote clients
Parameter Type	Configurable
Default [Range]	UNFENCED

This parameter specifies whether plug-ins are to run in fenced mode or unfenced mode. Unfenced mode is the only supported mode.

Related reference:

- "GET DATABASE MANAGER CONFIGURATION" on page 281
- "RESET DATABASE MANAGER CONFIGURATION Command" in the *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION" on page 384

Part 4. Appendixes

© Copyright IBM Corp. 1993-2004

Index

Special characters

A

© Copyright IBM Corp. 1993-2004

B

C

D

E

F

G

N

O

P

Q

R

S

T

U

V

W

X

Z

Notices

IBM may not offer the products, services, or features discussed in this document in all 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 give 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 (DBCS) information, contact the IBM Intellectual Property Department in your country/region or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, 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.

© Copyright IBM Corp. 1993-2004

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 may contain 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.

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

The following terms are trademarks of International Business Machines Corporation in the United States, other countries, or both, and have been used in at least one of the documents in the DB2 UDB documentation library.

ACF/VTAM
AISPO
AIX
AIXwindows
AnyNet
APPN
AS/400
BookManager
C Set++
C/370
CICS
Database 2
DataHub
DataJoiner
DataPropagator
DataRefresher
DB2
DB2 Connect
DB2 Extenders
DB2 OLAP Server
DB2 Information Integrator
DB2 Query Patroller
DB2 Universal Database
Distributed Relational Database Architecture
DRDA
eServer
Extended Services
FFST
First Failure Support Technology
IBM
IMS
IMS/ESA
iSeries
LAN Distance
MVS
MVS/ESA
MVS/XA
Net.Data
NetView
OS/390
OS/400
PowerPC
pSeries
QBIC
QMF
RACF
RISC System/6000
RS/6000
S/370
SP
SQL/400
SQL/DS
System/370
System/390
SystemView
Tivoli
VisualAge
VM/ESA
VSE/ESA
VTAM
WebExplorer
WebSphere
WIN-OS/2
z/OS
zSeries

The following terms are trademarks or registered trademarks of other companies and have been used in at least one of the documents in the DB2 UDB documentation library:

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Intel and Pentium are trademarks of Intel Corporation in the United States, other countries, or both.

Java and all Java-based trademarks 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.

Other company, product, or service names may be trademarks or service marks of others.

Contacting IBM

In the United States, call one of the following numbers to contact IBM:

- 1-800-IBM-SERV (1-800-426-7378) for customer service
- 1-888-426-4343 to learn about available service options
- 1-800-IBM-4YOU (426-4968) for DB2 marketing and sales

In Canada, call one of the following numbers to contact IBM:

- 1-800-IBM-SERV (1-800-426-7378) for customer service
- 1-800-465-9600 to learn about available service options
- 1-800-IBM-4YOU (1-800-426-4968) for DB2 marketing and sales

To locate an IBM office in your country or region, check IBM's Directory of Worldwide Contacts on the web at http://www.ibm.com/planetwide

Product information

Information regarding DB2 Universal Database products is available by telephone or by the World Wide Web at http://www.ibm.com/software/data/db2/udb

This site contains the latest information on the technical library, ordering books, product downloads, newsgroups, FixPaks, news, and links to web resources.

If you live in the U.S.A., then you can call one of the following numbers:

- 1-800-IBM-CALL (1-800-426-2255) to order products or to obtain general information.
- 1-800-879-2755 to order publications.

For information on how to contact IBM outside of the United States, go to the IBM Worldwide page at www.ibm.com/planetwide

© Copyright IBM Corp. 1993-2004

Printed in USA

SC09-7981-00