Chapter 6. SQL Statements

This chapter contains syntax diagrams, semantic descriptions, rules, and examples of the use of the SQL statements.

Table 18. SQL Statements

SQL Statement	Function	Page
ALTER BUFFERPOOL	Changes the definition of a buffer pool.	464
ALTER NICKNAME	Changes the definition of a nickname.	466
ALTER NODEGROUP	Changes the definition of a nodegroup.	469
ALTER SERVER	Changes the definition of a server.	473
ALTER TABLE	Changes the definition of a table.	477
ALTER TABLESPACE	Changes the definition of a table space.	503
ALTER TYPE (Structured)	Changes the definition of a structured type.	509
ALTER USER MAPPING	Changes the definition of a user authorization mapping.	516
ALTER VIEW	Changes the definition of a view by altering a reference type column to add a scope.	518
BEGIN DECLARE SECTION	Marks the beginning of a host variable declaration section.	520
CALL	Calls a stored procedure.	522
CLOSE	Closes a cursor.	530
COMMENT ON	Replaces or adds a comment to the description of an object.	532
COMMIT	Terminates a unit of work and commits the database changes made by that unit of work.	543
Compound SQL (Embedded)	Combines one or more other SQL statements into an executable block.	545
CONNECT (Type 1)	Connects to an application server according to the rules for remote unit of work.	550
CONNECT (Type 2)	Connects to an application server according to the rules for application-directed distributed unit of work.	558
CREATE ALIAS	Defines an alias for a table, view, or another alias.	566
CREATE BUFFERPOOL	Creates a new buffer pool.	569
CREATE DISTINCT TYPE	Defines a distinct data type.	572
CREATE EVENT MONITOR	Specifies events in the database to monitor.	579
CREATE FUNCTION	Registers a user-defined function.	589

Table 18. SQL Statements (continued)

SQL Statement	Function	Page
CREATE FUNCTION (External Scalar)	Registers a user-defined external scalar function.	590
CREATE FUNCTION (External Table)	Registers a user-defined external table function.	615
CREATE FUNCTION (OLE DB External Table)	Registers a user-defined OLE DB external table function.	631
CREATE FUNCTION (Source or Template)	Registers a user-defined sourced function.	639
CREATE FUNCTION (SQL Scalar, Table or Row)	Registers and defines a user-defined SQL function.	649
CREATE FUNCTION MAPPING	Defines a function mapping.	657
CREATE INDEX	Defines an index on a table.	662
CREATE INDEX EXTENSION	Defines an extension object for use with indexes on tables with structured or distinct type columns.	669
CREATE METHOD	Associates a method body with a previously defined method specification.	676
CREATE NICKNAME	Defines a nickname.	681
CREATE NODEGROUP	Defines a nodegroup.	684
CREATE PROCEDURE	Registers a stored procedure.	687
CREATE SCHEMA	Defines a schema.	704
CREATE SERVER	Defines a data source to a federated database.	708
CREATE TABLE	Defines a table.	712
CREATE TABLESPACE	Defines a table space.	764
CREATE TRANSFORM	Defines transformation functions.	774
CREATE TRIGGER	Defines a trigger.	780
CREATE TYPE (Structured)	Defines a structured data type.	792
CREATE TYPE MAPPING	Defines a mapping between data types.	816
CREATE USER MAPPING	Defines a mapping between user authorizations.	821
CREATE VIEW	Defines a view of one or more table, view or nickname.	823
CREATE WRAPPER	Registers a wrapper.	839
DECLARE CURSOR	Defines an SQL cursor.	841
DECLARE GLOBAL TEMPORARY TABLE	Defines the Global Temporary Table.	846
DELETE	Deletes one or more rows from a table.	855

Table 18. SQL Statements (continued)

SQL Statement	Function	Page
DESCRIBE	Describes the result columns of a prepared SELECT statement.	860
DISCONNECT	Terminates one or more connections when there is no active unit of work.	865
DROP	Deletes objects in the database.	868
END DECLARE SECTION	Marks the end of a host variable declaration section.	894
EXECUTE	Executes a prepared SQL statement.	895
EXECUTE IMMEDIATE	Prepares and executes an SQL statement.	900
EXPLAIN	Captures information about the chosen access plan.	903
FETCH	Assigns values of a row to host variables.	908
FLUSH EVENT MONITOR	Writes out the active internal buffer of an event monitor.	911
FREE LOCATOR	Removes the association between a locator variable and its value.	912
GRANT (Database Authorities)	Grants authorities on the entire database.	913
GRANT (Index Privileges)	Grants the CONTROL privilege on indexes in the database.	916
GRANT (Package Privileges)	Grants privileges on packages in the database.	918
GRANT (Schema Privileges)	Grants privileges on a schema.	921
GRANT (Server Privileges)	Grants privileges to query a specific data source.	924
GRANT (Table, View, or Nickname Privileges)	Grants privileges on tables, views and nicknames.	926
GRANT (Table Space Privileges)	Grants privileges on a tablespace.	934
INCLUDE	Inserts code or declarations into a source program.	936
INSERT	Inserts one or more rows into a table.	938
LOCK TABLE	Either prevents concurrent processes from changing a table or prevents concurrent processes from using a table.	947
OPEN	Prepares a cursor that will be used to retrieve values when the FETCH statement is issued.	949
PREPARE	Prepares an SQL statement (with optional parameters) for execution.	954
REFRESH TABLE	Refreshes the data in a summary table.	964
RELEASE (Connection)	Places one or more connections in the release-pending state.	965
RELEASE SAVEPOINT	Releases a savepoint within a transaction.	967
RENAME TABLE	Renames an existing table.	968
RENAME TABLESPACE	Renames an existing tablespace.	970

Table 18. SQL Statements (continued)

SQL Statement	Function	Page
REVOKE (Database Authorities)	Revokes authorities from the entire database.	972
REVOKE (Index Privileges)	Revokes the CONTROL privilege on given indexes.	975
REVOKE (Package Privileges)	Revokes privileges from given packages in the database.	977
REVOKE (Schema Privileges)	Revokes privileges on a schema.	980
REVOKE (Server Privileges)	Revokes privileges to query a specific data source.	982
REVOKE (Table, View, or Nickname Privileges)	Revokes privileges from given tables, views or nicknames.	984
REVOKE (Table Space Privileges)	Revokes the USE privilege on a given table space.	990
ROLLBACK	Terminates a unit of work and backs out the database changes made by that unit of work.	992
SAVEPOINT	Sets a savepoint within a transaction.	995
SELECT INTO	Specifies a result table of no more than one row and assigns the values to host variables.	998
SET CONNECTION	Changes the state of a connection from dormant to current, making the specified location the current server.	1000
SET CURRENT DEFAULT TRANSFORM GROUP	Changes the value of the CURRENT DEFAULT TRANSFORM GROUP special register.	1002
SET CURRENT DEGREE	Changes the value of the CURRENT DEGREE special register.	1004
SET CURRENT EXPLAIN MODE	Changes the value of the CURRENT EXPLAIN MODE special register.	1006
SET CURRENT EXPLAIN SNAPSHOT	Changes the value of the CURRENT EXPLAIN SNAPSHOT special register.	1008
SET CURRENT PACKAGESET	Sets the schema name for package selection.	1010
SET CURRENT QUERY OPTIMIZATION	Changes the value of the CURRENT QUERY OPTIMIZATION special register.	1012
SET CURRENT REFRESH AGE	Changes the value of the CURRENT REFRESH AGE special register.	1015
SET EVENT MONITOR STATE	Activates or deactivates an event monitor.	1017
SET INTEGRITY	Sets the check pending state and checks data for constraint violations.	1019
SET PASSTHRU	Opens a session for submitting a data source's native SQL directly to the data source.	1029
SET PATH	Changes the value of the CURRENT PATH special register.	1031
SET SCHEMA	Changes the value of the CURRENT SCHEMA special register.	1033

Table 18. SQL Statements (continued)

SQL Statement	Function	Page
SET SERVER OPTION	Sets server option settings.	1035
SET transition-variable	Assigns values to NEW transition variables.	1037
SIGNAL SQLSTATE	Signals an error.	1041
UPDATE	Updates the values of one or more columns in one or more rows of a table.	1043
VALUES INTO	Specifies a result table of no more than one row and assigns the values to host variables.	1054
WHENEVER	Defines actions to be taken on the basis of SQL return codes.	1056

How SQL Statements Are Invoked

The SQL statements described in this chapter are classified as *executable* or *nonexecutable*. The *Invocation* section in the description of each statement indicates whether or not the statement is executable.

An *executable statement* can be invoked in four ways:

- Embedded in an application program
- Embedded in an SQL procedure.
- Dynamically prepared and executed
- Issued interactively

Note: Statements embedded in REXX are prepared and executed dynamically.

Depending on the statement, some or all of these methods can be used. The *Invocation* section in the description of each statement tells which methods can be used.

A *nonexecutable statement* can only be embedded in an application program.

In addition to the statements described in this chapter, there is one more SQL statement construct: the *select-statement*. (See "select-statement" on page 439.) It is not included in this chapter because it is used differently from other statements.

A *select-statement* can be invoked in three ways:

- Included in DECLARE CURSOR and implicitly executed by OPEN, FETCH and CLOSE
- Dynamically prepared, referenced in DECLARE CURSOR, and implicitly executed by OPEN, FETCH and CLOSE

- Issued interactively.

The first two methods are called, respectively, the *static* and the *dynamic* invocation of *select-statement*.

The different methods of invoking an SQL statement are discussed below in more detail. For each method, the discussion includes the mechanism of execution, interaction with host variables, and testing whether or not the execution was successful.

Embedding a Statement in an Application Program

SQL statements can be included in a source program that will be submitted to the precompiler. Such statements are said to be *embedded* in the program. An embedded statement can be placed anywhere in the program where a host language statement is allowed. Each embedded statement must be preceded by the keywords EXEC and SQL.

Executable statements

An executable statement embedded in an application program is executed every time a statement of the host language would be executed if specified in the same place. Thus, a statement within a loop is executed every time the loop is executed, and a statement within a conditional construct is executed only when the condition is satisfied.

An embedded statement can contain references to host variables. A host variable referenced in this way can be used in two ways:

- As input (the current value of the host variable is used in the execution of the statement)
- As output (the variable is assigned a new value as a result of executing the statement).

In particular, all references to host variables in expressions and predicates are effectively replaced by current values of the variables; that is, the variables are used as input. The treatment of other references is described individually for each statement.

All executable statements should be followed by a test of an SQL return code. Alternatively, the WHENEVER statement (which is itself nonexecutable) can be used to change the flow of control immediately after the execution of an embedded statement.

All objects referenced in DML statements must exist when the statements are bound to a DB2 Universal Database.

Nonexecutable statements

An embedded nonexecutable statement is processed only by the precompiler. The precompiler reports any errors encountered in the statement. The statement is *never* processed during program execution. Therefore, such statements should not be followed by a test of an SQL return code.

Embedding a Statement in an SQL Procedure

Statements can be included in the SQL-procedure-body portion of the CREATE PROCEDURE statement. Such statements are said to be *embedded* in the SQL procedure. Statements that can be embedded in an SQL procedure are specified in "SQL Procedure Statement" on page 1060. Unlike statements embedded in an application, there is no need for any keywords preceding the SQL statement. Whenever an SQL statement description refers to a *host-variable*, an *SQL-variable* can be used when the statement is embedded in an SQL procedure.

Dynamic Preparation and Execution

An application program can dynamically build an SQL statement in the form of a character string placed in a host variable. In general, the statement is built from some data available to the program (for example, input from a workstation). The statement (other than a select-statement) so constructed can be prepared for execution by means of the (embedded) statement PREPARE and executed by means of the (embedded) statement EXECUTE. Alternatively, the (embedded) statement EXECUTE IMMEDIATE can be used to prepare and execute a statement in one step.

A statement that is going to be dynamically prepared must not contain references to host variables. It can instead contain parameter markers. (See "PREPARE" on page 954 for rules concerning the parameter markers.) When the prepared statement is executed, the parameter markers are effectively replaced by current values of the host variables specified in the EXECUTE statement. (See "EXECUTE" on page 895 for rules concerning this replacement.) Once prepared, a statement can be executed several times with different values of host variables. Parameter markers are not allowed in EXECUTE IMMEDIATE.

The successful or unsuccessful execution of the statement is indicated by the setting of an SQL return code in the SQLCA after the EXECUTE (or EXECUTE IMMEDIATE) statement. The SQL return code should be checked as described above. See "SQL Return Codes" on page 461 for more information.

Static Invocation of a select-statement

A *select-statement* can be included as a part of the (nonexecutable) statement DECLARE CURSOR. Such a statement is executed every time the cursor is opened by means of the (embedded) statement OPEN. After the cursor is open, the result table can be retrieved one row at a time by successive executions of the FETCH statement.

Used in this way, the *select-statement* can contain references to host variables. These references are effectively replaced by the values that the variables have at the moment of executing OPEN.

Dynamic Invocation of a select-statement

An application program can dynamically build a *select-statement* in the form of a character string placed in a host variable. In general, the statement is built from some data available to the program (for example, a query obtained from a workstation). The statement so constructed can be prepared for execution by means of the (embedded) statement PREPARE, and referenced by a (nonexecutable) statement DECLARE CURSOR. The statement is then executed every time the cursor is opened by means of the (embedded) statement OPEN. After the cursor is open, the result table can be retrieved one row at a time by successive executions of the FETCH statement.

Used in this way, the *select-statement* must not contain references to host variables. It can contain parameter markers instead. (See "PREPARE" on page 954 for rules concerning the parameter markers.) The parameter markers are effectively replaced by the values of the host variables specified in the OPEN statement. (See "OPEN" on page 949 for rules concerning this replacement.)

Interactive Invocation

A capability for entering SQL statements from a workstation is part of the architecture of the database manager. A statement entered in this way is said to be issued interactively.

A statement issued interactively must be an executable statement that does not contain parameter markers or references to host variables, because these make sense only in the context of an application program.

SQL Return Codes

An application program containing executable SQL statements can use either the SQLCODE or SQLSTATE values to handle return codes from SQL statements. There are two ways in which an application can get access to these values.

- Include a structure named SQLCA. An SQLCA is provided automatically in REXX. In other languages, an SQLCA can be obtained by using the INCLUDE SQLCA statement.

  The SQLCA includes an integer variable named SQLCODE and a character string variable named SQLSTATE.
- When LANGLEVEL SQL92E is specified as a precompile option, a variable SQLCODE or SQLSTATE may be declared in the SQL declare section of the program. If neither of these variables is declared in the SQL declare section, it is assumed that a variable SQLCODE is declared elsewhere in the program. When using LANGLEVEL SQL92E, the program should not have an INCLUDE SQLCA statement.

Occasionally, warning conditions are mentioned in addition to error conditions with respect to return codes. A warning SQLCODE is a positive value and a warning SQLSTATE has the first two characters set to '01'.

SQLCODE

An SQLCODE is set by the database manager after each SQL statement is executed. All database managers conform to the ISO/ANSI SQL standard, as follows:

- If SQLCODE = 0 and SQLWARN0 is blank, execution was successful.
- If SQLCODE = 100, "no data" was found. For example, a FETCH statement returned no data, because the cursor was positioned after the last row of the result table.
- If SQLCODE > 0 and not = 100, execution was successful with a warning
- If SQLCODE = 0 and SQLWARN0 = 'W', execution was successful, however, one or more warning indicators were set. Refer to "Appendix B. SQL Communications (SQLCA)" on page 1107 for more details.
- If SQLCODE < 0, execution was not successful.

The meaning of SQLCODE values other than 0 and 100 is product-specific. See the *Message Reference* for the product-specific meanings.

SQLSTATE

SQLSTATE is also set by the database manager after execution of each SQL statement. Thus, application programs can check the execution of SQL statements by testing SQLSTATE instead of SQLCODE.

SQLSTATE provides application programs with common codes for common error conditions. Furthermore, SQLSTATE is designed so that application programs can test for specific errors or classes of errors. The coding scheme is the same for all IBM database managers and is based on the ISO/ANSI SQL92 standard. For a complete list of the possible values of SQLSTATE, see the *Message Reference*.

SQL Comments

Static SQL statements can include host language or SQL comments. SQL comments are introduced by two hyphens.

These rules apply to the use of SQL comments:

- The two hyphens must be on the same line, not separated by a space.
- Comments can be started wherever a space is valid (except within a delimiter token or between 'EXEC' and 'SQL').
- Comments are terminated by the end of the line.
- Comments are not allowed within statements that are dynamically prepared (using PREPARE or EXECUTE IMMEDIATE).
- In COBOL, the hyphens must be preceded by a space.

Example: This example shows how to include comments in an SQL statement within a C program:

```
EXEC SQL
  CREATE VIEW PRJ_MAXPER         -- projects with most support personnel
    AS SELECT PROJNO, PROJNAME   -- number and name of project
         FROM PROJECT
         WHEREDEPTNO = 'E21'    -- systems support dept code
         AND PRSTAFF > 1;
```

ALTER BUFFERPOOL

The ALTER BUFFERPOOL statement is used to do the following:

- modify the size of the buffer pool on all partitions (or nodes) or on a single partition
- turn on or off the use of extended storage
- add this buffer pool definition to a new nodegroup.

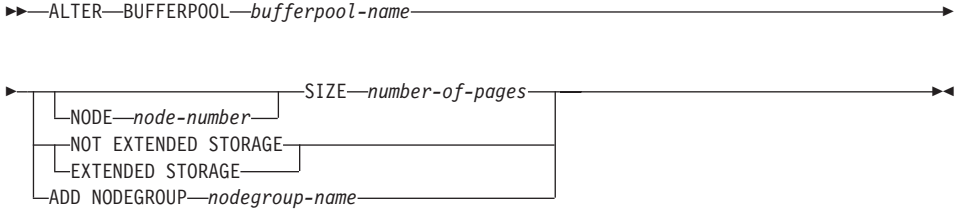
Invocation

This statement can be embedded in an application program or issued interactively. 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 authorization ID of the statement must have SYSCTRL or SYSADM authority.

Syntax



Description

bufferpool-name
Names the buffer pool. This is a one-part name. It is an SQL identifier (either ordinary or delimited). It must be a buffer pool described in the catalog.

NODE *node-number*
Specifies the partition on which size of the buffer pool is modified. The partition must be in one of the nodegroups for the buffer pool (SQLSTATE 42729). If this clause is not specified, then the size of the buffer pool is modified on all partitions on which the buffer pool exists that used the default size for the buffer pool (did not have a size specified in the *except-on-nodes-clause* of the CREATE buffer pool statement).

SIZE *number-of-pages*
The size of the buffer pool specified as the number of pages. [56]

EXTENDED STORAGE
If the extended storage configuration is turned on [57], pages that are being migrated out of this buffer pool, will be cached in the extended storage.

NOT EXTENDED STORAGE
Even if the extended storage configuration is turned on, pages that are being migrated out of this buffer pool, will NOT be cached in the extended storage.

ADD NODEGROUP *nodegroup-name*
Adds this nodegroup to the list of nodegroups to which the buffer pool definition is applicable. For any partition in the nodegroup that does not already have the bufferpool defined, the bufferpool is created on the partition using the default size specified for the bufferpool. Table spaces in *nodegroup-name* may specify this buffer pool. The nodegroup must currently exist in the database (SQLSTATE 42704).

Notes

- Although the buffer pool definition is transactional and the changes to the buffer pool definition will be reflected in the catalog tables on commit, no changes to the actual buffer pool will take effect until the next time the database is started. The current attributes of the buffer pool will exist until then, and there will not be any impact to the buffer pool in the interim. Tables created in table spaces of new nodegroups will use the default buffer pool.
- There should be enough real memory on the machine for the total of all the buffer pools, as well as for the rest of the database manager and application requirements.

56. The size can be specified with a value of (-1) which will indicate that the buffer pool size should be taken from the BUFFPAGE database configuration parameter.

57. Extended storage configuration is turned on by setting the database configuration parameters NUM_ESTORE_SEGS and ESTORE_SEG_SIZE to non-zero values. See *Administration Guide* for details.

ALTER NICKNAME

The ALTER NICKNAME statement modifies the federated database's representation of a data source table or view by:

- Changing the local names of the table's or view's columns
- Changing the local data types of these columns
- Adding, changing, or deleting options for these columns

Invocation

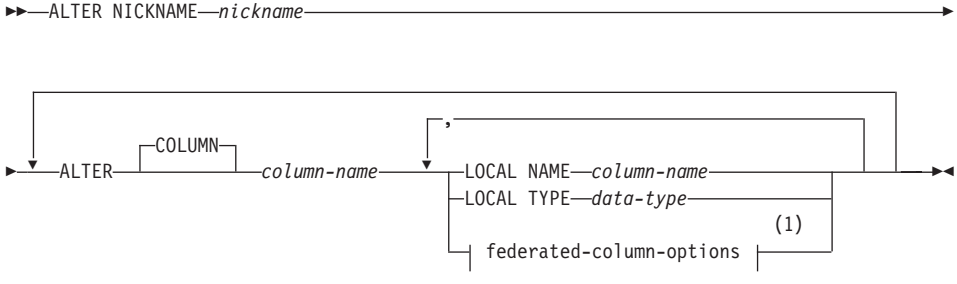
This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared. However, if the bind option DYNAMICRULES BIND applies, the statement cannot be dynamically prepared (SQLSTATE 42509).

Authorization

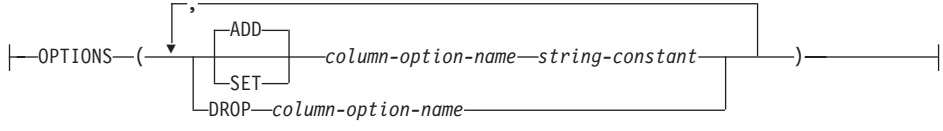
The privileges held by the authorization ID of the statement must include at least one of the following:

- SYSADM or DBADM authority
- ALTER privilege on the nickname specified in the statement
- CONTROL privilege on the nickname specified in the statement
- ALTERIN privilege on the schema, if the schema name of the nickname exists
- Definer of the nickname as recorded in the DEFINER column of the catalog view for the nickname

Syntax



federated-column-options:



Notes:

1 If the user needs to specify the federated-column-options clause in addition to the LOCAL NAME parameter, the LOCAL TYPE parameter, or both, the user must specify the federated-column-options clause last.

Description

nickname
Identifies the nickname for the data source table or view that contains the column specified after the COLUMN keyword. It must be a nickname described in the catalog.

ALTER COLUMN *column-name*
Names the column to be altered. The *column-name* is the federated server's current name for the column of the table or view at the data source. The *column-name* must identify an existing column of the data source table or view referenced by *nickname*.

LOCAL NAME *column-name*
Is the new name by which the federated server is to reference the column identified by the ALTER COLUMN *column-name* parameter. This new name must be a valid DB2 identifier.

LOCAL TYPE *data-type*
Maps the specified column's data type to a local data type other than the one that it maps to now. The new type is denoted by *data-type*.

The *data-type* cannot be LONG VARCHAR, LONG VARGRAPHIC, DATALINK, a large object (LOB) data type, or a user-defined type.

OPTIONS
Indicates what column options are to be enabled, reset, or dropped for the column specified after the COLUMN keyword. Refer to "Column Options" on page 1247 for descriptions of *column-option-name*s and their settings.

ADD
Enables a column option.

SET
Changes the setting of a column option.

column-option-name
Names a column option that is to be enabled or reset.

string-constant
Specifies the setting for *column-option-name* as a character string constant.

DROP *column-option-name*
Drops a column option.

Rules

- If a view has been created on a nickname, the ALTER NICKNAME statement cannot be used to change the local names or data types for the columns in the table or view that the nickname references (SQLSTATE 42601). The statement can be used, however, to enable, reset, or drop column options for these columns.

Notes

- If ALTER NICKNAME is used to change the local name for a column in a table or view that a nickname references, queries of the column must reference it by its new name.
- A column option cannot be specified more than once in the same ALTER NICKNAME statement (SQLSTATE 42853). When a column option is enabled, reset, or dropped, any other column options that are in use are not affected.
- When the local specification of a column's data type is changed, the database manager invalidates any statistics (HIGH2KEY, LOW2KEY, and so on) gathered for that column.
- The ALTER NICKNAME statement cannot be processed within a unit of work (UOW) if the nickname referenced in this statement is already referenced by a SELECT statement in the same UOW (SQLSTATE 55007).

Examples

Example 1: The nickname NICK1 references a DB2 Universal Database for AS/400 table called T1. Also, COL1 is the local name that references this table's first column, C1. Change the local name for C1 to NEWCOL.

```
ALTER NICKNAME NICK1
   ALTER COLUMN COL1
   LOCAL NAME NEWCOL
```

Example 2: The nickname EMPLOYEE references a DB2 Universal Database for OS/390 table called EMP. Also, SALARY is the local name that references EMP_SAL, one of this table's columns. The column's data type, FLOAT, maps to the local data type, DOUBLE. Change the mapping so that FLOAT maps to DECIMAL (10, 5).

```
ALTER NICKNAME EMPLOYEE
   ALTER COLUMN SALARY
   LOCAL TYPE DECIMAL(10,5)
```

Example 3: Indicate that in an Oracle table, a column with the data type of VARCHAR doesn't have trailing blanks. The nickname for the table is NICK2, and the local name for the column is COL1.

```
ALTER NICKNAME NICK2
   ALTER COLUMN COL1
   OPTIONS ( ADD VARCHAR_NO_TRAILING_BLANKS 'Y' )
```

ALTER NODEGROUP

The ALTER NODEGROUP statement is used to:

- add one or more partitions or nodes to a nodegroup
- drop one or more partitions from a nodegroup.

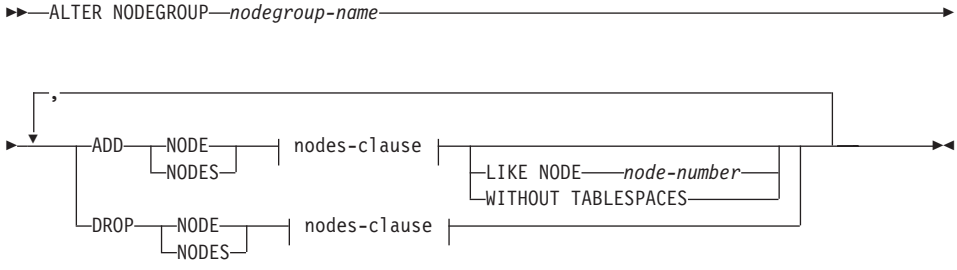
Invocation

This statement can be embedded in an application program or issued interactively. 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).

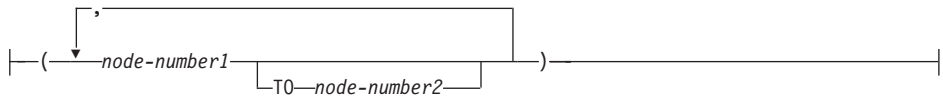
Authorization

The authorization ID of the statement must have SYSCTRL or SYSADM authority.

Syntax



nodes-clause:



Description

nodegroup-name
Names the nodegroup. This is a one-part name. It is an SQL identifier (either ordinary or delimited). It must be a nodegroup described in the catalog. IBMCATGROUP and IBMTEMPGROUP cannot be specified (SQLSTATE 42832).

ADD NODE
Specifies the specific partition or partitions to add to the nodegroup. NODES is a synonym for NODE. Any specified partition must not already be defined in the nodegroup (SQLSTATE 42728).

DROP NODE
Specifies the specific partition or partitions to drop from the nodegroup. NODES is a synonym for NODE. Any specified partition must already be defined in the nodegroup (SQLSTATE 42729).

nodes-clause
Specifies the partition or partitions to be added or dropped.

node-number1
Specify a specific partition number.

TO *node-number2*
Specify a range of partition numbers. The value of *node-number2* must be greater than or equal to the value of *node-number1* (SQLSTATE 428A9).

LIKE NODE *node-number*
Specifies that the containers for the existing table spaces in the nodegroup will be the same as the containers on the specified *node-number*. The partition specified must be a partition that existed in the nodegroup prior to this statement and is not included in a DROP NODE clause of the same statement.

WITHOUT TABLESPACES
Specifies that the default table spaces are not created on the newly added partition or partitions. The ALTER TABLESPACE using the FOR NODE clause must be used to define containers for use with the table spaces that are defined on this nodegroup. If this option is not specified, the default containers are specified on newly added partitions for each table space defined on the nodegroup.

Rules

- Each partition or node specified by number must be defined in the `db2nodes.cfg` file (SQLSTATE 42729). See "Data Partitioning Across Multiple Partitions" on page 59 for information about this file.
- Each *node-number* listed in the ON NODES clause must be for a unique partition (SQLSTATE 42728).
- A valid partition number is between 0 and 999 inclusive (SQLSTATE 42729).
- A partition cannot appear in both the ADD and DROP clauses (SQLSTATE 42728).
- There must be at least one partition remaining in the nodegroup. The last partition cannot be dropped from a nodegroup (SQLSTATE 428C0).
- If neither the LIKE NODE clause nor the WITHOUT TABLESPACES clause is specified when adding a partition, the default is to use the lowest partition number of the existing partitions in the nodegroup (say it is *2*) and proceed as if LIKE NODE *2* had been specified. For an existing partition to be used as the default it must have containers defined for all

the table spaces in the nodegroup (column IN_USE of SYSCAT.NODEGROUPDEF is not 'T').

Notes

- When a partition or node is added to a nodegroup, a catalog entry is made for the partition (see SYSCAT.NODEGROUPDEF). The partitioning map is changed immediately to include the new partition along with an indicator (IN_USE) that the partition is in the partitioning map if either:
  - no table spaces are defined in the nodegroup or
  - no tables are defined in the table spaces defined in the nodegroup and the WITHOUT TABLESPACES clause was not specified.

  The partitioning map is not changed and the indicator (IN_USE) is set to indicate that the partition is not included in the partitioning map if either:
  - tables exist in table spaces in the nodegroup or
  - table spaces exist in the nodegroup and the WITHOUT TABLESPACES clause was specified.

  To change the partitioning map, the REDISTRIBUTE NODEGROUP command must be used. This redistributes any data, changes the partitioning map, and changes the indicator. Table space containers need to be added before attempting to redistribute data if the WITHOUT TABLESPACES clause was specified.
- When a partition is dropped from a nodegroup, the catalog entry for the partition (see SYSCAT.NODEGROUPDEF) is updated. If there are no tables defined in the table spaces defined in the nodegroup, the partitioning map is changed immediately to exclude the dropped partition and the entry for the partition in the nodegroup is dropped. If tables exist, the partitioning map is not changed and the indicator (IN_USE) is set to indicate that the partition is waiting to be dropped. The REDISTRIBUTE NODEGROUP command must be used to redistribute the data and drop the entry for the partition from the nodegroup.

Example

Assume that you have a six-partition database that has the following partitions: 0, 1, 2, 5, 7, and 8. Two partitions are added to the system with partition numbers 3 and 6.

- Assume that you want to add both partitions or nodes 3 and 6 to a nodegroup called MAXGROUP and have the table space containers like those on partition 2. The statement is as follows:

```
ALTER  NODEGROUP MAXGROUP
ADD NODES (3,6) LIKE NODE 2
```

- Assume that you want to drop partition 1 and add partition 6 to nodegroup MEDGROUP. You will define the table space containers separately for partition 6 using ALTER TABLESPACE. The statement is as follows:

```
ALTER NODEGROUP MEDGROUP
 ADD NODE(6) WITHOUT TABLESPACES
 DROP NODE(1)
```

ALTER SERVER

The ALTER SERVER statement[58] is used to:

- Modify the definition of a specific data source, or the definition of a category of data sources
- Make changes in the configuration of a specific data source, or the configuration of a category of data sources—changes that will persist over multiple connections to the federated database.

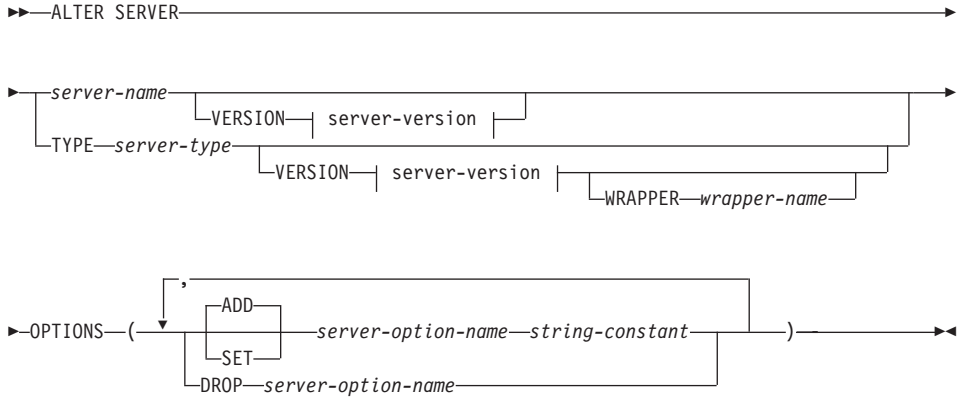
Invocation

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. However, if the bind option DYNAMICRULES BIND applies, the statement cannot be dynamically prepared (SQLSTATE 42509).

Authorization

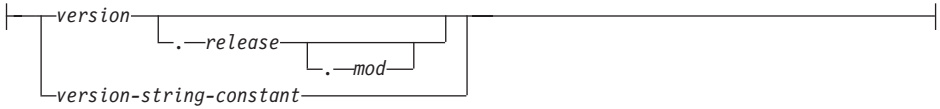
The authorization ID of the statement must include either SYSADM or DBADM authority on the federated database.

Syntax



58. In this statement, the word SERVER and the parameter names that start with *server-* refer only to data sources in a federated system. They do not refer to the federated server in such a system, or to DRDA application servers. For information about federated systems, see "DB2 Federated Systems" on page 41. For information about DRDA application servers, see "Distributed Relational Database" on page 29.

server-version:



Description

server-name
Identifies the federated server's name for the data source to which the changes being requested are to apply. The data source must be one that is described in the catalog.

VERSION
After *server-name*, VERSION and its parameter specify a new version of the data source that *server-name* denotes.

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').

TYPE *server-type*
Specifies the type of data source to which the changes being requested are to apply. The server type must be one that is listed in the catalog.

VERSION
After *server-type*, VERSION and its parameter specify the version of the data sources for which server options are to be enabled, reset, or dropped.

WRAPPER *wrapper-name*
Specifies the name of the wrapper that the federated server uses to interact with data sources of the type and version denoted by *server-type* and *server-version*. The wrapper must be listed in the catalog.

OPTIONS
Indicates what server options are to be enabled, reset, or dropped for the data source denoted by *server-name*, or for the category of data sources

denoted by *server-type* and its associated parameters. Refer to "Server Options" on page 1249 for descriptions of *server-option-name*s and their settings.

ADD
Enables a server option.

SET
Changes the setting of a server option.

server-option-name
Names a server option that is to be enabled or reset.

string-constant
Specifies the setting for *server-option-name* as a character string constant.

DROP *server-option-name*
Drops a server option.

Notes

- This statement does not support the DBNAME and NODE server options (SQLSTATE 428EE).
- A server option cannot be specified more than once in the same ALTER SERVER statement (SQLSTATE 42853). When a server option is enabled, reset, or dropped, any other server options that are in use are not affected.
- An ALTER SERVER statement 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 a SELECT statement that references a nickname for a table or view within this data source (SQLSTATE 55007).
  - 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 55007).
- If the server option is set to one value for a type of data source, and set to another value for an instance of the type, the second value overrides the first one for the instance. For example, assume that PLAN_HINTS is set to 'Y' for server type ORACLE, and to 'N' for an Oracle data source named DELPHI. This configuration causes plan hints to be enabled at all Oracle data sources except DELPHI.

Examples

Example 1: Ensure that when authorization IDs are sent to your Oracle 8.0.3 data sources, the case of the IDs will remain unchanged. Also, assume that these data sources have started to run on an upgraded CPU that's half as fast as your local CPU. Inform the optimizer of this statistic.

```
ALTER SERVER
   TYPE ORACLE
   VERSION 8.0.3
   OPTIONS
      ( ADD FOLD_ID 'N',
      SET CPU_RATIO '2.0' )
```

Example 2: Indicate that a DB2 Universal Database for AS/400 Version 3.0 data source called SUNDIAL has been upgraded to Version 3.1.

```
ALTER SERVER SUNDIAL
   VERSION 3.1
```