LOCK TABLE

The LOCK TABLE statement either prevents concurrent application processes from changing a table or prevents concurrent application processes from using a table.

Invocation

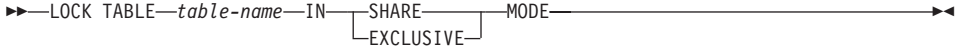
This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared.

Authorization

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

- SELECT privilege on the table
- CONTROL privilege on the table
- SYSADM or DBADM authority.

Syntax



Description

table-name
Identifies the table. The *table-name* must identify a table that exists at the application server, but it must not identify a catalog table. It cannot be a nickname (SQLSTATE 42809) or a declared temporary table (SQLSTATE 42995). If the *table-name* is a typed table, it must be the root table of the table hierarchy (SQLSTATE 428DR).

IN SHARE MODE
Prevents concurrent application processes from executing any but read-only operations on the table.

IN EXCLUSIVE MODE
Prevents concurrent application processes from executing any operations on the table. Note that EXCLUSIVE MODE does not prevent concurrent application processes that are running at isolation level Uncommitted Read (UR) from executing read-only operations on the table.

Notes

- Locking is used to prevent concurrent operations. A lock is not necessarily acquired during the execution of the LOCK TABLE statement if a suitable lock already exists. The lock that prevents concurrent operations is held at least until the termination of the unit of work.

- In a partitioned database, a table lock is first acquired at the first partition in the nodegroup (the partition with the lowest number) and then at other partitions. If the LOCK TABLE statement is interrupted, the table may be locked on some partitions but not on others. If this occurs, either issue another LOCK TABLE statement to complete the locking on all partitions, or issue a COMMIT or ROLLBACK statement to release the current locks.
- This statement affects all partitions in the nodegroup.

Example

Obtain a lock on the table EMP. Do not allow other programs either to read or update the table.

```
LOCK TABLE EMP IN EXCLUSIVE MODE
```

OPEN

The OPEN statement opens a cursor so that it can be used to fetch rows from its result table.

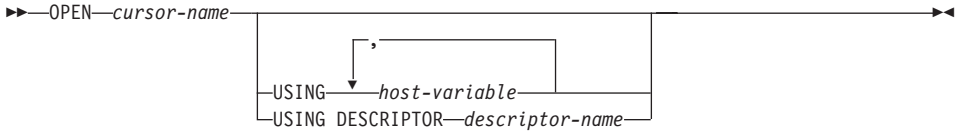
Invocation

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

See "DECLARE CURSOR" on page 841 for the authorization required to use a cursor.

Syntax



Description

cursor-name
Names a cursor that is defined in a DECLARE CURSOR statement that was stated earlier in the program. When the OPEN statement is executed, the cursor must be in the closed state.

The DECLARE CURSOR statement must identify a SELECT statement, in one of the following ways:

- Including the SELECT statement in the DECLARE CURSOR statement
- Including a *statement-name* that names a prepared SELECT statement.

The result table of the cursor is derived by evaluating that SELECT statement, using the current values of any host variables specified in it or in the USING clause of the OPEN statement. The rows of the result table may be derived during the execution of the OPEN statement and a temporary table may be created to hold them; or they may be derived during the execution of subsequent FETCH statements. In either case, the cursor is placed in the open state and positioned before the first row of its result table. If the table is empty the state of the cursor is effectively "after the last row".

USING
Introduces a list of host variables whose values are substituted for the parameter markers (question marks) of a prepared statement. (For an explanation of parameter markers, see "PREPARE" on page 954.) If the

DECLARE CURSOR statement names a prepared statement that includes parameter markers, USING must be used. If the prepared statement does not include parameter markers, USING is ignored.

host-variable
Identifies a variable described in the program in accordance with the rules for declaring host variables. The number of variables must be the same as the number of parameter markers in the prepared statement. The *n*th variable corresponds to the *n*th parameter marker in the prepared statement. Where appropriate, locator variables and file reference variables can be provided as the source of values for parameter markers.

DESCRIPTOR *descriptor-name*
Identifies an SQLDA that must contain a valid description of host variables.

Before the OPEN statement is processed, the user must set the following fields in the SQLDA:

- SQLN to indicate the number of SQLVAR occurrences provided in the SQLDA
- SQLDABC to indicate the number of bytes of storage allocated for the SQLDA
- SQLD to indicate the number of variables used in the SQLDA when processing the statement
- SQLVAR occurrences to indicate the attributes of the variables.

The SQLDA must have enough storage to contain all SQLVAR occurrences. Therefore, the value in SQLDABC must be greater than or equal to 16 + SQLN*(N), where N is the length of an SQLVAR occurrence.

If LOB result columns need to be accommodated, there must be two SQLVAR entries for every select-list item (or column of the result table). See "Effect of DESCRIBE on the SQLDA" on page 1120, which discusses SQLDOUBLED and LOB columns.

SQLD must be set to a value greater than or equal to zero and less than or equal to SQLN. For more information, see "Appendix C. SQL Descriptor Area (SQLDA)" on page 1113.

Rules

- When the SELECT statement of the cursor is evaluated, each parameter marker in the statement is effectively replaced by its corresponding host variable. For a typed parameter marker, the attributes of the target variable are those specified by the CAST specification. For an untyped parameter

marker, the attributes of the target variable are determined according to the context of the parameter marker. See "Rules" on page 955 for the rules affecting parameter markers.

- Let V denote a host variable that corresponds to parameter marker P. The value of V is assigned to the target variable for P in accordance with the rules for assigning a value to a column. Thus:
  - V must be compatible with the target.
  - If V is a string, its length must not be greater than the length attribute of the target.
  - If V is a number, the absolute value of its integral part must not be greater than the maximum absolute value of the integral part of the target.
  - If the attributes of V are not identical to the attributes of the target, the value is converted to conform to the attributes of the target.

  When the SELECT statement of the cursor is evaluated, the value used in place of P is the value of the target variable for P. For example, if V is CHAR(6), and the target is CHAR(8), the value used in place of P is the value of V padded with two blanks.
- The USING clause is intended for a prepared SELECT statement that contains parameter markers. However, it can also be used when the SELECT statement of the cursor is part of the DECLARE CURSOR statement. In this case the OPEN statement is executed as if each host variable in the SELECT statement were a parameter marker, except that the attributes of the target variables are the same as the attributes of the host variables in the SELECT statement. The effect is to override the values of the host variables in the SELECT statement of the cursor with the values of the host variables specified in the USING clause.

Notes

- ***Closed state of cursors:*** All cursors in a program are in the closed state when the program is initiated and when it initiates a ROLLBACK statement.

  All cursors, except open cursors declared WITH HOLD, are in a closed state when a program issues a COMMIT statement.

  A cursor can also be in the closed state because a CLOSE statement was executed or an error was detected that made the position of the cursor unpredictable.
- To retrieve rows from the result table of a cursor, execute a FETCH statement when the cursor is open. The only way to change the state of a cursor from closed to open is to execute an OPEN statement.
- ***Effect of temporary tables:*** In some cases, the result table of a cursor is derived during the execution of FETCH statements. In other cases, the temporary table method is used instead. With this method the entire result

table is transferred to a temporary table during the execution of the OPEN statement. When a temporary table is used, the results of a program can differ in these two ways:

- An error can occur during OPEN that would otherwise not occur until some later FETCH statement.
- INSERT, UPDATE, and DELETE statements executed in the same transaction while the cursor is open cannot affect the result table.

Conversely, if a temporary table is not used, INSERT, UPDATE, and DELETE statements executed while the cursor is open can affect the result table if issued from the same unit of work. The *Application Development Guide* describes how locking can be used to control the effect of INSERT, UPDATE, and DELETE operations executed by concurrent units of work. Your result table can also be affected by operations executed by your own unit of work, and the effect of such operations is not always predictable. For example, if cursor C is positioned on a row of its result table defined as SELECT * FROM T, and a new row is inserted into T, the effect of that insert on the result table is not predictable because its rows are not ordered. Thus a subsequent FETCH C may or may not retrieve the new row of T.

- Statement caching affects cursors declared open by the OPEN statement. See the "Notes" on page 896 for information.

Examples

Example 1: Write the embedded statements in a COBOL program that will:

1. Define a cursor C1 that is to be used to retrieve all rows from the DEPARTMENT table for departments that are administered by (ADMRDEPT) department 'A00'.
2. Place the cursor C1 before the first row to be fetched.

```
EXEC SQL  DECLARE C1 CURSOR FOR
                SELECT DEPTNO, DEPTNAME, MGRNO
                  FROM DEPARTMENT
                  WHERE ADMRDEPT = 'A00'
END-EXEC.


EXEC SQL  OPEN C1
END-EXEC.
```

Example 2: Code an OPEN statement to associate a cursor DYN_CURSOR with a dynamically defined select-statement in a C program. Assuming two parameter markers are used in the predicate of the select-statement, two host variable references are supplied with the OPEN statement to pass integer and varchar(64) values between the application and the database. (The related host variable definitions, PREPARE statement, and DECLARE CURSOR statement are also shown in the example below.)

```
EXEC SQL  BEGIN DECLARE SECTION;
  static short    hv_int;
  char            hv_vchar64[64];
  char            stmt1_str[200];
EXEC SQL  END DECLARE SECTION;

EXEC SQL  PREPARE STMT1_NAME FROM :stmt1_str;
EXEC SQL  DECLARE DYN_CURSOR CURSOR FOR STMT1_NAME;

EXEC SQL  OPEN DYN_CURSOR USING :hv_int, :hv_vchar64;
```

Example 3: Code an OPEN statement as in example 2, but in this case the number and data types of the parameter markers in the WHERE clause are not known.

```
EXEC SQL  BEGIN DECLARE SECTION;
  char    stmt1_str[200];
EXEC SQL  END DECLARE SECTION;
EXEC SQL  INCLUDE SQLDA;

EXEC SQL  PREPARE STMT1_NAME FROM :stmt1_str;
EXEC SQL  DECLARE DYN_CURSOR CURSOR FOR STMT1_NAME;

EXEC SQL  OPEN DYN_CURSOR USING DESCRIPTOR :sqlda;
```

PREPARE

The PREPARE statement is used by application programs to dynamically prepare an SQL statement for execution. The PREPARE statement creates an executable SQL statement, called a *prepared statement*, from a character string form of the statement, called a *statement string*.

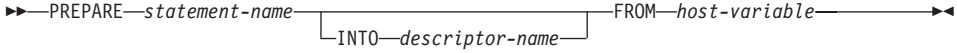
Invocation

This statement can be embedded only in an application program. It is an executable statement that cannot be dynamically prepared.

Authorization

For statements where authorization checking is performed at statement preparation time (DML), the privileges held by the authorization ID of the statement must include those required to execute the SQL statement specified by the PREPARE statement. For statements where authorization checking is performed at statement execution (DDL, GRANT, and REVOKE statements), no authorization is required to use the statement; however, the authorization is checked when the prepared statement is executed.

Syntax



Description

statement-name
Names the prepared statement. If the name identifies an existing prepared statement, that previously prepared statement is destroyed. The name must not identify a prepared statement that is the SELECT statement of an open cursor.

INTO
If INTO is used, and the PREPARE statement is successfully executed, information about the prepared statement is placed in the SQLDA specified by the descriptor-name.

descriptor-name
Is the name of an SQLDA.[103]

FROM
Introduces the statement string. The statement string is the value of the specified host variable.

host-variable
Must identify a host variable that is described in the program in

103. The DESCRIBE statement may be used as an alternative to this clause. See "DESCRIBE" on page 860.

accordance with the rules for declaring character string variables. It must be a character-string variable (either fixed-length or varying-length).

Rules

- ***Rules for statement strings:*** The statement string must be an executable statement that can be dynamically prepared. It must be one of the following SQL statements:
  - ALTER
  - COMMENT ON
  - COMMIT
  - CREATE
  - DECLARE GLOBAL TEMPORARY TABLE
  - DELETE
  - DROP
  - EXPLAIN
  - FLUSH EVENT MONITOR
  - GRANT
  - INSERT
  - LOCK TABLE
  - REFRESH TABLE
  - RELEASE SAVEPOINT
  - RENAME TABLE
  - RENAME TABLESPACE
  - REVOKE
  - ROLLBACK
  - SAVEPOINT
  - select-statement
  - SET CURRENT DEFAULT TRANSFORM GROUP
  - SET CURRENT DEGREE
  - SET CURRENT EXPLAIN MODE
  - SET CURRENT EXPLAIN SNAPSHOT
  - SET CURRENT QUERY OPTIMIZATION
  - SET CURRENT REFRESH AGE
  - SET EVENT MONITOR STATE
  - SET INTEGRITY
  - SET PASSTHRU
  - SET PATH

- SET SCHEMA
- SET SERVER OPTION
- UPDATE

- ***Parameter Markers:*** Although a statement string cannot include references to host variables, it may include *parameter markers*; those can be replaced by the values of host variables when the prepared statement is executed.A parameter marker is a question mark (?) that is declared where a host variable could be stated if the statement string were a static SQL statement. For an explanation of how parameter markers are replaced by values, see "OPEN" on page 949 and "EXECUTE" on page 895.

  There are two types of parameter markers:

  Typed parameter marker
  A parameter marker that is specified along with its target data type. It has the general form:

  ```
  CAST(? AS data-type)
  ```

  This notation is not a function call, but a "promise" that the type of the parameter at run time will be of the data type specified or some data type that can be converted to the specified data type. For example, in:

  ```
  UPDATE EMPLOYEE
    SET LASTNAME = TRANSLATE(CAST(? AS VARCHAR(12)))
  WHERE EMPNO = ?
  ```

  the value of the argument of the TRANSLATE function will be provided at run time. The data type of that value will either be VARCHAR(12), or some type that can be converted to VARCHAR(12).

  Untyped parameter marker
  A parameter marker that is specified without its target data type. It has the form of a single question mark. The data type of an untyped parameter marker is provided by context. For example, the untyped parameter marker in the predicate of the above update statement is the same as the data type of the EMPNO column.

  Typed parameter markers can be used in dynamic SQL statements wherever a host variable is supported and the data type is based on the promise made in the CAST function.

  Untyped parameters markers can be used in dynamic SQL statements in selected locations where host variables are supported. These locations and the resulting data type are found in Table 28 on page 957. The locations are grouped in this table into expressions, predicates and functions to assist in determining applicability of an untyped parameter marker. When an untyped parameter marker is used in a function (including arithmetic operators, CONCAT and datetime operators) with an unqualified function

name, the qualifier is set to 'SYSIBM' for the purposes of function resolution.

Table 28. Untyped Parameter Marker Usage

Untyped Parameter Marker Location	Data Type
Expressions (including select list, CASE and VALUES)	
Alone in a select list	Error
Both operands of a single arithmetic operator, after considering operator precedence and order of operation rules. Includes cases such as: `? + ? + 10`	Error
One operand of a single operator in an arithmetic expression (not a datetime expression) Includes cases such as: `? + ? * 10`	The data type of the other operand.
Labelled duration within a datetime expression. (Note that the portion of a labelled duration that indicates the type of units cannot be a parameter marker.)	DECIMAL(15,0)
Any other operand of a datetime expression (for instance 'timecol + ?' or '? - datecol').	Error
Both operands of a CONCAT operator	Error
One operand of a CONCAT operator where the other operand is a non-CLOB character data type	If one operand is either CHAR(n) or VARCHAR(n), where n is less than 128, then other is VARCHAR(254 - n). In all other cases the data type is VARCHAR(254).
One operand of a CONCAT operator where the other operand is a non-DBCLOB graphic data type.	If one operand is either GRAPHIC(n) or VARGRAPHIC(n), where n is less than 64, then other is VARCHAR(127 - n). In all other cases the data type is VARCHAR(127).
One operand of a CONCAT operator where the other operand is a large object string.	Same as that of the other operand.

Table 28. Untyped Parameter Marker Usage (continued)

Untyped Parameter Marker Location	Data Type
As a value on the right hand side of a SET clause of an UPDATE statement.	The data type of the column. If the column is defined as a user-defined distinct type, then it is the source data type of the user-defined distinct type. If the column is defined as a user-defined structured type, then it is the structured type, also indicating the returns type of the transform function.
The expression following CASE in a simple CASE expression	Error
At least one of the result-expressions in a CASE expression (both Simple and Searched) with the rest of the result-expressions either untyped parameter marker or NULL.	Error
Any or all expressions following WHEN in a simple CASE expression.	Result of applying the "Rules for Result Data Types" on page 107 to the expression following CASE and the expressions following WHEN that are not untyped parameter markers.
A result-expression in a CASE expression (both Simple and Searched) where at least one result-expression is not NULL and not an untyped parameter marker.	Result of applying the Rules for Result Data Types to all result-expressions that are other than NULL or untyped parameter markers.
Alone as a column-expression in a single-row VALUES clause that is not within an INSERT statement.	Error.
Alone as a column-expression in a multi-row VALUES clause that is not within an INSERT statement, and for which the column-expressions in the same position in all other row-expressions are untyped parameter markers.	Error
Alone as a column-expression in a multi-row VALUES clause that is not within an INSERT statement, and for which the expression in the same position of at least one other row-expression is not an untyped parameter marker or NULL.	Result of applying the "Rules for Result Data Types" on page 107 on all operands that are other than untyped parameter markers.

Table 28. Untyped Parameter Marker Usage (continued)

Untyped Parameter Marker Location	Data Type
Alone as a column-expression in a single-row VALUES clause within an INSERT statement.	The data type of the column. If the column is defined as a user-defined distinct type, then it is the source data type of the user-defined distinct type. If the column is defined as a user-defined structured type, then it is the structured type, also indicating the returns type of the transform function.
Alone as a column-expression in a multi-row VALUES clause within an INSERT statement.	The data type of the column. If the column is defined as a user-defined distinct type, then it is the source data type of the user-defined distinct type. If the column is defined as a user-defined structured type, then it is the structured type, also indicating the returns type of the transform function.
As a value on the right side of a SET special register statement	The data type of the special register.
Predicates	
Both operands of a comparison operator	Error
One operand of a comparison operator where the other operand other than an untyped parameter marker.	The data type of the other operand
All operands of a BETWEEN predicate	Error
Either 1st and 2nd, or 1st and 3rd operands of a BETWEEN predicate	Same as that of the only non-parameter marker.
Remaining BETWEEN situations (i.e. one untyped parameter marker only)	Result of applying the "Rules for Result Data Types" on page 107 on all operands that are other than untyped parameter markers.
All operands of an IN predicate	Error
Both the 1st and 2nd operands of an IN predicate.	Result of applying the Rules for Result Data Types on all operands of the IN list (operands to the right of IN keyword) that are other than untyped parameter markers.
The 1st operand of an IN predicate where the right hand side is a fullselect.	Data type of the selected column

Table 28. Untyped Parameter Marker Usage (continued)

Untyped Parameter Marker Location	Data Type
Any or all operands of the IN list of the IN predicate	Results of applying the Rules for Result Data Types on all operands of the IN predicate (operands to the left and right of the IN predicate) that are other than untyped parameter markers.
The 1st operand and zero or more operands in the IN list excluding the 1st operand of the IN list	Result of applying the Rules for Result Data Types on all operands of the IN list (operands to the right of IN keyword) that are other than untyped parameter markers.
All three operands of the LIKE predicate.	Match expression (operand 1) and pattern expression (operand 2) are VARCHAR(32672). Escape expression (operand 3) is VARCHAR(2).
The match expression of the LIKE predicate when either the pattern expression or the escape expression is other than an untyped parameter marker.	Either VARCHAR(32672) or VARGRAPHIC(16336) depending on the data type of the first operand that is not an untyped parameter marker.
The pattern expression of the LIKE predicate when either the match expression or the escape expression is other than an untyped parameter marker.	Either VARCHAR(32672) or VARGRAPHIC(16336) depending on the data type of the first operand that is not an untyped parameter marker. If the data type of the match expression is BLOB, the data type of the pattern expression is assumed to be BLOB(32672).
The escape expression of the LIKE predicate when either the match expression or the pattern expression is other than an untyped parameter marker.	Either VARCHAR(2) or VARGRAPHIC(1) depending on the data type of the first operand that is not an untyped parameter marker. If the data type of the match expression or pattern expression is BLOB, the data type of the escape expression is assumed to be BLOB(1).
Operand of the NULL predicate	error
Functions	
All operands of COALESCE (also called VALUE) or NULLIF	Error
Any operand of COALESCE where at least one operand is other than an untyped parameter marker.	Result of applying the "Rules for Result Data Types" on page 107 on all operands that are other than untyped parameter markers.

Table 28. Untyped Parameter Marker Usage (continued)

Untyped Parameter Marker Location	Data Type
An operand of NULLIF where the other operand is other than an untyped parameter marker.	The data type of the other operand
POSSTR (both operands)	Both operands are VARCHAR(32672).
POSSTR (one operand where the other operand is a character data type).	VARCHAR(32672).
POSSTR (one operand where the other operand is a graphic data type).	VARGRAPHIC(16336).
POSSTR (the search-string operand when the other operand is a BLOB).	BLOB(32672).
SUBSTR (1st operand)	VARCHAR(32672)
SUBSTR (2nd and 3rd operands)	INTEGER
The 1st operand of the TRANSLATE scalar function.	Error
The 2nd and 3rd operands of the TRANSLATE scalar function.	VARCHAR(32672) if the first operand is a character type. VARGRAPHIC(16336) if the first operand is a graphic type.
The 4th operand of the TRANSLATE scalar function.	VARCHAR(1) if the first operand is a character type. VARGRAPHIC(1) if the first operand is a graphic type.
The 2nd operand of the TIMESTAMP scalar function.	TIME
Unary minus	DOUBLE PRECISION
Unary plus	DOUBLE PRECISION
All other operands of all other scalar functions including user-defined functions.	Error
Operand of a column function	Error

Notes

- When a PREPARE statement is executed, the statement string is parsed and checked for errors. If the statement string is invalid, the error condition is reported in the SQLCA. Any subsequent EXECUTE or OPEN statement that references this statement will also receive the same error (due to an implicit prepare done by the system) unless the error has been corrected.
- Prepared statements can be referred to in the following kinds of statements, with the restrictions shown:

In...	The prepared statement ...
DECLARE CURSOR	must be SELECT

EXECUTE	must *not* be SELECT

- A prepared statement can be executed many times. Indeed, if a prepared statement is not executed more than once and does not contain parameter markers, it is more efficient to use the EXECUTE IMMEDIATE statement rather than the PREPARE and EXECUTE statements.
- Statement caching affects repeated preparations. See the "Notes" on page 896 for information.
- See the *Application Development Guide* for examples of dynamic SQL statements in the supported host languages.

Examples

Example 1: Prepare and execute a non-select-statement in a COBOL program. Assume the statement is contained in a host variable HOLDER and that the program will place a statement string into the host variable based on some instructions from the user. The statement to be prepared does not have any parameter markers.

```
EXEC SQL  PREPARE STMT_NAME FROM :HOLDER
END-EXEC.

EXEC SQL  EXECUTE STMT_NAME
END-EXEC.
```

Example 2: Prepare and execute a non-select-statement as in example 1, except code it for a C program. Also assume the statement to be prepared can contain any number of parameter markers.

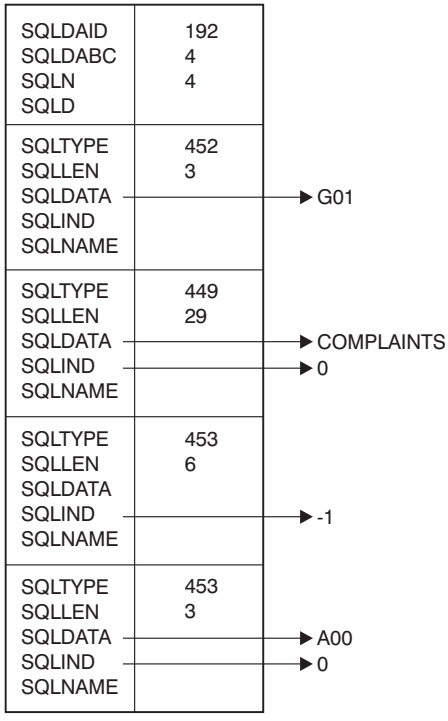
```
EXEC SQL  PREPARE STMT_NAME FROM :holder;
EXEC SQL  EXECUTE STMT_NAME USING DESCRIPTOR :insert_da;
```

Assume that the following statement is to be prepared:

```
INSERT INTO DEPT VALUES(?, ?, ?, ?)
```

The columns in the DEPT table are defined as follows:

```
DEPT_NO   CHAR(3) NOT NULL, -- department number
DEPTNAME  VARCHAR(29), -- department name
MGRNO     CHAR(6), -- manager number
ADMRDEPT  CHAR(3)  -- admin department number
```



To insert department number G01 named COMPLAINTS, which has no manager and reports to department A00, the structure INSERT_DA should have the above values before issuing the EXECUTE statement.

REFRESH TABLE

The REFRESH TABLE statement refreshes the data in a summary 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.

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 table.

Syntax

Description

table-name
Specifies a table name.

The name, including the implicit or explicit schema, must identify a table that already exists at the current server. The table must allow the REFRESH TABLE statement (SQLSTATE 42809). This includes summary tables defined with:

- REFRESH IMMEDIATE
- REFRESH DEFERRED

RELEASE (Connection)

This statement places one or more connections in the release-pending state.

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

None Required.

Syntax

Notes:

1 Note that an application server named CURRENT or ALL can only be identified by a host variable.

Description

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.

The specified database-alias or the database-alias contained in the host variable must identify an existing connection of the application process. If the database-alias does not identify an existing connection, an error (SQLSTATE 08003) is raised.

CURRENT
Identifies the current connection of the application process. The application process must be in the connected state. If not, an error (SQLSTATE 08003) is raised.

ALL
Identifies all existing connections of the application process. This form of the RELEASE statement places all existing connections of the application process in the release-pending state. All connections will therefore be destroyed during the next commit operation. An error or warning does not occur if no connections exist when the statement is executed. The optional keyword SQL is included to be compatible with DB2/MVS SQL syntax.

Notes

Examples

Example 1: The SQL connection to IBMSTHDB is no longer needed by the application. The following statement will cause it to be destroyed during the next commit operation:

```
EXEC SQL RELEASE IBMSTHDB;
```

Example 2: The current connection is no longer needed by the application. The following statement will cause it to be destroyed during the next commit operation:

```
EXEC SQL RELEASE CURRENT;
```

Example 3: If an application has no need to access the databases after a commit but will continue to run for a while, then it is better not to tie up those connections unnecessarily. The following statement can be executed before the commit to ensure all connections will be destroyed at the commit:

```
EXEC SQL RELEASE ALL;
```

RELEASE SAVEPOINT

The RELEASE SAVEPOINT statement is used to indicate that the application no longer wishes to have the named savepoint maintained. After this statement has been invoked, rollback to the savepoint is no longer possible.

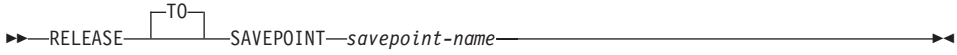
Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared.

Authorization

None required.

Syntax



Description

savepoint-name
The named savepoint is released. Rollback to that savepoint is no longer possible. If the named savepoint does not exist, an error is issued (SQLSTATE 3B001).

Notes

- The name of the savepoint that was released can now be re-used in another SAVEPOINT statement, regardless of whether the UNIQUE keyword was specified on an earlier SAVEPOINT statement specifying this same savepoint name.

Example

Example 1: Release a savepoint named SAVEPOINT1.

```
RELEASE SAVEPOINT SAVEPOINT1
```

RENAME TABLE

The RENAME TABLE statement renames an existing table.

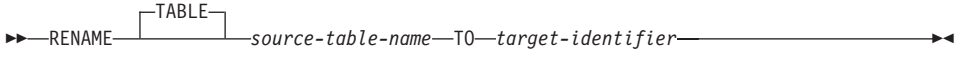
Invocation

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. 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 either SYSADM or DBADM authority or CONTROL privilege.

Syntax



Description

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 can be an alias identifying the table. It must not be the name of a catalog table (SQLSTATE 42832), a summary table, a typed table (SQLSTATE 42997), a nickname, or an object of other than table or alias (SQLSTATE 42809).

target-identifier
Specifies the new name for the table without a schema name. The schema name of the *source-table-name* is used to qualify the new name for the table. The qualified name must *not* identify a table, view, or alias that already exists in the database (SQLSTATE 42710).

Rules

The source table must not:

- Be referenced in any existing view definitions or summary 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.

Notes

- Catalog entries are updated to reflect the new table name.
- *All* authorizations associated with the source table name are *transferred* to the new table 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).
- Any packages that are dependent on the source table are invalidated.
- 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 may be renamed if none of the primary key or unique constraints are referenced by any foreign key.

Example

Change the name of the EMP table to EMPLOYEE.

```
RENAME TABLE EMP TO EMPLOYEE
```

RENAME TABLESPACE

The RENAME TABLESPACE statement renames an existing 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. 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 either SYSADM or SYSCTRL authority.

Syntax

Description

source-tablespace-name
Specifies the existing table space that is to be renamed, as a one-part name. It is an SQL identifier (either ordinary or delimited). The table space name must identify a table space that already exists in the catalog (SQLSTATE 42704).

target-tablespace-name
Specifies the new name for the table space, as a one-part name. It is an SQL identifier (either ordinary or delimited). The new table space name must *not* identify a table space that already exists in the catalog (SQLSTATE 42710), and it cannot start with 'SYS' (SQLSTATE 42939).

Rules

- The SYSCATSPACE table space cannot be renamed (SQLSTATE 42832).
- Any table spaces with "rollforward pending" or "rollforward in progress" states cannot be renamed (SQLSTATE 55039)

Notes

- Renaming a table space will update the minimum recovery time of a table space to the point in time when the rename took place. This implies that a roll forward at the table space level must be to at least this point in time.
- The new table space name must be used when restoring a table space from a backup image, where the rename was done after the backup was created. Refer to the *Administrative API Reference* or the *Command Reference* for more information on restoring backups.

Example

Change the name of the table space USERSPACE1 to DATA2000:

```
RENAME TABLESPACE USERSPACE1 TO DATA2000
```

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

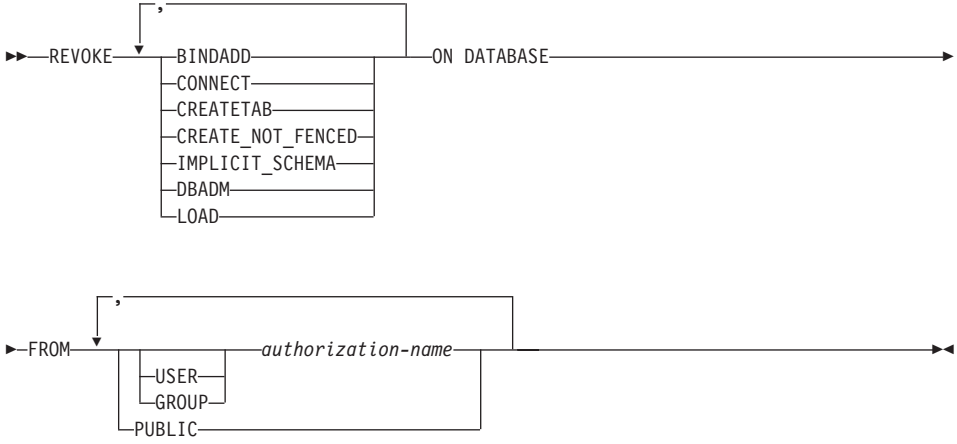
Authorization

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

- 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_NOT_FENCED
Revokes the authority to register functions that execute in the database manager's process. However, once a function has been registered as not fenced, it continues to run in this manner even if CREATE_NOT_FENCED is subsequently revoked from the authorization ID that registered the function.

The CREATE_NOT_FENCED 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).

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 BINDADD, CONNECT, CREATETAB, IMPLICIT_SCHEMA, or CREATE_NOT_FENCED authority.

LOAD
Revoke the authority to LOAD in this database.

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.

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.
  - If DCE authentication is used, then an error is raised (SQLSTATE 56092).

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 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).

REVOKE (Index Privileges)

This form of the REVOKE statement revokes the CONTROL privilege on an index.

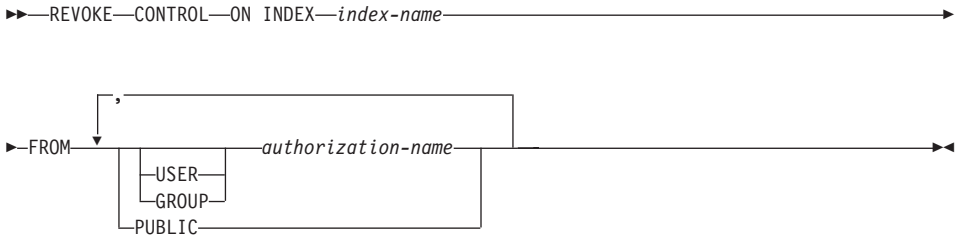
Invocation

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

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.
  If DCE authentication is used, then an error is raised (SQLSTATE 56092).

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
```

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

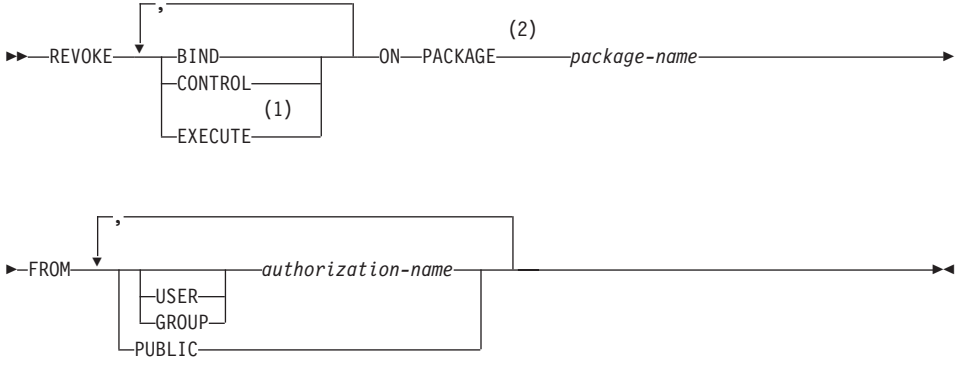
Authorization

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

- 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 the referenced package.

The BIND privileges 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 *package-name*
Specifies the package on which privileges are 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.

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.
  - If DCE authentication is used, then an error is raised (SQLSTATE 56092).

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
```

REVOKE (Schema Privileges)

This form of the REVOKE statement revokes the privileges on a schema.

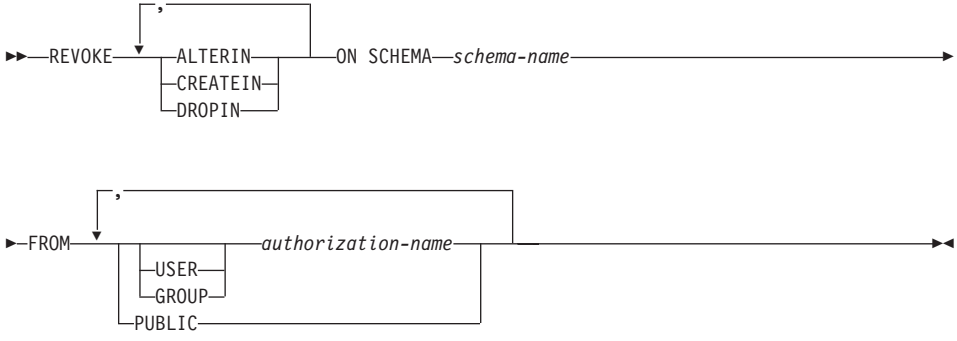
Invocation

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

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.
  - If DCE authentication is used, then an error is raised (SQLSTATE 56092).

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
```

REVOKE (Server Privileges)

This form of the REVOKE statement revokes the privilege to access and use a specified data source in pass-through mode.

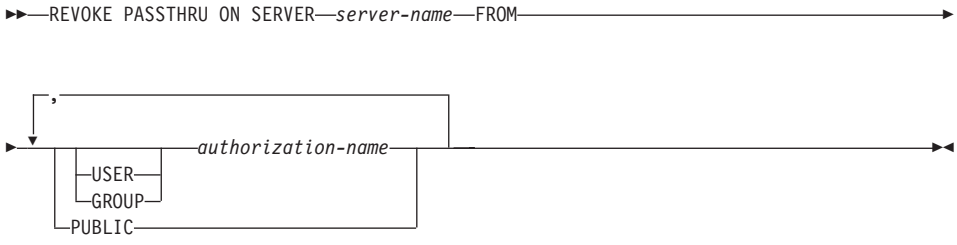
Invocation

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. 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 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.*

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.

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

Authorization

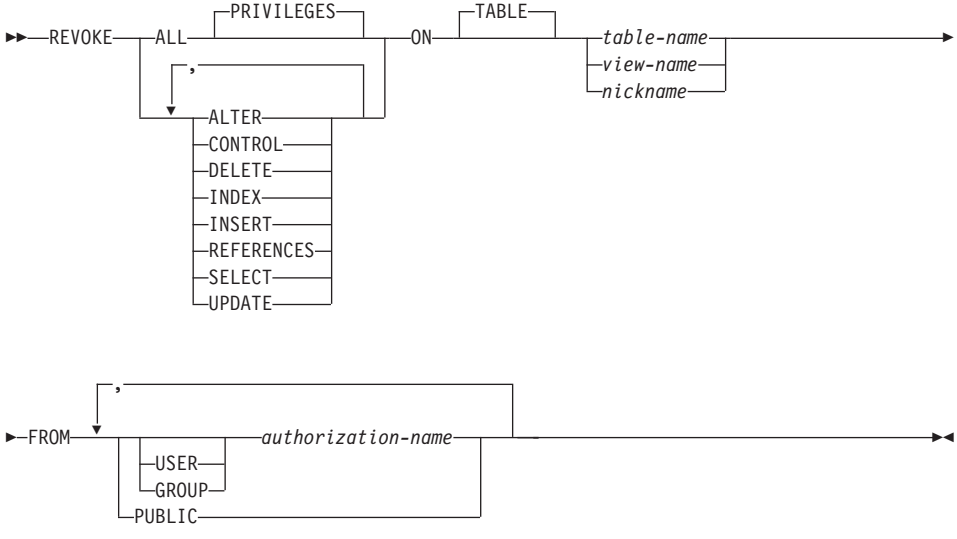
The privileges held by the authorization ID of the statement must include at least one of the following:

- SYSADM or DBADM authority
- 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 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 or updatable view.

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 or updatable view, 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 "Notes" on page 832.

UPDATE
Revokes the privilege to update rows in the table or updatable view. 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.

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.
  - If DCE authentication is used, then an error is raised (SQLSTATE 56092).

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)), then the view will be made inoperative (see the "Notes" section in "CREATE VIEW" on page 823 for information on inoperative views).

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 summary table loses a SELECT privilege on a table on which the summary table definition depends, (or a table upon which the summary table definition depends is dropped), then the summary table will be made inoperative (see the "Notes" on page 753 for information on inoperative summary tables).

  However, if a DBADM or SYSADM explicitly revokes all privileges on the summary table from the DEFINER, then the record in SYSTABAUTH for the DEFINER will be deleted, but nothing will happen to the summary 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 may fail if the SQL function cannot be dropped because some other object is dependent on the function (SQLSTATE 42893).

Note: "Rules" on page 884 lists the dependencies that objects such as tables and views can have on one another.

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
```

REVOKE (Table Space Privileges)

This form of the REVOKE statement revokes the USE privilege on a table space.

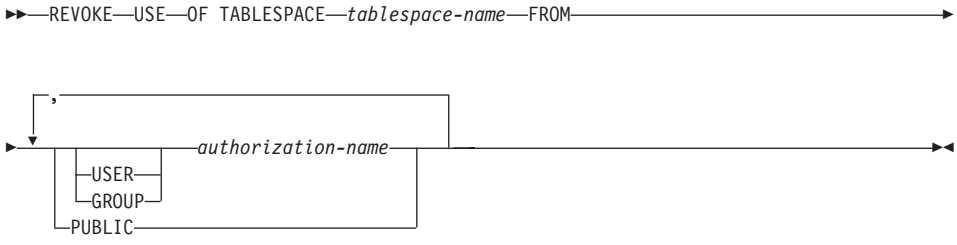
Invocation

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

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).
  - If DCE authentication is used, 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
```

ROLLBACK

The ROLLBACK statement is used to back out of the database changes that were made within a unit of work or a savepoint.

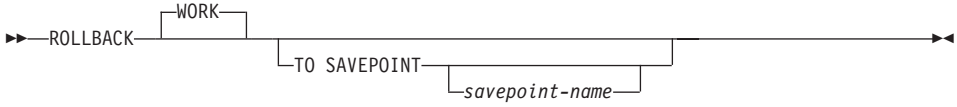
Invocation

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 issuing the ROLLBACK statement:

- SET CONNECTION,
- SET CURRENT DEGREE,
- SET CURRENT DEFAULT TRANSFORM GROUP,
- SET CURRENT EXPLAIN MODE,
- SET CURRENT EXPLAIN SNAPSHOT,
- SET CURRENT PACKAGESET,
- SET CURRENT QUERY OPTIMIZATION,
- SET CURRENT REFRESH AGE,
- SET EVENT MONITOR STATE,
- SET PASSTHRU,
- SET PATH,
- SET SCHEMA,
- SET SERVER OPTION.

TO SAVEPOINT
Indicates that a partial rollback (ROLLBACK TO SAVEPOINT) is to be performed. If no savepoint is active, an SQL error is returned (SQLSTATE

3B502). After a successful ROLLBACK, the savepoint continues to exist. If a *savepoint-name* is not provided, rollback is to the most recently set savepoint.

If this clause is omitted, the ROLLBACK WORK statement rolls back the entire transaction. Furthermore, savepoints within the transaction are released.

savepoint-name
Indicate the savepoint to which to rollback. After a successful ROLLBACK, the savepoint defined by *savepoint-name* continues to exist. If the savepoint name does not exist, an error is returned (SQLSTATE 3B001). 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. See the "Notes" on page 896 for information.
- Savepoints are not allowed in atomic execution contexts such as atomic compound statements and triggers.
- 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.[104]
    - 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.

104. A FETCH must be performed before a positioned UPDATE or DELETE statement is issued.

- 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
```