function defined no longer has the SELECT WITH GRANT OPTION privilege. If such a function is used in a view or trigger, it cannot be dropped and the REVOKE is restricted as a result. Otherwise, the REVOKE cascades and drops such functions.

Notes

- It is valid to drop a user-defined function while it is in use. Also, a cursor can be open over a statement which contains a reference to a user-defined function, and while this cursor is open the function can be dropped without causing the cursor fetches to fail.
- If a package which depends on a user-defined function is executing, it is not possible for another authorization ID to drop the function until the package completes its current unit of work. At that point, the function is dropped and the package becomes inoperative. The next request for this package results in an error indicating that the package must be explicitly rebound.
- The removal of a function body (this is very different from dropping the function) can occur while an application which needs the function body is executing. This may or may not cause the statement to fail, depending on whether the function body still needs to be loaded into storage by the database manager on behalf of the statement.
- For any dropped table that includes currently linked files through DATALINK columns, the files are unlinked, and will be either restored or deleted, depending on the datalink column definition.
- If a table containing a DATALINK column is dropped while any DB2 Data Links Managers configured to the database are unavailable, either through DROP TABLE or DROP TABLESPACE, then the operation will fail (SQLSTATE 57050).
- In addition to the dependencies recorded for any explicitly specified UDF, the following dependencies are recorded when transforms are implicitly required:
  1. When the structured type parameter or result of a function or method requires a transform, a dependency is recorded for the function or method on the required TO SQL or FROM SQL transform function.
  2. When an SQL statement included in a package requires a transform function, a dependency is recorded for the package on the designated TO SQL or FROM SQL transform function.

  Since the above describes the only circumstances under which dependencies are recorded due to implicit invocation of transforms, no objects other than functions, methods, or packages can have a dependency on implicitly invoked transform functions. On the other hand, explicit calls to transform functions (in views and triggers, for example) do result in the usual dependencies of these other types of objects on transform functions. As a

result, a DROP TRANSFORM statement may also fail due to these "explicit" type dependencies of objects on the transform(s) being dropped (SQLSTATE 42893).

- Since the dependency catalogs do not distinguish between depending on a function as a transform versus depending on a function by explicit function call, it is suggested that explicit calls to transform functions are not written. In such an instance, the transform property on the function cannot be dropped, or packages will be marked inoperative, simply because they contain explicit invocations in an SQL expression.

Examples

Example 1: Drop table TDEPT.

```
DROP TABLE TDEPT
```

Example 2: Drop the view VDEPT.

```
DROP VIEW VDEPT
```

Example 3: The authorization ID HEDGES attempts to drop an alias.

```
DROP ALIAS A1
```

The alias HEDGES.A1 is removed from the catalogs.

Example 4: Hedges attempts to drop an alias, but specifies T1 as the alias-name, where T1 is the name of an existing table (not the name of an alias).

```
DROP ALIAS T1
```

This statement fails (SQLSTATE 42809).

Example 5:

Drop the BUSINESS_OPS nodegroup. To drop the nodegroup, the two table spaces (ACCOUNTING and PLANS) in the nodegroup must first be dropped.

```
DROP TABLESPACE ACCOUNTING
DROP TABLESPACE PLANS
DROP NODEGROUP BUSINESS_OPS
```

Example 6: Pellow wants to drop the CENTRE function, which he created in his PELLOW schema, using the signature to identify the function instance to be dropped.

```
DROP FUNCTION CENTRE (INT,FLOAT)
```

Example 7: McBride wants to drop the FOCUS92 function, which she created in the PELLOW schema, using the specific name to identify the function instance to be dropped.

```
DROP SPECIFIC FUNCTION PELLOW.FOCUS92
```

Example 8: Drop the function ATOMIC_WEIGHT from the CHEM schema, where it is known that there is only one function with that name.

```
DROP FUNCTION CHEM.ATOMIC_WEIGHT
```

Example 9: Drop the trigger SALARY_BONUS, which caused employees under a specified condition to receive a bonus to their salary.

```
DROP TRIGGER SALARY_BONUS
```

Example 10: Drop the distinct data type named shoesize, if it is not currently in use.

```
DROP DISTINCT TYPE SHOESIZE
```

Example 11: Drop the SMITHPAY event monitor.

```
DROP EVENT MONITOR SMITHPAY
```

Example 12: Drop the schema from Example 2 under CREATE SCHEMA using RESTRICT. Notice that the table called PART must be dropped first.

```
DROP TABLE PART
DROP SCHEMA INVENTRY RESTRICT
```

Example 13: Macdonald wants to drop the DESTROY procedure, which he created in the EIGLER schema, using the specific name to identify the procedure instance to be dropped.

```
DROP SPECIFIC PROCEDURE  EIGLER.DESTROY
```

Example 14: Drop the procedure OSMOSIS from the BIOLOGY schema, where it is known that there is only one procedure with that name.

```
DROP PROCEDURE BIOLOGY.OSMOSIS
```

Example 15: User SHAWN used one authorization ID to access the federated database and another to access the database at an Oracle data source called ORACLE1. A mapping was created between the two authorizations, but SHAWN no longer needs to access the data source. Drop the mapping.

```
DROP USER MAPPING FOR SHAWN SERVER ORACLE1
```

Example 16: An index of a data source table that a nickname references has been deleted. Drop the index specification that was created to let the optimizer know about this index.

```
DROP INDEX INDEXSPEC
```

Example 17: Drop the MYSTRUCT1 transform group.

```
DROP TRANSFORM MYSTRUCT1 FOR POLYGON
```

Example 18: Drop the method BONUS for the EMP data type in the PERSONNEL schema.

```
DROP METHOD BONUS (SALARY DECIMAL(10,2)) FOR PERSONNEL.EMP
```

END DECLARE SECTION

The END DECLARE SECTION statement marks the end of a host variable declare section.

Invocation

This statement can only be embedded in an application program. It is not an executable statement. It must not be specified in REXX.

Authorization

None required.

Syntax

Description

The END DECLARE SECTION statement can be coded in the application program wherever declarations can appear according to the rules of the host language. It indicates the end of a host variable declaration section. A host variable section starts with a BEGIN DECLARE SECTION statement (see "BEGIN DECLARE SECTION" on page 520).

The BEGIN DECLARE SECTION and the END DECLARE SECTION statements must be paired and may not be nested.

Host variable declarations can be specified by using the SQL INCLUDE statement. Otherwise, a host variable declaration section must not contain any statements other than host variable declarations.

Host variables referenced in SQL statements must be declared in a host variable declare section in all host languages, other than REXX.[97] Furthermore, the declaration of each variable must appear before the first reference to the variable.

Variables declared outside a declare section must not have the same name as variables declared within a declare section.

Example

See "BEGIN DECLARE SECTION" on page 520 for examples that use the END DECLARE SECTION statement.

97. See "Rules" on page 520 for information on how host variables can be declared in REXX in the case of LOB locators and file reference variables.

EXECUTE

The EXECUTE statement executes a prepared SQL statement.

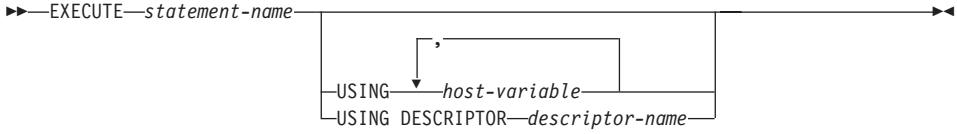
Invocation

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 execution time (DDL, GRANT, and REVOKE statements), 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 preparation time (DML), no authorization is required to use this statement.

Syntax



Description

statement-name
Identifies the prepared statement to be executed. The *statement-name* must identify a statement that was previously prepared and the prepared statement must not be a SELECT statement.

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

host-variable, **...**
Identifies a host variable that is declared 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. Locator variables and file reference variables, where appropriate, can be provided as the source of values for parameter markers.

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

Before the EXECUTE statement is processed, the user must set the following fields in the input 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 input data needs to be accommodated, there must be two SQLVAR entries for every parameter marker.

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.

Notes

- Before the prepared statement is executed, each parameter marker is effectively replaced by the value of 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 prepared statement is executed, 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.

- ***Dynamic SQL Statement Caching:***

  The information required to execute dynamic and static SQL statements is placed in the database package cache when static SQL statements are first referenced or when dynamic SQL statements are first prepared. This information stays in the package cache until it becomes invalid, the cache space is required for another statement, or the database is shut down.

  When an SQL statement is executed or prepared, the package information relevant to the application issuing the request is loaded from the system catalog into the package cache. The actual executable section for the individual SQL statement is also placed into the cache: static SQL sections are read in from the system catalog and placed in the package cache when the statement is first referenced; Dynamic SQL sections are placed directly in the cache after they have been created. Dynamic SQL sections can be created by an explicit statement, such as a PREPARE or EXECUTE IMMEDIATE statement. Once created, sections for dynamic SQL statements may be recreated by an implicit prepare of the statement performed by the system if the original section has been deleted for space management reasons or has become invalid due to changes in the environment.

  Each SQL statement is cached at a database level and can be shared among applications. Static SQL statements are shared among applications using the same package; Dynamic SQL statements are shared among applications using the same compilation environment and the exact same statement text. The text of each SQL statement issued by an application is cached locally within the application for use in the event that an implicit prepare is required. Each PREPARE statement in the application program can cache one statement. All EXECUTE IMMEDIATE statements in an application program share the same space and only one cached statement exists for all these EXECUTE IMMEDIATE statements at a time. If the same PREPARE or any EXECUTE IMMEDIATE statement is issued multiple times with a different SQL statement each time, only the last statement will be cached for reuse. The optimal use of the cache is to issue a number of different PREPARE statements once at the start of the application and then to issue an EXECUTE or OPEN statement as required.

  With the caching of dynamic SQL statements, once a statement has been created, it can be reused over multiple units of work without the need to prepare the statement again. The system will recompile the statement as required if environment changes occur.

  The following events are examples of environment or data object changes which can cause cached dynamic statements to be implicitly prepared on the next PREPARE, EXECUTE, EXECUTE IMMEDIATE, or OPEN request:

- ALTER NICKNAME
- ALTER SERVER
- ALTER TABLE
- ALTER TABLESPACE
- ALTER TYPE
- CREATE FUNCTION
- CREATE FUNCTION MAPPING
- CREATE INDEX
- CREATE TABLE
- CREATE TEMPORARY TABLESPACE
- CREATE TRIGGER
- CREATE TYPE
- DROP (all objects)
- RUNSTATS on any table or index
- any action that causes a view to become inoperative
- UPDATE of statistics in any system catalog table
- SET CURRENT DEGREE
- SET PATH
- SET QUERY OPTIMIZATION
- SET SCHEMA
- SET SERVER OPTION

The following list outlines the behavior that can be expected from cached dynamic SQL statements:

- *PREPARE Requests:* Subsequent preparations of the same statement will not incur the cost of compiling the statement if the section is still valid. The cost and cardinality estimates for the current cached section will be returned. These values may differ from the values returned from any previous PREPARE for the same SQL statement.

  There will be no need to issue a PREPARE statement subsequent to a COMMIT or ROLLBACK statement.
- *EXECUTE Requests:* EXECUTE statements may occasionally incur the cost of implicitly preparing the statement if it has become invalid since the original PREPARE. If a section is implicitly prepared, it will use the current environment and not the environment of the original PREPARE statement.
- *EXECUTE IMMEDIATE Requests:* Subsequent EXECUTE IMMEDIATE statements for the same statement will not incur the cost of compiling the statement if the section is still valid.

- *OPEN Requests:* OPEN requests for dynamically defined cursors may occasionally incur the cost of implicitly preparing the statement if it has become invalid since the original PREPARE statement. If a section is implicitly prepared, it will use the current environment and not the environment of the original PREPARE statement.
- *FETCH Requests:* No behavior changes should be expected.
- *ROLLBACK:* Only those dynamic SQL statements prepared or implicitly prepared during the unit of work affected by the rollback operation will be invalidated.
- *COMMIT:* Dynamic SQL statements will not be invalidated but any locks acquired will be freed. Cursors not defined as WITH HOLD cursors will be closed and their locks freed. Open WITH HOLD cursors will hold onto their package and section locks to protect the active section during, and after, commit processing.

If an error occurs during an implicit prepare, an error will be returned for the request causing the implicit prepare (SQLSTATE 56098).

Examples

Example 1: In this C example, an INSERT statement with parameter markers is prepared and executed. `h1 - h4` are host variables that correspond to the format of TDEPT.

```
strcpy (s,"INSERT INTO TDEPT VALUES(?,?,?,?)");
EXEC SQL PREPARE DEPT_INSERT FROM :s;
   .
   .
(Check for successful execution and put values into :h1, :h2, :h3, :h4)
   .
   .
EXEC SQL EXECUTE DEPT_INSERT USING :h1, :h2,
:h3, :h4;
```

Example 2: This EXECUTE statement uses an SQLDA.

```
EXECUTE S3 USING DESCRIPTOR :sqlda3
```

EXECUTE IMMEDIATE

The EXECUTE IMMEDIATE statement:

- Prepares an executable form of an SQL statement from a character string form of the statement.
- Executes the SQL statement.

EXECUTE IMMEDIATE combines the basic functions of the PREPARE and EXECUTE statements. It can be used to prepare and execute SQL statements that contain neither host variables nor parameter markers.

Invocation

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

Authorization

The authorization rules are those defined for the SQL statement specified by EXECUTE IMMEDIATE.

Syntax

Description

host-variable
A host variable must be specified and it must identify a host variable that is described in the program in accordance with the rules for declaring character-string variables. It must be a character-string variable less than the maximum statement size of 65 535. Note that a CLOB(65535) can contain a maximum size statement but a VARCHAR can not.

The value of the identified host variable is called the statement string.

The statement string must be one of the following SQL statements:

- ALTER
- COMMENT ON
- COMMIT
- CREATE
- DELETE
- DECLARE GLOBAL TEMPORARY TABLE
- DROP
- GRANT
- INSERT

- LOCK TABLE
- REFRESH TABLE
- RELEASE SAVEPOINT
- RENAME TABLE
- RENAME TABLESPACE
- REVOKE
- ROLLBACK
- SAVEPOINT
- SET CURRENT DEGREE
- SET CURRENT EXPLAIN MODE
- SET CURRENT EXPLAIN SNAPSHOT
- SET CURRENT QUERY OPTIMIZATION
- SET CURRENT REFRESH AGE
- SET CURRENT TRANSFORM GROUP
- SET EVENT MONITOR STATE
- SET INTEGRITY
- SET PASSTHRU
- SET PATH
- SET SCHEMA
- SET SERVER OPTION
- UPDATE

The statement string must not include parameter markers or references to host variables, and must not begin with EXEC SQL. It must not contain a statement terminator, with the exception of the CREATE TRIGGER statement which can contain a semi-colon (;) to separate triggered SQL statements, or the CREATE PROCEDURE statement to separate SQL statements in the SQL procedure body.

When an EXECUTE IMMEDIATE statement is executed, the specified statement string is parsed and checked for errors. If the SQL statement is invalid, it is not executed and the error condition that prevents its execution is reported in the SQLCA. If the SQL statement is valid, but an error occurs during its execution, that error condition is reported in the SQLCA.

Notes

- Statement caching affects the behavior of an EXECUTE IMMEDIATE statement. See “Dynamic SQL Statement Caching” on page 897 for information.

Example

Use C program statements to move an SQL statement to the host variable qstring (char[80]) and prepare and execute whatever SQL statement is in the host variable qstring.

```
if ( strcmp(accounts,"BIG") == 0 )
  strcpy (qstring,"INSERT INTO WORK_TABLE SELECT *
    FROM EMP_ACT WHERE ACTNO < 100");
else
  strcpy (qstring,"INSERT INTO WORK_TABLE SELECT *
    FROM EMP_ACT WHERE ACTNO >= 100");
  .
  .
  .
EXEC SQL  EXECUTE IMMEDIATE :qstring;
```

EXPLAIN

The EXPLAIN statement captures information about the access plan chosen for the supplied explainable statement and places this information into the Explain tables. (See "Appendix K. Explain Tables and Definitions" on page 1291 for information on the Explain tables and table definitions.)

An *explainable statement* is a DELETE, INSERT, SELECT, SELECT INTO, UPDATE, VALUES, or VALUES INTO SQL statement.

Invocation

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

The statement to be explained is not executed.

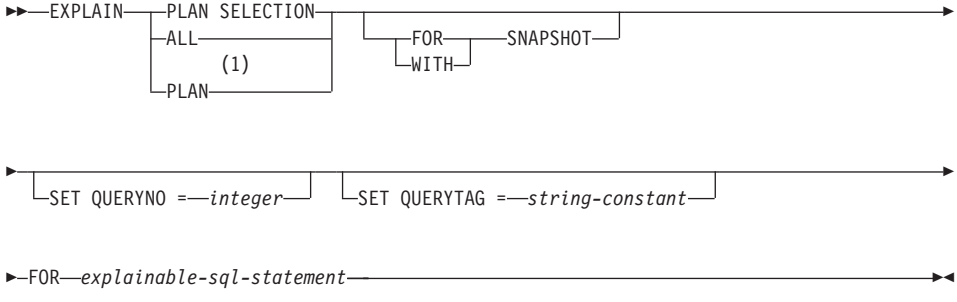
Authorization

The authorization rules are those defined for the SQL statement specified in the EXPLAIN statement. For example, if a DELETE statement was used as the *explainable-sql-statement* (see statement syntax that follows), then the authorization rules for a DELETE statement would be applied when the DELETE statement is explained.

The authorization rules for static EXPLAIN statements are those rules that apply for static versions of the statement passed as the *explainable-sql-statement*. Dynamically prepared EXPLAIN statements use the authorization rules for the dynamic preparation of the statement provided for the *explainable-sql-statement* parameter.

The current authorization ID must have insert privilege on the Explain tables.

Syntax



Notes:

1 The PLAN option is supported only for syntax toleration of existing DB2

for MVS EXPLAIN statements. There is no PLAN table.
Specifying PLAN is equivalent to specifying PLAN SELECTION.

Description

PLAN SELECTION
Indicates that the information from the plan selection phase of SQL compilation is to be inserted into the Explain tables.

ALL
Specifying ALL is equivalent to specifying PLAN SELECTION.

PLAN
The PLAN option provides syntax toleration for existing database applications from other systems. Specifying PLAN is equivalent to specifying PLAN SELECTION.

FOR SNAPSHOT
This clause indicates that only an Explain Snapshot is to be taken and placed into the SNAPSHOT column of the EXPLAIN_STATEMENT table. No other Explain information is captured other than that present in the EXPLAIN_INSTANCE and EXPLAIN_STATEMENT tables.

The Explain Snapshot information is intended for use with Visual Explain.

WITH SNAPSHOT
This clause indicates that, in addition to the regular Explain information, an Explain Snapshot is to be taken.

The default behavior of the EXPLAIN statement is to only gather regular Explain information and not the Explain Snapshot.

The Explain Snapshot information is intended for use with Visual Explain.

default (neither FOR SNAPSHOT nor WITH SNAPSHOT specified)
Puts Explain information into the Explain tables. No snapshot is taken for use with Visual Explain.

SET QUERYNO = *integer*
Associates *integer*, via the QUERYNO column in the EXPLAIN_STATEMENT table, with *explainable-sql-statement*. The integer value supplied must be a positive value.

If this clause is not specified for a dynamic EXPLAIN statement, a default value of one (1) is assigned. For a static EXPLAIN statement, the default value assigned is the statement number assigned by the precompiler.

SET QUERYTAG = *string-constant*
Associates *string-constant*, via the QUERYTAG column in the EXPLAIN_STATEMENT table, with *explainable-sql-statement*. *string-constant*

can be any character string up to 20 bytes in length. If the value supplied is less than 20 bytes in length, the value is padded on the right with blanks to the required length.

If this clause is not specified for an EXPLAIN statement, blanks are used as the default value.

FOR *explainable-sql-statement*
Specifies the SQL statement to be explained. This statement can be any valid DELETE, INSERT, SELECT, SELECT INTO, UPDATE, VALUES, or VALUES INTO SQL statement. If the EXPLAIN statement is embedded in a program, the *explainable-sql-statement* can contain references to host variables (these variables must be defined in the program). Similarly, if EXPLAIN is being dynamically prepared, the *explainable-sql-statement* can contain parameter markers.

The *explainable-sql-statement* must be a valid SQL statement that could be prepared and executed independently of the EXPLAIN statement. It cannot be a statement name or host variable. SQL statements referring to cursors defined through CLP are not valid for use with this statement.

To explain dynamic SQL within an application, the entire EXPLAIN statement must be dynamically prepared.

Notes

The following table shows the interaction of the snapshot keywords and the Explain information.

Keyword Specified	Capture Explain Information?	Take Snapshot for Visual Explain?
none	Yes	No
FOR SNAPSHOT	No	Yes
WITH SNAPSHOT	Yes	Yes

If neither the FOR SNAPSHOT nor WITH SNAPSHOT clause is specified, then no Explain snapshot is taken.

The Explain tables must be created by the user prior to the invocation of EXPLAIN. (See "Appendix K. Explain Tables and Definitions" on page 1291 for information on the Explain tables and table definitions.) The information generated by this statement is stored in these explain tables in the schema designated at the time the statement is compiled.

If any errors occur during the compilation of the *explainable-sql-statement* supplied, then no information is stored in the Explain tables.

The access plan generated for the *explainable-sql-statement* is not saved and thus, cannot be invoked at a later time. The Explain information for the *explainable-sql-statement* is inserted when the EXPLAIN statement itself is compiled.

For a static EXPLAIN SQL statement, the information is inserted into the Explain tables at bind time and during an explicit rebind (see REBIND in the *Command Reference*). During precompilation, the static EXPLAIN statements are commented out in the modified application source file. At bind time, the EXPLAIN statements are stored in the SYSCAT.STATEMENTS catalog. When the package is run, the EXPLAIN statement is not executed. Note that the section numbers for all statements in the application will be sequential and will include the EXPLAIN statements. An alternative to using a static EXPLAIN statement is to use a combination of the EXPLAIN and EXPLSNAP BIND/PREP options. Static EXPLAIN statements can be used to cause the Explain tables to be populated for one specific static SQL statement out of many; simply prefix the target statement with the appropriate EXPLAIN statement syntax and bind the application without using either of the Explain BIND/PREP options. The EXPLAIN statement can also be used when it is advantageous to set the QUERYNO or QUERYTAG field at the time of the actual Explain invocation.

For an incremental bind EXPLAIN SQL statement, the Explain tables are populated when the EXPLAIN statement is submitted for compilation. When the package is run, the EXPLAIN statement performs no processing (though the statement will be successful). When populating the explain tables, the explain table qualifier and authorization ID used during population will be those of the package owner. The EXPLAIN statement can also be used when it is advantageous to set the QUERYNO or QUERYTAG field at the time of the actual Explain invocation.

For dynamic EXPLAIN statements, the Explain tables are populated at the time the EXPLAIN statement is submitted for compilation. An Explain statement can be prepared with the PREPARE statement but, if executed, will perform no processing (though the statement will be successful). An alternative to issuing dynamic EXPLAIN statements is to use a combination of the CURRENT EXPLAIN MODE and CURRENT EXPLAIN SNAPSHOT special registers to explain dynamic SQL statements. The EXPLAIN statement should be used when it is advantageous to set the QUERYNO or QUERYTAG field at the time of the actual Explain invocation.

Examples

Example 1: Explain a simple SELECT statement and tag with QUERYNO = 13.

```
EXPLAIN PLAN SET QUERYNO = 13 FOR SELECT C1 FROM T1;
```

This statement is successful.

Example 2:

Explain a simple SELECT statement and tag with QUERYTAG = 'TEST13'.

```
EXPLAIN PLAN SELECTION SET QUERYTAG = 'TEST13'
  FOR SELECT C1 FROM T1;
```

This statement is successful.

Example 3: Explain a simple SELECT statement and tag with QUERYNO = 13 and QUERYTAG = 'TEST13'.

```
EXPLAIN PLAN SELECTION SET QUERYNO = 13 SET QUERYTAG = 'TEST13'
  FOR SELECT C1 FROM T1;
```

This statement is successful.

Example 4: Attempt to get Explain information when Explain tables do not exist.

```
EXPLAIN ALL FOR SELECT C1 FROM T1;
```

This statement would fail as the Explain tables have not been defined (SQLSTATE 42704).

FETCH

The FETCH statement positions a cursor on the next row of its result table and assigns the values of that row to host variables.

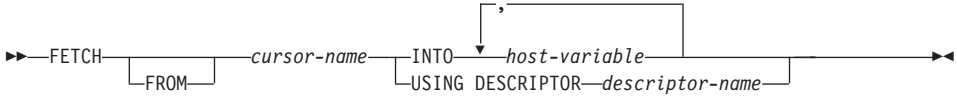
Invocation

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 an explanation of the authorization required to use a cursor.

Syntax



Description

cursor-name
Identifies the cursor to be used in the fetch operation. The *cursor-name* must identify a declared cursor as explained in "DECLARE CURSOR" on page 841. The DECLARE CURSOR statement must precede the FETCH statement in the source program. When the FETCH statement is executed, the cursor must be in the open state.

If the cursor is currently positioned on or after the last row of the result table:

- SQLCODE is set to +100, and SQLSTATE is set to '02000'.
- The cursor is positioned after the last row.
- Values are not assigned to host variables.

If the cursor is currently positioned before a row, it will be repositioned on that row, and values will be assigned to host variables as specified by INTO or USING.

If the cursor is currently positioned on a row other than the last row, it will be repositioned on the next row and values of that row will be assigned to host variables as specified by INTO or USING.

INTO *host-variable*, **...**
Identifies one or more host variables that must be described in accordance with the rules for declaring host variables. The first value in the result

row is assigned to the first host variable in the list, the second value to the second host variable, and so on. For LOB values in the select-list, the target can be a regular host variable (if it is large enough), a locator variable, or a file-reference variable.

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

Before the FETCH 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 or structured type 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, LOB , and structured type 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.

The *n*th variable identified by the INTO clause or described in the SQLDA corresponds to the *n*th column of the result table of the cursor. The data type of each variable must be compatible with its corresponding column.

Each assignment to a variable is made according to the rules described in Chapter 3. If the number of variables is less than the number of values in the row, the SQLWARN3 field of the SQLDA is set to 'W'. Note that there is no warning if there are more variables than the number of result columns. If an assignment error occurs, the value is not assigned to the variable, and no more values are assigned to variables. Any values that have already been assigned to variables remain assigned.

Notes

- An open cursor has three possible positions:
  - Before a row
  - On a row
  - After the last row.
- If a cursor is on a row, that row is called the current row of the cursor. A cursor referenced in an UPDATE or DELETE statement must be positioned on a row. A cursor can only be on a row as a result of a FETCH statement.
- When retrieving into LOB locators in situations where it is not necessary to retain the locator across FETCH statements, it is good practice to issue a FREE LOCATOR statement before issuing the next FETCH statement, as locator resources are limited.
- It is possible for an error to occur that makes the state of the cursor unpredictable.
- It is possible that a warning may not be returned on a FETCH. It is also possible that the returned warning applies to a previously fetched row. This occurs as a result of optimizations such as the use of system temporary tables or pushdown operators (see *Administration Guide*).
- Statement caching affects the behavior of an EXECUTE IMMEDIATE statement. See the "Notes" on page 896 for information.
- DB2 CLI supports additional fetching capabilities. For instance when a cursor's result table is read-only, the SQLFetchScroll() function can be used to position the cursor at any spot within that result table.

Examples

Example 1: In this C example, the FETCH statement fetches the results of the SELECT statement into the program variables `dnum`, `dname`, and `mnum`. When no more rows remain to be fetched, the not found condition is returned.

```
EXEC SQL DECLARE C1 CURSOR FOR
  SELECT DEPTNO, DEPTNAME, MGRNO FROM TDEPT
    WHERE ADMRDEPT = 'A00';

EXEC SQL OPEN C1;

while (SQLCODE==0) {
  EXEC SQL FETCH C1 INTO :dnum, :dname, :mnum;
}

EXEC SQL CLOSE C1;
```

Example 2: This FETCH statement uses an SQLDA.

```
FETCH CURS USING DESCRIPTOR :sqlda3
```

FLUSH EVENT MONITOR

The FLUSH EVENT MONITOR statement writes current database monitor values for all active monitor types associated with event monitor *event-monitor-name* to the event monitor I/O target. Hence, at any time a partial event record is available for event monitors that have low record generation frequency (such as a database event monitor). Such records are noted in the event monitor log with a *partial record* identifier.

When an event monitor is flushed, its active internal buffers are written to the event monitor output object.

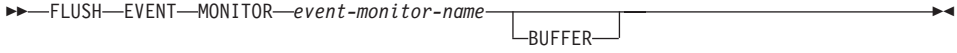
Invocation

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

Authorization

The privileges held by the authorization ID must include either SYSADM or DBADM authority (SQLSTATE 42502).

Syntax



Description

event-monitor-name
Name of the event monitor. This is a one-part name. It is an SQL identifier.

BUFFER
Indicates that the event monitor buffers are to be written out. If BUFFER is specified, then a partial record is not generated. Only the data already present in the event monitor buffers are written out.

Notes

- Flushing out the event monitor will not cause the event monitor values to be reset. This means that the event monitor record that would have been generated if no flush was performed, will still be generated when the normal monitor event is triggered.

FREE LOCATOR

The FREE LOCATOR statement removes the association between a locator variable and its value.

Invocation

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

Authorization

None required.

Syntax

Description

LOCATOR *variable-name,* **...**
Identifies one or more locator variables that must be declared in accordance with the rules for declaring locator variables.

The locator-variable must currently have a locator assigned to it. That is, a locator must have been assigned during this unit of work (by a FETCH statement or a SELECT INTO statement) and must not subsequently have been freed (by a FREE LOCATOR statement); otherwise, an error is raised (SQLSTATE 0F001).

If more than one locator is specified, all locators that can be freed will be freed, regardless of errors detected in other locators in the list.

Example

In a COBOL program, free the BLOB locator variables TKN-VIDEO and TKN-BUF and the CLOB locator variable LIFE-STORY-LOCATOR.

```
EXEC SQL
FREE LOCATOR :TKN-VIDEO, :TKN-BUF, :LIFE-STORY-LOCATOR
END-EXEC.
```

GRANT (Database Authorities)

This form of the GRANT statement grants authorities that apply to the entire database (rather than privileges that apply to specific objects within the database).

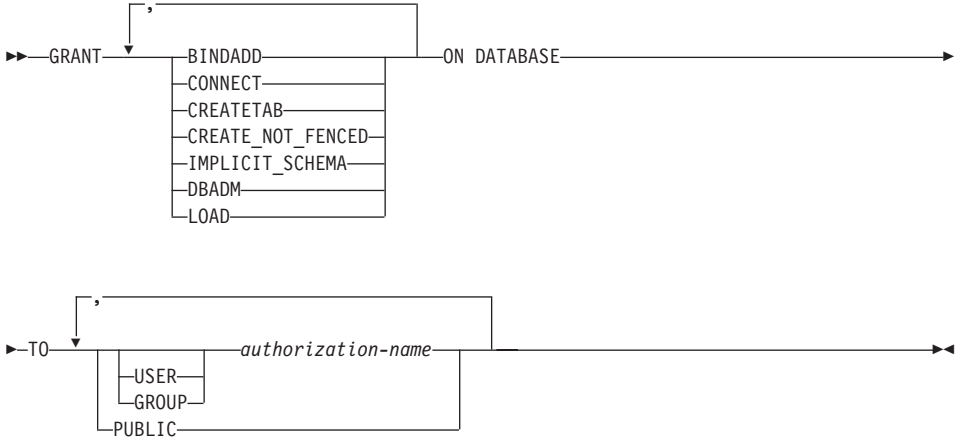
Invocation

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

Authorization

To grant DBADM authority, SYSADM authority is required. To grant other authorities, either DBADM or SYSADM authority is required.

Syntax



Description

BINDADD
Grants the authority to create packages. The creator of a package automatically has the CONTROL privilege on that package and retains this privilege even if the BINDADD authority is subsequently revoked.

CONNECT
Grants the authority to access the database.

CREATETAB
Grants the authority to create base tables. The creator of a base table automatically has the CONTROL privilege on that table. The creator retains this privilege even if the CREATETAB authority is subsequently revoked.

There is no explicit authority required for view creation. A view can be created at any time if the authorization ID of the statement used to create the view has either CONTROL or SELECT privilege on each base table of the view.

CREATE_NOT_FENCED
Grants the authority to register functions that execute in the database manager's process. Care must be taken that functions so registered will not have adverse side effects (see the FENCED or NOT FENCED clause on page 601 for more information).

Once a function has been registered as not fenced, it continues to run in this manner even if CREATE_NOT_FENCED is subsequently revoked.

IMPLICIT_SCHEMA
Grants the authority to implicitly create a schema.

DBADM
Grants the database administrator authority. A database administrator has all privileges against all objects in the database and may grant these privileges to others.

BINDADD, CONNECT, CREATETAB, CREATE_NOT_FENCED and IMPLICIT_SCHEMA are automatically granted to an *authorization-name* that is granted DBADM authority.

LOAD
Grants the authority to load in this database. This authority gives a user the right to use the LOAD utility in this database. SYSADM and DBADM also have this authority by default. However, if a user only has LOAD authority (not SYSADM or DBADM), the user is also required to have table-level privileges. In addition to LOAD privilege, the user is required to have:

- INSERT privilege on the table for LOAD with mode INSERT, TERMINATE (to terminate a previous LOAD INSERT), or RESTART (to restart a previous LOAD INSERT)
- INSERT and DELETE privilege on the table for LOAD with mode REPLACE, TERMINATE (to terminate a previous LOAD REPLACE), or RESTART (to restart a previous LOAD REPLACE)
- INSERT privilege on the exception table, if such a table is used as part of LOAD

TO
Specifies to whom the authorities are granted.

USER
Specifies that the *authorization-name* identifies a user.

GROUP
Specifies that the *authorization-name* identifies a group name.

authorization-name,...
Lists the authorization IDs of one or more users or groups.

The list of authorization IDs cannot include the authorization ID of the user issuing the statement (SQLSTATE 42502).

PUBLIC
Grants the authorities to all users. DBADM cannot be granted to PUBLIC.

Rules

- If neither USER nor GROUP is specified, then
  - If the authorization-name is defined in the operating system only as GROUP, then GROUP is assumed.
  - If the authorization-name is defined in the operating system only as USER or if it is undefined, USER is assumed.
  - If the authorization-name is defined in the operating system as both, or DCE authentication is used, an error (SQLSTATE 56092) is raised.

Examples

Example 1: Give the users WINKEN, BLINKEN, and NOD the authority to connect to the database.

```
GRANT CONNECT ON DATABASE TO USER WINKEN, USER BLINKEN, USER NOD
```

Example 2: GRANT BINDADD authority on the database to a group named D024. There is both a group and a user called D024 in the system.

```
GRANT BINDADD ON DATABASE TO GROUP D024
```

Observe that, the GROUP keyword must be specified; otherwise, an error will occur since both a user and a group named D024 exist. Any member of the D024 group will be allowed to bind packages in the database, but the D024 user will not be allowed (unless this user is also a member of the group D024, had been granted BINDADD authority previously, or BINDADD authority had been granted to another group of which D024 was a member).

GRANT (Index Privileges)

This form of the GRANT statement grants the CONTROL privilege on indexes.

Invocation

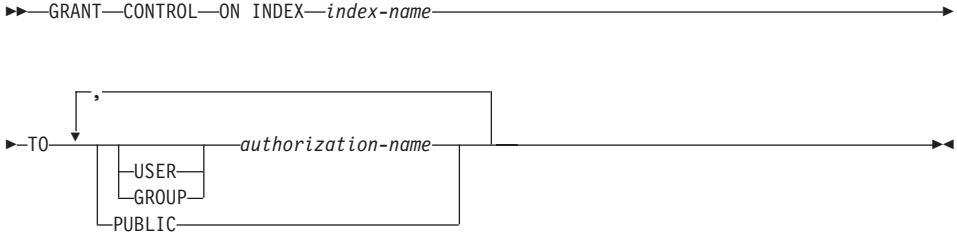
This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared. However, if the bind option DYNAMICRULES BIND applies, the statement cannot be dynamically prepared (SQLSTATE 42509).

Authorization

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

- DBADM authority
- SYSADM authority.

Syntax



Description

CONTROL
Grants the privilege to drop the index. This is the CONTROL authority for indexes, which is automatically granted to creators of indexes.

ON INDEX *index-name*
Identifies the index for which the CONTROL privilege is to be granted.

TO
Specifies to whom the privileges are granted.

USER
Specifies that the *authorization-name* identifies a user.

GROUP
Specifies that the *authorization-name* identifies a group name.

authorization-name,...
Lists the authorization IDs of one or more users or groups.

The list of authorization IDs cannot include the authorization ID of the user issuing the statement (SQLSTATE 42502).

PUBLIC
Grants the privileges to all users.

Rules

- If neither USER nor GROUP is specified, then
  - If the authorization-name is defined in the operating system only as GROUP, then GROUP is assumed.
  - If the authorization-name is defined in the operating system only as USER or if it is undefined, USER is assumed.
  - If the authorization-name is defined in the operating system as both, or DCE authentication is used, an error (SQLSTATE 56092) is raised.

Example

```
GRANT CONTROL ON INDEX DEPTIDX TO USER USER4
```

GRANT (Package Privileges)

This form of the GRANT statement grants privileges on a package.

Invocation

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

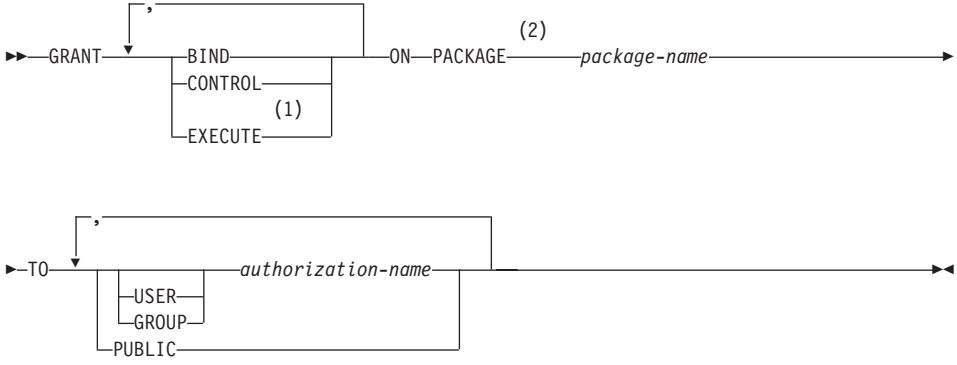
Authorization

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 grant the CONTROL privilege, SYSADM or DBADM authority is required.

Syntax



Notes:

1 RUN can be used as a synonym for EXECUTE.

2 PROGRAM can be used as a synonym for PACKAGE.

Description

BIND
Grants the privilege to bind a package. The BIND privilege is really a rebind privilege, because the package must have already been bound (by someone with BINDADD authority) to have existed at all.

In addition to the BIND privilege, the user must hold the necessary privileges on each table referenced by static DML statements contained in the program. This is necessary because authorization on static DML statements is checked at bind time.

CONTROL
Grants the privilege to rebind, drop, or execute the package, and extend package privileges to other users. The CONTROL privilege for packages is automatically granted to creators of packages. A package owner is the package binder, or the ID specified with the OWNER option at bind/precompile time.

BIND and EXECUTE are automatically granted to an *authorization-name* that is granted CONTROL privilege.

EXECUTE
Grants the privilege to execute the package.

ON PACKAGE *package-name*
Specifies the name of the package on which privileges are to be granted.

TO
Specifies to whom the privileges are granted.

USER
Specifies that the *authorization-name* identifies a user.

GROUP
Specifies that the *authorization-name* identifies a group name.

authorization-name,...
Lists the authorization IDs of one or more users or groups.

The list of authorization IDs cannot include the authorization ID of the user issuing the statement (SQLSTATE 42502).

PUBLIC
Grants the privileges to all users.

Rules

- If neither USER nor GROUP is specified, then
  - If the *authorization-name* is defined in the operating system only as GROUP, then GROUP is assumed.
  - If the *authorization-name* is defined in the operating system only as USER or if it is undefined, USER is assumed.
  - If the *authorization-name* is defined in the operating system as both, or DCE authentication is used, an error (SQLSTATE 56092) is raised.

Examples

Example 1: Grant the EXECUTE privilege on PACKAGE CORPDATA.PKGA to PUBLIC.

```
GRANT EXECUTE
  ON PACKAGE CORPDATA.PKGA
  TO PUBLIC
```

Example 2: GRANT EXECUTE privilege on package CORPDATA.PKGA to a user named EMPLOYEE. There is neither a group nor a user called EMPLOYEE.

```
GRANT EXECUTE ON PACKAGE
   CORPDATA.PKGA TO EMPLOYEE
```

or

```
GRANT EXECUTE ON PACKAGE
   CORPDATA.PKGA TO USER EMPLOYEE
```

GRANT (Schema Privileges)

This form of the GRANT statement grants privileges on a schema.

Invocation

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

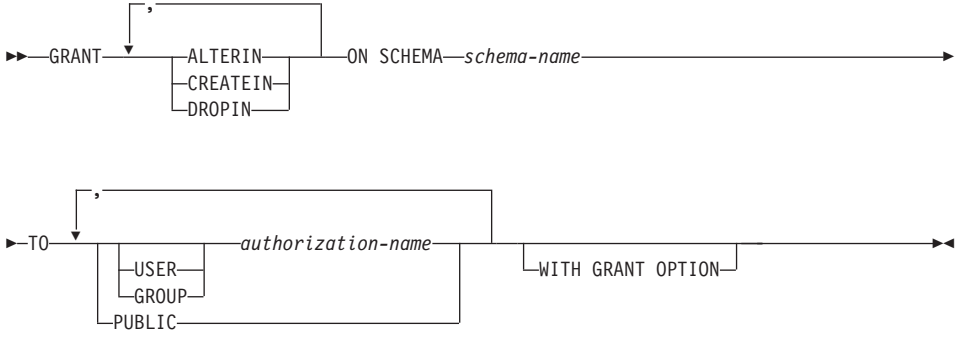
Authorization

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

- WITH GRANT OPTION for each identified privilege on *schema-name*
- SYSADM or DBADM authority

Privileges cannot be granted on schema names SYSIBM, SYSCAT, SYSFUN and SYSSTAT by any user.

Syntax



Description

ALTERIN
Grants the privilege to alter or comment on all objects in the schema. The owner of an explicitly created schema automatically receives ALTERIN privilege.

CREATEIN
Grants the privilege to create objects in the schema. Other authorities or privileges required to create the object (such as CREATETAB) are still required. The owner of an explicitly created schema automatically receives CREATEIN privilege. An implicitly created schema has CREATEIN privilege automatically granted to PUBLIC.

DROPIN
Grants the privilege to drop all objects in the schema. The owner of an explicitly created schema automatically receives DROPIN privilege.

ON SCHEMA *schema-name*
Identifies the schema on which the privileges are to be granted.

TO
Specifies to whom the privileges are granted.

USER
Specifies that the *authorization-name* identifies a user.

GROUP
Specifies that the *authorization-name* identifies a group name.

authorization-name,...
Lists the authorization IDs of one or more users or groups.

The list of authorization IDs cannot include the authorization ID of the user issuing the statement (SQLSTATE 42502).

PUBLIC
Grants the privileges to all users.

WITH GRANT OPTION
Allows the specified *authorization-name*s to GRANT the privileges to others.

If the WITH GRANT OPTION is omitted, the specified *authorization-name*s can only grant the privileges to others if they:

- have DBADM authority or
- received the ability to grant privileges from some other source.

Rules

- If neither USER nor GROUP is specified, then
  - If the authorization-name is defined in the operating system only as GROUP, then GROUP is assumed.
  - If the authorization-name is defined in the operating system only as USER or if it is undefined, USER is assumed.
  - If the authorization-name is defined in the operating system as both, or DCE authentication is used, an error (SQLSTATE 56092) is raised.
- In general, the GRANT statement will process the granting of privileges that the authorization ID of the statement is allowed to grant, returning a warning (SQLSTATE 01007) if one or more privileges was not granted. If no privileges were granted, an error is returned (SQLSTATE 42501).[98]

98. If the package used for processing the statement was precompiled with LANGLEVEL set to SQL92E for MIA, a warning is returned (SQLSTATE 01007) unless the grantor has NO privileges on the object of the grant.

Examples

Example 1: Grant USER2 to the ability to create objects in schema CORPDATA.

```
GRANT CREATEIN ON SCHEMA CORPDATA TO USER2
```

Example 2: Grant user BIGGUY the ability to create and drop objects in schema CORPDATA.

```
GRANT CREATEIN, DROPIN ON SCHEMA CORPDATA TO BIGGUY
```

GRANT (Server Privileges)

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

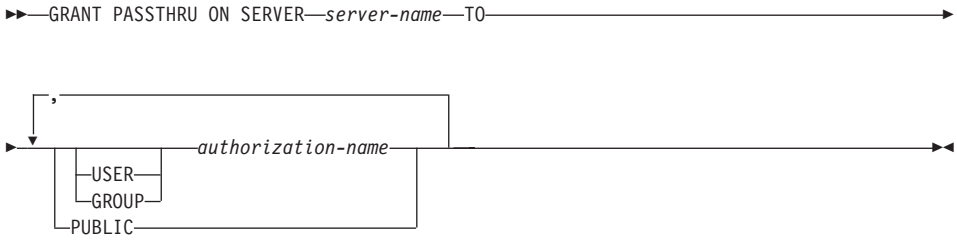
Invocation

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

Syntax



Description

server-name
Names the data source for which the privilege to use in pass-through mode is being granted. *server-name* must identify a data source that is described in the catalog.

TO
Specifies to whom the privilege is granted.

USER
Specifies that the *authorization-name* identifies a user.

GROUP
Specifies that the *authorization-name* identifies a group name.

authorization-name,...
Lists the authorization IDs of one or more users or groups.

The list of authorization IDs cannot include the authorization ID of the user issuing the statement (SQLSTATE 42502).

PUBLIC
Grants to all users the privilege to pass through to *server-name.*

Examples

Example 1: Give R. Smith and J. Jones the privilege to pass through to data source SERVALL. Their authorization IDs are RSMITH and JJONES.

```
GRANT PASSTHRU ON SERVER SERVALL
   TO USER RSMITH,
   USER JJONES
```

Example 2: Grant the privilege to pass through to data source EASTWING to a group whose authorization ID is D024. There is a user whose authorization ID is also D024.

```
GRANT PASSTHRU ON SERVER EASTWING TO GROUP D024
```

The GROUP keyword must be specified; otherwise, an error will occur because D024 is a user's ID as well as the specified group's ID (SQLSTATE 56092). Any member of group D024 will be allowed to pass through to EASTWING. Therefore, if user D024 belongs to the group, this user will be able to pass through to EASTWING.

GRANT (Table, View, or Nickname Privileges)

This form of the GRANT statement grants privileges on a table, view, or nickname.

Invocation

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

Authorization

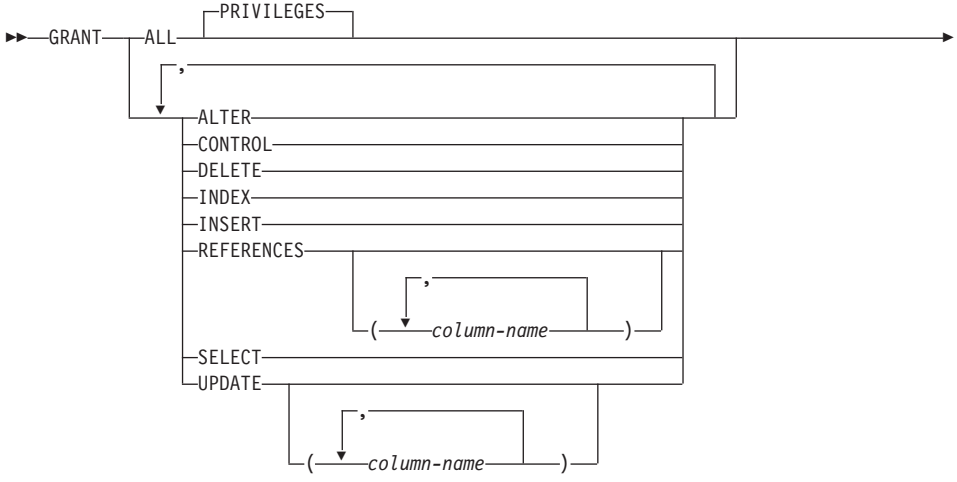
The privileges held by the authorization ID of the statement must include at least one of the following:

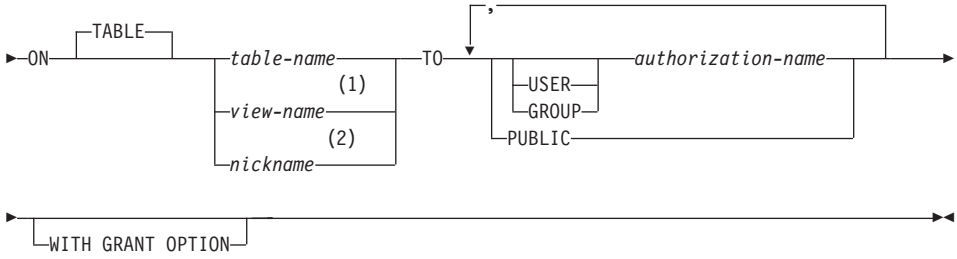
- CONTROL privilege on the referenced table, view, or nickname
- The WITH GRANT OPTION for each identified privilege. If ALL is specified, the authorization ID must have some grantable privilege on the identified table, view, or nickname.
- SYSADM or DBADM authority.

To grant the CONTROL privilege, SYSADM or DBADM authority is required.

To grant privileges on catalog tables and views, either SYSADM or DBADM authority is required.

Syntax





Notes:

1 ALTER, INDEX, and REFERENCES privileges are not applicable to views.

2 DELETE, INSERT, SELECT, and UPDATE privileges are not applicable to nicknames.

Description

ALL or **ALL PRIVILEGES**
Grants all the appropriate privileges, except CONTROL, on the base table, view, or nickname named in the ON clause.

If the authorization ID of the statement has CONTROL privilege on the table, view, or nickname, or DBADM or SYSADM authority, then all the privileges applicable to the object (except CONTROL) are granted. Otherwise, the privileges granted are all those grantable privileges that the authorization ID of the statement has on the identified table, view, or nickname.

If ALL is not specified, one or more of the keywords in the list of privileges must be specified.

ALTER
Grants the privilege to:

- Add columns to a base table definition.
- Create or drop a primary key or unique constraint on a base table. For more information on the authorization required to create or drop a primary key or a unique constraint, see "ALTER TABLE" on page 477.
- Create or drop a foreign key on a base table.

  The REFERENCES privilege on each column of the parent table is also required.
- Create or drop a check constraint on a base table.
- Create a trigger on a base table.
- Add, reset, or drop a column option for a nickname.
- Change a nickname column name or data type.

- Add or change a comment on a base table, a view, or a nickname.

CONTROL
Grants:

- All of the appropriate privileges in the list, that is:
  - ALTER, CONTROL, DELETE, INSERT, INDEX, REFERENCES, SELECT, and UPDATE to base tables
  - CONTROL, DELETE, INSERT, SELECT, and UPDATE to views
  - ALTER, CONTROL, INDEX, and REFERENCES to nicknames
- The ability to grant the above privileges (except for CONTROL) to others.
- The ability to drop the base table, view, or nickname.

  This ability cannot be extended to others on the basis of holding CONTROL privilege. The only way that it can be extended is by granting the CONTROL privilege itself and that can only be done by someone with SYSADM or DBADM authority.
- The ability to execute the RUNSTATS utility on the table and indexes. See the *Command Reference* for information on RUNSTATS.
- The ability to issue SET INTEGRITY statement on the base table or summary table.

The definer of a base table, summary table, or nickname automatically receives the CONTROL privilege.

The definer of a view automatically receives the CONTROL privilege if the definer holds the CONTROL privilege on all tables, views, and nicknames identified in the fullselect.

DELETE
Grants the privilege to delete rows from the table or updatable view.

INDEX
Grants the privilege to create an index on a table, or an index specification on a nickname. The creator of an index or index specification automatically has the CONTROL privilege on the index or index specification (authorizing the creator to drop the index or index specification). In addition, the creator retains the CONTROL privilege even if the INDEX privilege is revoked.

INSERT
Grants the privilege to insert rows into the table or updatable view and to run the IMPORT utility.

REFERENCES
Grants the privilege to create and drop a foreign key referencing the table as the parent.

If the authorization ID of the statement has one of:

- DBADM or SYSADM authority
- CONTROL privilege on the table
- REFERENCES WITH GRANT OPTION on the table

then the grantee(s) can create referential constraints using all columns of the table as parent key, even those added later using the ALTER TABLE statement. Otherwise, the privileges granted are all those grantable column REFERENCES privileges that the authorization ID of the statement has on the identified table. For more information on the authorization required to create or drop a foreign key, see "ALTER TABLE" on page 477.

The privilege can be granted on a nickname although foreign keys cannot be defined to reference nicknames.

REFERENCES (*column-name***,...)**
Grants the privilege to create and drop a foreign key using only those columns specified in the column list as a parent key. Each *column-name* must be an unqualified name that identifies a column of the table identified in the ON clause. Column level REFERENCES privilege cannot be granted on typed tables, typed views, or nicknames (SQLSTATE 42997).

SELECT
Grants the privilege to:

- Retrieve rows from the table or view.
- Create views on the table.
- Run the EXPORT utility against the table or view. See the *Command Reference* for information on EXPORT.

UPDATE
Grants the privilege to use the UPDATE statement on the table or updatable view identified in the ON clause.

If the authorization ID of the statement has one of:

- DBADM or SYSADM authority
- CONTROL privilege on the table or view
- UPDATE WITH GRANT OPTION on the table or view

then the grantee(s) can update all updatable columns of the table or view on which the grantor has with grant privilege as well as those columns added later using the ALTER TABLE statement. Otherwise, the privileges granted are all those grantable column UPDATE privileges that the authorization ID of the statement has on the identified table or view.

UPDATE (*column-name***,...)**
Grants the privilege to use the UPDATE statement to update only those

columns specified in the column list. Each *column-name* must be an unqualified name that identifies a column of the table or view identified in the ON clause. Column level UPDATE privilege cannot be granted on typed tables, typed views, or nicknames (SQLSTATE 42997).

ON TABLE *table-name* or *view-name* or *nickname*
Specifies the table, view, or nickname on which privileges are to be granted.

No privileges may be granted on an inoperative view or an inoperative summary table (SQLSTATE 51024). No privileges may be granted on a declared temporary table (SQLSTATE 42995).

TO
Specifies to whom the privileges are granted.

USER
Specifies that the *authorization-name* identifies a user.

GROUP
Specifies that the *authorization-name* identifies a group name.

authorization-name,...
Lists the authorization IDs of one or more users or groups.[99]

A privilege granted to a group is not used for authorization checking on static DML statements in a package. Nor is it used when checking authorization on a base table while processing a CREATE VIEW statement.

In DB2 Universal Database, table privileges granted to groups only apply to statements that are dynamically prepared. For example, if the INSERT privilege on the PROJECT table has been granted to group D204 but not UBIQUITY (a member of D204) UBIQUITY could issue the statement:

```
EXEC SQL EXECUTE IMMEDIATE :INSERT_STRING;
```

where the content of the string is:

```
INSERT INTO PROJECT (PROJNO, PROJNAME, DEPTNO, RESPEMP)
VALUES ('AD3114', 'TOOL PROGRAMMING', 'D21', '000260');
```

but could not precompile or bind a program with the statement:

```
EXEC SQL INSERT INTO PROJECT (PROJNO, PROJNAME, DEPTNO, RESPEMP)
VALUES ('AD3114', 'TOOL PROGRAMMING', 'D21', '000260');
```

99. Restrictions in previous versions on grants to authorization ID of the user issuing the statement have been removed.

PUBLIC
Grants the privileges to all users.[100]

WITH GRANT OPTION
Allows the specified *authorization-name*s to GRANT the privileges to others.

If the specified privileges include CONTROL, the WITH GRANT OPTION applies to all the applicable privileges except for CONTROL (SQLSTATE 01516).

Rules

- If neither USER nor GROUP is specified, then
  - If the *authorization-name* is defined in the operating system only as GROUP, then GROUP is assumed.
  - If the *authorization-name* is defined in the operating system only as USER or if it is undefined, USER is assumed.
  - If the *authorization-name* is defined in the operating system as both, or DCE authentication is used, an error (SQLSTATE 56092) is raised.
- In general, the GRANT statement will process the granting of privileges that the authorization ID of the statement is allowed to grant, returning a warning (SQLSTATE 01007) if one or more privileges was not granted. If no privileges were granted, an error is returned (SQLSTATE 42501).[101] If CONTROL privilege is specified, privileges will only be granted if the authorization ID of the statement has SYSADM or DBADM authority (SQLSTATE 42501).

Notes

- Privileges may be granted independently at every level of a table hierarchy. A user with a privilege on a supertable may affect the subtables. For example, an update specifying the supertable *T* may show up as a change to a row in the subtable *S* of *T* done by a user with UPDATE privilege on *T* but without UPDATE privilege on *S*. A user can only operate directly on the subtable if the necessary privilege is held on the subtable.
- Granting nickname privileges has no effect on data source object (table or view) privileges. Typically, data source privileges are required for the table or view that a nickname references when attempting to retrieve data.
- DELETE, INSERT, SELECT, and UPDATE privileges are not defined for nicknames since operations on nicknames depend on the privileges of the authorization ID used at the data source when the statement referencing the nickname is processed.

---

100. Restrictions in previous versions on the use of privileges granted to PUBLIC for static SQL statements and CREATE VIEW statements have been removed.

101. If the package used for processing the statement was precompiled with LANGLEVEL set to SQL92E or MIA, a warning is returned (SQLSTATE 01007) unless the grantor has NO privileges on the object of the grant.

Examples

Example 1: Grant all privileges on the table WESTERN_CR to PUBLIC.

```
GRANT ALL ON WESTERN_CR
 TO PUBLIC
```

Example 2: Grant the appropriate privileges on the CALENDAR table so that users PHIL and CLAIRE can read it and insert new entries into it. Do not allow them to change or remove any existing entries.

```
GRANT SELECT, INSERT ON CALENDAR
 TO USER  PHIL, USER CLAIRE
```

Example 3: Grant all privileges on the COUNCIL table to user FRANK and the ability to extend all privileges to others.

```
GRANT ALL ON COUNCIL
 TO USER FRANK WITH GRANT OPTION
```

Example 4: GRANT SELECT privilege on table CORPDATA.EMPLOYEE to a user named JOHN. There is a user called JOHN and no group called JOHN.

```
GRANT SELECT ON CORPDATA.EMPLOYEE TO JOHN
```

or

```
GRANT SELECT
  ON CORPDATA.EMPLOYEE TO USER JOHN
```

Example 5: GRANT SELECT privilege on table CORPDATA.EMPLOYEE to a group named JOHN. There is a group called JOHN and no user called JOHN.

```
GRANT SELECT ON CORPDATA.EMPLOYEE TO JOHN
```

or

```
GRANT SELECT ON CORPDATA.EMPLOYEE TO GROUP JOHN
```

Example 6: GRANT INSERT and SELECT on table T1 to both a group named D024 and a user named D024.

```
GRANT INSERT, SELECT ON TABLE T1
   TO GROUP D024, USER D024
```

In this case, both the members of the D024 group and the user D024 would be allowed to INSERT into and SELECT from the table T1. Also, there would be two rows added to the SYSCAT.TABAUTH catalog view.

Example 7: GRANT INSERT, SELECT, and CONTROL on the CALENDAR table to user FRANK. FRANK must be able to pass the privileges on to others.

```
GRANT CONTROL ON TABLE CALENDAR
   TO FRANK WITH GRANT OPTION
```

The result of this statement is a warning (SQLSTATE 01516) that CONTROL was not given the WITH GRANT OPTION. Frank now has the ability to grant any privilege on CALENDAR including INSERT and SELECT as required. FRANK cannot grant CONTROL on CALENDAR to other users unless he has SYSADM or DBADM authority.

Example 8: User JON created a nickname for an Oracle table that had no index. The nickname is ORAREM1. Later, the Oracle DBA defined an index for this table. User SHAWN now wants DB2 to know that this index exists, so that the optimizer can devise strategies to access the table more efficiently. SHAWN can inform DB2 of the index by creating an index specification for ORAREM1. Give SHAWN the index privilege on this nickname, so that he can create the index specification.

```
GRANT INDEX ON NICKNAME ORAREM1
   TO USER SHAWN
```

GRANT (Table Space Privileges)

This form of the GRANT statement grants privileges on a table space.

Invocation

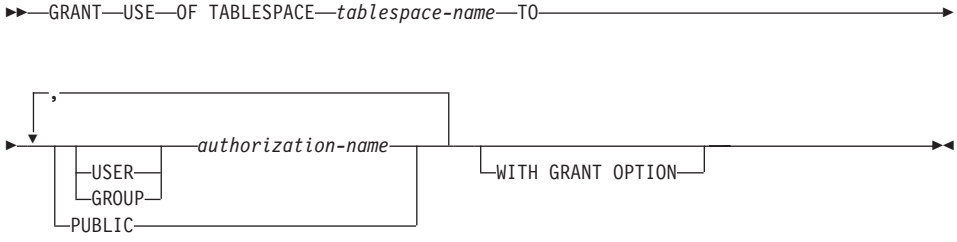
This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared. However, if the bind option DYNAMICRULES BIND applies, the statement cannot be dynamically prepared (SQLSTATE 42509).

Authorization

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

- WITH GRANT OPTION for use of the table space
- SYSADM, SYSCTRL, or DBADM authority

Syntax



Description

USE
Grants the privilege to specify or default to the table space when creating a table. The creator of a table space automatically receives USE privilege with grant option.

OF TABLESPACE *tablespace-name*
Identifies the table space on which the USE privilege is to be granted. The table space cannot be SYSCATSPACE (SQLSTATE 42838) or a system temporary table space (SQLSTATE 42809).

TO
Specifies to whom the USE privilege is granted.

USER
Specifies that the *authorization-name* identifies a user.

GROUP
Specifies that the *authorization-name* identifies a group name.

authorization-name
Lists the authorization IDs of one or more users or groups.

The list of authorization IDs cannot include the authorization ID of the user issuing the statement (SQLSTATE 42502).

PUBLIC
Grants the USE privilege to all users.

WITH GRANT OPTION
Allows the specified *authorization-name* to GRANT the USE privilege to others.

If the WITH GRANT OPTION is omitted, the specified *authorization-name* can only GRANT the USE privilege to others if they:

- have SYSADM or DBADM authority or
- received the ability to GRANT the USE privilege from some other source.

Notes

If neither USER nor GROUP is specified, then

- If the *authorization-name* is defined in the operating system only as GROUP, then GROUP is assumed.
- If the *authorization-name* is defined in the operating system only as USER, or if it is undefined, then USER is assumed.
- If the *authorization-name* is defined in the operating system as both, or DCE authentication is used, an error is returned (SQLSTATE 56092).

Examples

Example 1: Grant user BOBBY the ability to create tables in table space PLANS and to grant this privilege to others.

```
GRANT USE OF TABLESPACE PLANS TO BOBBY WITH GRANT OPTION
```

INCLUDE

The INCLUDE statement inserts declarations into a source program.

Invocation

This statement can only be embedded in an application program. It is not an executable statement.

Authorization

None required.

Syntax

Description

SQLCA
Indicates the description of an SQL communication area (SQLCA) is to be included. For a description of the SQLCA, see "Appendix B. SQL Communications (SQLCA)" on page 1107.

SQLDA
Indicates the description of an SQL descriptor area (SQLDA) is to be included. For a description of the SQLDA, see "Appendix C. SQL Descriptor Area (SQLDA)" on page 1113.

name
Identifies an external file containing text that is to be included in the source program being precompiled. It may be an SQL identifier without a filename extension or a literal in single quotes (' '). An SQL identifier assumes the filename extension of the source file being precompiled. If a filename extension is not provided by a literal in quotes then none is assumed.

For host language specific information, see the *Application Development Guide*.

Notes

- When a program is precompiled, the INCLUDE statement is replaced by source statements. Thus, the INCLUDE statement should be specified at a point in the program such that the resulting source statements are acceptable to the compiler.
- The external source file must be written in the host language specified by the *name*. If it is greater than 18 characters or contains characters not allowed in an SQL identifier then it must be in single quotes. INCLUDE *name* statements may be nested though not cyclical (for example, if A and B

are modules and A contains an INCLUDE *name* statement, then it is not valid for A to call B and then B to call A).

- When the LANGLEVEL precompile option is specified with the SQL92E value, INCLUDE SQLCA should not be specified. SQLSTATE and SQLCODE variables may be defined within the host variable declare section.

Example

Include an SQLCA in a C program.

```
EXEC SQL INCLUDE SQLCA;

EXEC SQL DECLARE C1 CURSOR FOR
  SELECT DEPTNO, DEPTNAME, MGRNO FROM TDEPT
    WHERE ADMRDEPT = 'A00';

EXEC SQL OPEN C1;

while (SQLCODE==0) {
  EXEC SQL FETCH C1 INTO :dnum, :dname, :mnum;

(Print results)

}

EXEC SQL CLOSE C1;
```

INSERT

The INSERT statement inserts rows into a table or view. Inserting a row into a view also inserts the row into the table on which the view is based.

Invocation

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

Authorization

To execute this statement, the privileges held by the authorization ID of the statement must include at least one of the following:

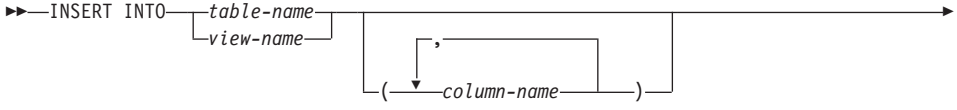
- INSERT privilege on the table or view where rows are to be inserted
- CONTROL privilege on the table or view where rows are to be inserted
- SYSADM or DBADM authority.

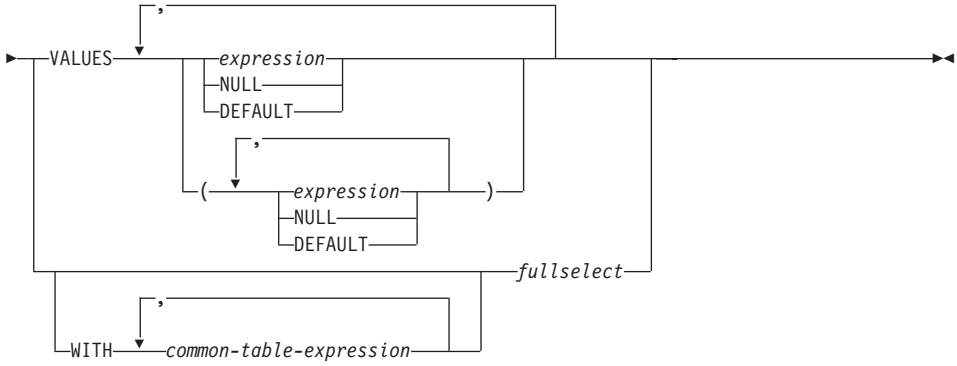
In addition, for each table or view referenced in any fullselect used in the INSERT statement, the privileges held by the authorization ID of the statement must include at least one of the following:

- SELECT privilege
- CONTROL privilege
- SYSADM or DBADM authority.

GROUP privileges are not checked for static INSERT statements.

Syntax





Note: See "Chapter 5. Queries" on page 393 for the syntax of *common-table-expression* and *fullselect*.

Description

INTO *table-name* or *view-name*
Identifies the object of the insert operation. The name must identify a table or view that exists at the application server, but it must not identify a catalog table, a summary table, a view of a catalog table, or a read-only view.

A value cannot be inserted into a view column that is derived from:

- A constant, expression, or scalar function
- The same base table column as some other column of the view
- A column derived from a nickname.

If the object of the insert operation is a view with such columns, a list of column names must be specified, and the list must not identify these columns.

(*column-name***,...)**
Specifies the columns for which insert values are provided. Each name must be an unqualified name that identifies a column of the table or view. The same column must not be identified more than once. A view column that cannot accept insert values must not be identified.

Omission of the column list is an implicit specification of a list in which every column of the table or view is identified in left-to-right order. This list is established when the statement is prepared and therefore does not include columns that were added to a table after the statement was prepared.

The implicit column list is established at prepare time. Hence an INSERT statement embedded in an application program does not use any columns that might have been added to the table or view after prepare time.

VALUES
Introduces one or more rows of values to be inserted.

Each host variable named must be described in the program in accordance with the rules for declaring host variables.

The number of values for each row must equal the number of names in the column list. The first value is inserted in the first column in the list, the second value in the second column, and so on.

expression
An *expression* can be as defined in "Expressions" on page 157.

NULL
Specifies the null value and should only be specified for nullable columns.

DEFAULT
Specifies that the default value is to be used. The result of specifying DEFAULT depends on how the column was defined, as follows:

- If the column was defined as a generated column based on an expression, the column value is generated by the system, based on that expression.
- If the IDENTITY clause is used, the value is generated by the database manager.
- If the WITH DEFAULT clause is used, the value inserted is as defined for the column (see *default-clause* in "CREATE TABLE" on page 712).
- If the WITH DEFAULT clause, GENERATED clause, and the NOT NULL clause are not used, the value inserted is NULL.
- If the NOT NULL clause is used and the GENERATED clause is not used, or the WITH DEFAULT clause is not used or DEFAULT NULL is used, the DEFAULT keyword cannot be specified for that column (SQLSTATE 23502).

WITH *common-table-expression*
Defines a common table expression for use with the fullselect that follows. See "common-table-expression" on page 440 for an explanation of the *common-table-expression*.

fullselect
Specifies a set of new rows in the form of the result table of a fullselect. There may be one, more than one, or none. If the result table is empty, SQLCODE is set to +100 and SQLSTATE is set to '02000'.

When the base object of the INSERT and the base object of the fullselect or any subquery of the fullselect, are the same table, the fullselect is completely evaluated before any rows are inserted.

The number of columns in the result table must equal the number of names in the column list. The value of the first column of the result is inserted in the first column in the list, the second value in the second column, and so on.

Rules

- ***Default values:*** The value inserted in any column that is not in the column list is either the default value of the column or null. Columns that do not allow null values and are not defined with NOT NULL WITH DEFAULT must be included in the column list. Similarly, if you insert into a view, the value inserted into any column of the base table that is not in the view is either the default value of the column or null. Hence, all columns of the base table that are not in the view must have either a default value or allow null values. The only value that can be inserted into a generated column defined with the GENERATED ALWAYS clause is DEFAULT (SQLSTATE 428C9).
- ***Length:*** If the insert value of a column is a number, the column must be a numeric column with the capacity to represent the integral part of the number. If the insert value of a column is a string, the column must either be a string column with a length attribute at least as great as the length of the string, or a datetime column if the string represents a date, time, or timestamp.
- ***Assignment:*** Insert values are assigned to columns in accordance with the assignment rules described in Chapter 3.
- ***Validity:*** If the table named, or the base table of the view named, has one or more unique indexes, each row inserted into the table must conform to the constraints imposed by those indexes. If a view whose definition includes WITH CHECK OPTION is named, each row inserted into the view must conform to the definition of the view. For an explanation of the rules governing this situation, see "CREATE VIEW" on page 823.
- ***Referential Integrity:*** For each constraint defined on a table, each non-null insert value of the foreign key must be equal to a primary key value of the parent table.
- ***Check Constraint:*** Insert values must satisfy the check conditions of the check constraints defined on the table. An INSERT to a table with check constraints defined has the constraint conditions evaluated once for each row that is inserted.
- ***Triggers:*** Insert statements may cause triggers to be executed. A trigger may cause other statements to be executed or may raise error conditions based on the insert values.

- *Datalinks:* Insert statements that include DATALINK values will result in an attempt to link the file if a URL value is included (not empty string or blanks) and the column is defined with FILE LINK CONTROL. Errors in the DATALINK value or in linking the file will cause the insert to fail (SQLSTATE 428D1 or 57050).

Notes

- After execution of an INSERT statement that is embedded within a program, the value of the third variable of the SQLERRD(3) portion of the SQLCA indicates the number of rows that were inserted. SQLERRD(5) contains the count of all triggered insert, update and delete operations.
- Unless appropriate locks already exist, one or more exclusive locks are acquired at the execution of a successful INSERT statement. Until the locks are released, an inserted row can only be accessed by:
  - The application process that performed the insert.
  - Another application process using isolation level UR through a read-only cursor, SELECT INTO statement, or subselect used in a subquery.
- For further information about locking, see the description of the COMMIT, ROLLBACK, and LOCK TABLE statements.
- If an application is running against a partitioned database, and it is bound with option INSERT BUF, then INSERT with VALUES statements which are not processed using EXECUTE IMMEDIATE may be buffered. DB2 assumes that such an INSERT statement is being processed inside a loop in the application's logic. Rather than execute the statement to completion, it attempts to buffer the new row values in one or more buffers. As a result the actual insertions of the rows into the table are performed later, asynchronous with the application's INSERT logic. Be aware that this asynchronous insertion may cause an error related to an INSERT to be returned on some other SQL statement that follows the INSERT in the application.

  This has the potential to dramatically improve INSERT performance, but is best used with clean data, due to the asynchronous nature of the error handling. See buffered insert in the *Application Development Guide* for further details.
- When a row is inserted into a table that has an identity column, DB2 generates a value for the identity column.
  - For a GENERATED ALWAYS identity column, DB2 always generates the value.
  - For a GENERATED BY DEFAULT column, if a value is not explicitly specified (with a VALUES clause, or subselect), DB2 generates a value.

  The first value generated by DB2 is the value of the START WITH specification for the identity column.

- When a value is inserted for a user-defined distinct type identity column, the entire computation is done in the source type, and the result is cast to the distinct type before the value is actually assigned to the column.[102]
- When inserting into a GENERATED ALWAYS identity column, DB2 will always generate a value for the column, and users must not specify a value at insertion time. If a GENERATED ALWAYS identity column is listed in the column-list of the INSERT statement, with a non-DEFAULT value in the VALUES clause, an error occurs (SQLSTATE 428C9).

  For example, assuming that EMPID is defined as an identity column that is GENERATED ALWAYS, then the command:

  ```
  INSERT INTO T2 (EMPID, EMPNAME, EMPADDR)
    VALUES (:hv_valid_emp_id, :hv_name, :hv_addr)
  ```

  will result in an error.
- When inserting into a GENERATED BY DEFAULT column, DB2 will allow an actual value for the column to be specified within the VALUES clause, or from a subselect. However, when a value is specified in the VALUES clause, DB2 does not perform any verification of the value. In order to guarantee uniqueness of the values, a unique index on the identity column must be created.

  When inserting into a table with a GENERATED BY DEFAULT identity column, without specifying a column list, the VALUES clause can specify the DEFAULT keyword to represent the value for the identity column. DB2 will generate the value for the identity column.

  ```
  INSERT INTO T2 (EMPID, EMPNAME, EMPADDR)
    VALUES (DEFAULT, :hv_name, :hv_addr)
  ```

  In this example, EMPID is defined as an identity column, and thus the value inserted into this column is generated by DB2.
- The rules for inserting into an identity column with a subselect are similar to those for an insert with a VALUES clause. A value for an identity column may only be specified if the identity column is defined as GENERATED BY DEFAULT.

  For example, assume T1 and T2 are tables with the same definition, both containing columns *intcol1* and *identcol2* (both are type INTEGER and the second column has the identity attribute). Consider the following insert:

  ```
  INSERT INTO T2
     SELECT *
     FROM T1
  ```

  This example is logically equivalent to:

102. There is no casting of the previous value to the source type prior to the computation.

```
INSERT INTO T2 (intcol1,identcol2)
   SELECT intcol1, identcol2
   FROM T1
```

In both cases, the INSERT statement is providing an explicit value for the identity column of T2. This explicit specification can be given a value for the identity column, but the identity column in T2 must be defined as GENERATED BY DEFAULT. Otherwise, an error will result (SQLSTATE 428C9).

If there is a table with a column defined as a GENERATED ALWAYS identity, it is still possible to propagate all other columns from a table with the same definition. For example, given the example tables T1 and T2 described above, the intcol1 values from T1 to T2 can be propagated with the following SQL:

```
INSERT INTO T2 (intcol1)
   SELECT intcol1
   FROM T1
```

Note that, because identcol2 is not specified in the column-list, it will be filled in with its default (generated) value.

- When inserting a row into a single column table where the column is defined as a GENERATED ALWAYS identity column, it is possible to specify a VALUES clause with the DEFAULT keyword. In this case, the application does not provide any value for the table, and DB2 generates the value for the identity column.

  ```
  INSERT INTO IDTABLE
     VALUES(DEFAULT)
  ```

  Assuming the same single column table for which the column has the identity attribute, to insert multiple rows with a single INSERT statement, the following INSERT statement could be used:

  ```
  INSERT INTO IDTABLE
     VALUES (DEFAULT), (DEFAULT), (DEFAULT), (DEFAULT)
  ```

- When DB2 generates a value for an identity column, that generated value is consumed; the next time that a value is needed, DB2 will generate a new value. This is true even when an INSERT statement involving an identity column fails or is rolled back.

  For example, assume that a unique index has been created on the identity column. If a duplicate key violation is detected in generating a value for an identity column, an error occurs (SQLSTATE 23505) and the value generated for the identity column is considered to be consumed. This can occur when the identity column is defined as GENERATED BY DEFAULT and the system tries to generate a new value, but the user has explicitly specified values for the identity column in previous INSERT statements. Reissuing the same INSERT statement in this case can lead to success. DB2

will generate the next value for the identity column, and it is possible that this next value will be unique, and that this INSERT statement will be successful.

- If the maximum value for the identity column is exceeded (or minimum value for a descending sequence) in generating a value for an identity column, an error occurs (SQLSTATE 23522). In this situation, the user would have to DROP and CREATE a new table with an identity column having a larger range (that is, change the data type or increment value for the column to allow for a larger range of values).

  For example, an identity column may have been defined with a data type of SMALLINT, and eventually the column runs out of assignable values. To redefine the identity column as INTEGER, the data would need to be unloaded, the table would have to be dropped and recreated with a new definition for the column, and then the data would be reloaded. When the table is redefined, it needs to specify a START WITH value for the identity column such that the next value generated by DB2 will be the next value in the original sequence. To determine the end value, issue a query using MAX of the identity column (for an ascending sequence), or MIN of the identity column (for a descending sequence), before unloading the data.

Examples

Example 1: Insert a new department with the following specifications into the DEPARTMENT table:

- Department number (DEPTNO) is 'E31'
- Department name (DEPTNAME) is 'ARCHITECTURE'
- Managed by (MGRNO) a person with number '00390'
- Reports to (ADMRDEPT) department 'E01'.

```
INSERT INTO DEPARTMENT
   VALUES ('E31', 'ARCHITECTURE', '00390', 'E01')
```

Example 2: Insert a new department into the DEPARTMENT table as in example 1, but do not assign a manager to the new department.

```
INSERT INTO DEPARTMENT (DEPTNO, DEPTNAME, ADMRDEPT )
   VALUES ('E31', 'ARCHITECTURE', 'E01')
```

Example 3: Insert two new departments using one statement into the DEPARTMENT table as in example 2, but do not assign a manager to the new department.

```
INSERT INTO DEPARTMENT (DEPTNO, DEPTNAME, ADMRDEPT)
   VALUES ('B11', 'PURCHASING', 'B01'),
          ('E41', 'DATABASE ADMINISTRATION', 'E01')
```

Example 4: Create a temporary table MA_EMP_ACT with the same columns as the EMP_ACT table. Load MA_EMP_ACT with the rows from the EMP_ACT table with a project number (PROJNO) starting with the letters 'MA'.

```
CREATE TABLE MA_EMP_ACT
      ( EMPNO CHAR(6)  NOT NULL,
        PROJNO CHAR(6)  NOT NULL,
        ACTNO SMALLINT  NOT NULL,
        EMPTIME DEC(5,2),
        EMSTDATE DATE,
        EMENDATE  DATE )
INSERT INTO MA_EMP_ACT
   SELECT * FROM EMP_ACT
     WHERE SUBSTR(PROJNO, 1, 2) = 'MA'
```

Example 5: Use a C program statement to add a skeleton project to the PROJECT table. Obtain the project number (PROJNO), project name (PROJNAME), department number (DEPTNO), and responsible employee (RESPEMP) from host variables. Use the current date as the project start date (PRSTDATE). Assign a NULL value to the remaining columns in the table.

```
EXEC SQL  INSERT INTO PROJECT (PROJNO, PROJNAME, DEPTNO, RESPEMP, PRSTDATE)
            VALUES (:PRJNO, :PRJNM, :DPTNO, :REMP, CURRENT DATE);
```