DECLARE GLOBAL TEMPORARY TABLE

The DECLARE GLOBAL TEMPORARY TABLE statement defines a temporary table for the current session. The declared temporary table description does not appear in the system catalog. It is not persistent and cannot be shared with other sessions. Each session that defines a declared global temporary table of the same name has its own unique description of the temporary table. When the session terminates, the rows of the table are deleted, and the description of the temporary table is dropped.

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:

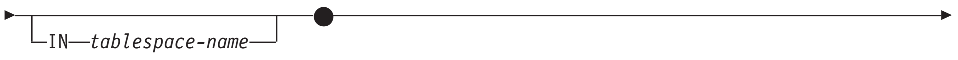
- SYSADM or DBADM authority
- USE privilege on the USER TEMPORARY table space.

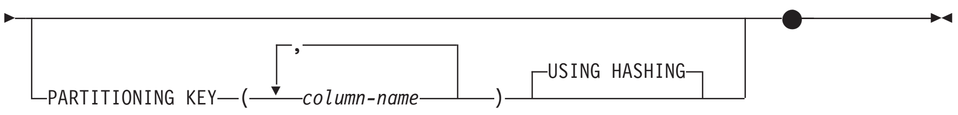
When defining a table using LIKE or a fullselect, the privileges held by the authorization ID of the statement must also include at least one of the following on each identified table or view:

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

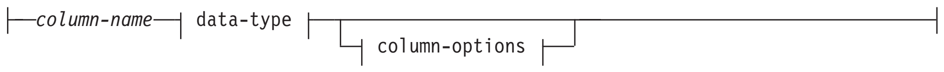
Syntax

```
►►──DECLARE GLOBAL TEMPORARY TABLE──table-name──────────────────────────────────►


         ┌─,─────────────────────────┐
         ▼                           │
►──┬─(────┤ column-definition ├──────┴─)──────────────────────┬─────────────────►
   ├─LIKE───┬─table-name2─┬──┬───────────────────────┬────────┤
   │        └─view-name───┘  └─┤ copy-options ├──────┘        │
   └─AS──(──fullselect──)──DEFINITION ONLY──┬────────────────────────┬──┘
                                            └─┤ copy-options ├───────┘


                         ┌─ON COMMIT DELETE ROWS───┐
►──●──┬──────────────┬──●─┼─────────────────────────┼──●──NOT LOGGED──●──────────►
      └─WITH REPLACE─┘   └─ON COMMIT PRESERVE ROWS─┘
```

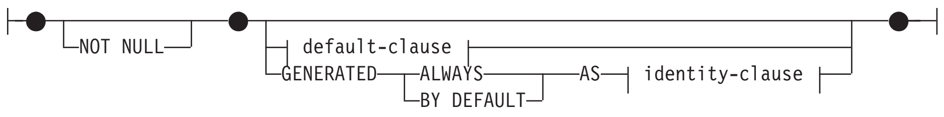




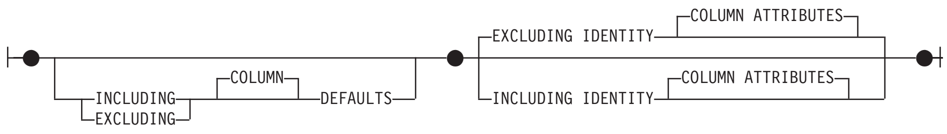
column-definition:



column-options:



copy-options:



Description

table-name
Names the temporary table. The qualifier, if specified explicitly, must be SESSION, otherwise an error is returned (SQLSTATE 428EK). If the qualifier is not specified, SESSION is implicitly assigned.

Each session that defines a declared global temporary table with the same *table-name* has its own unique description of that declared global temporary table. The WITH REPLACE clause must be specified if *table-name* identifies a declared temporary table that already exists in the session (SQLSTATE 42710).

It is possible that a table, view, alias, or nickname already exists in the catalog, with the same name and the schema name SESSION. In this case:

- A declared global temporary table *table-name* may still be defined without any error or warning

- Any references to SESSION.*table-name* will resolve to the declared global temporary table rather than the SESSION.*table-name* already defined in the catalog.

column-definition
Defines the attributes of a column of the temporary table.

column-name
Names a column of the table. The name cannot be qualified and the same name cannot be used for more than one column of the table (SQLSTATE 42711).

A table may have the following:

- a 4K page size with maximum of 500 columns where the byte counts of the columns must not be greater than 4005 in a 4K page size. Refer to “Row Size” on page 756 for more details.
- an 8K page size with maximum of 1 012 columns where the byte counts of the columns must not be greater than 8 101. Refer to “Row Size” on page 756 for more details.
- a 16K page size with maximum of 1 012 columns where the byte counts of the columns must not be greater than 16 293. Refer to “Row Size” on page 756 for more details.
- a 32K page size with maximum of 1 012 columns where the byte counts of the columns must not be greater than 32 677. Refer to “Row Size” on page 756 for more details.

data-type
See *data-type* in “CREATE TABLE” on page 712 for allowable types. Note that BLOB, CLOB, DBCLOB, LONG VARCHAR, LONG VARGRAPHIC, DATALINK, reference, and structured types cannot be used with declared global temporary tables (SQLSTATE 42962). This exception includes distinct types sourced on these restricted types.

FOR BIT DATA can be specified as part of character string data types.

column-options
Defines additional options related to the columns of the table.

NOT NULL
Prevents the column from containing null values. See *NOT NULL* in “CREATE TABLE” on page 712 for specification of null values.

default-clause
See *default-clause* in “CREATE TABLE” on page 712 for specification of defaults.

identity-clause
See *identity-clause* in “CREATE TABLE” on page 712 for specification of identity columns.

LIKE *table-name2* **or** *view-name*

Specifies that the columns of the table have exactly the same name and description as the columns of the identified table (*table-name2*) or view (*view-name*), or nickname (*nickname*). The name specified after LIKE must identify a table, view or nickname that exists in the catalog or a declared temporary table. A typed table or typed view cannot be specified (SQLSTATE 428EC).

The use of LIKE is an implicit definition of *n* columns, where *n* is the number of columns in the identified table or view.

- If a table is identified, then the implicit definition includes the column name, data type and nullability characteristic of each of the columns of *table-name2*. If EXCLUDING COLUMN DEFAULTS is not specified, then the column default is also included.
- If a view is identified, then the implicit definition includes the column name, data type, and nullability characteristic of each of the result columns of the fullselect defined in *view-name*.

Column default and identity column attributes may be included or excluded, based on the copy-attributes clauses.

The implicit definition does not include any other attributes of the identified table or view. Thus, the new table does not have any unique constraints, foreign key constraints, triggers, or indexes. The table is created in the table space either implicitly or explicitly, as specified by the IN clause.

The names used for *table-name2* and *view-name* can not be the same as the name of the global temporary table that is being created (SQLSTATE 428EC).

AS (*fullselect***) DEFINITION ONLY**

Specifies that the table definition is based on the column definitions from the result of a query expression. The use of AS (*fullselect*) is an implicit definition of *n* columns for the declared global temporary table, where *n* is the number of columns that would result from *fullselect*. The columns of the new table are defined by the columns that result from the *fullselect*. Every select list element must have a unique name (SQLSTATE 42711). The AS clause can be used in the select-clause to provide unique names.

The implicit definition includes the column name, data type, and nullability characteristic of each of the result columns of *fullselect*.

copy-options

These options specify whether or not to copy additional attributes of the source result table definition (table, view, or fullselect).

INCLUDING COLUMN DEFAULTS
Column defaults for each updatable column of the source result table definition are copied. Columns that are not updatable will not have a default defined in the corresponding column of the created table.

If LIKE *table-name2* is specified, and *table-name2* identifies a base table or declared temporary table, then INCLUDING COLUMN DEFAULTS is the default.

EXCLUDING COLUMN DEFAULTS
Column defaults are not copied from the source result table definition.

This clause is the default, except when LIKE *table-name* is specified and *table-name* identifies a base table or declared temporary table.

INCLUDING IDENTITY COLUMN ATTRIBUTES
If available, identity column attributes (START WITH, INCREMENT BY, and CACHE values) are copied from the source's result table definition. It is possible to copy these attributes if the element of the corresponding column in the table, view, or fullselect is the name of a column of a table, or the name of a column of a view, which directly or indirectly maps to the column name of a base table with the identity property. In all other cases, the columns of the new temporary table will not get the identity property. For example:

- the select list of the fullselect includes multiple instances of the name of an identity column (that is, selecting the same column more than once)
- the select list of the fullselect includes multiple identity columns (that is, it involves a join)
- the identity column is included in an expression in the select list
- the fullselect includes a set operation (union, except, or intersect).

EXCLUDING IDENTITY COLUMN ATTRIBUTES
Identity column attributes are not copied from the source result table definition.

ON COMMIT
Specifies the action taken on the global temporary table when a COMMIT operation is performed.

DELETE ROWS
All rows of the table will be deleted if no WITH HOLD cursor is open on the table. This is the default.

PRESERVE ROWS
Rows of the table will be preserved.

NOT LOGGED
Changes to the table are not logged, including creation of the table. When

a ROLLBACK (or ROLLBACK TO SAVEPOINT) operation is performed and the table was changed in the unit of work (or savepoint), then all rows of the table are deleted. If the table was created in the unit of work (or savepoint), then that table will be dropped. If the table was dropped in the unit of work (or savepoint) then the table will be restored, but with no rows. Furthermore, if a statement that performs an INSERT, UPDATE, or DELETE operation on the table encounters an error, all rows of the table are deleted.

WITH REPLACE
Indicates that, in the case that a declared global temporary table already exists with the specified name, the existing table is replaced with the temporary table defined by this statement (and all rows of the existing table are deleted).

When WITH REPLACE is not specified, then the name specified must not identify a declared global temporary table that already exists in the current session (SQLSTATE 42710).

IN *tablespace-name*
Identifies the table space in which the global temporary table will be instantiated. The table space must exist and be a USER TEMPORARY table space (SQLSTATE 42838), over which the authorization ID of the statement has USE privilege (SQLSTATE 42501). If this clause is not specified, a table space for the table is determined by choosing the USER TEMPORARY table space with the smallest sufficient page size over which the authorization ID of the statement has USE privilege. When more than one table space qualifies, preference is given according to who was granted the USE privilege:

1. the authorization ID
2. a group to which the authorization ID belongs
3. PUBLIC

If more than one table space still qualifies, the final choice is made by the database manager. When no USER TEMPORARY table space qualifies, an error is raised (SQLSTATE 42727).

Determination of the table space may change when:

- table spaces are dropped or created
- USE privileges are granted or revoked.

The sufficient page size of a table is determined by either the byte count of the row or the number of columns. Refer to “Row Size” on page 756 for more information.

PARTITIONING KEY (*column-name,...***)**
Specifies the partitioning key used when data in the table is partitioned.

Each *column-name* must identify a column of the table and the same column must not be identified more than once.

If this clause is not specified, and this table resides in a multiple partition nodegroup, then the partitioning key is defined as the first column of declared temporary table.

For declared temporary tables, in table spaces defined on single-partition nodegroups, any collection of columns can be used to define the partitioning key. If you do not specify this parameter, no partitioning key is created.

Note that partitioning key columns cannot be updated (SQLSTATE 42997).

USING HASHING
Specifies the use of the hashing function as the partitioning method for data distribution. This is the only partitioning method supported.

Notes

- **Referencing a declared global temporary table:** The description of a declared global temporary table does not appear in the DB2 catalog (SYSCAT.TABLES); therefore, it is not persistent and is not sharable across database connections. This means that each session that defines a declared global temporary table called *table-name* has its own possibly unique description of that declared global temporary table.

  In order to reference the declared global temporary table in an SQL statement (other than the DECLARE GLOBAL TEMPORARY TABLE statement), the table must be explicitly or implicitly qualified by the schema name SESSION. If *table-name* is not qualified by SESSION, declared global temporary tables are not considered when resolving the reference.

  A reference to SESSION.*table-name* in a connection that has not declared a global temporary table by that name will attempt to resolve from persistent objects in the catalog. If no such object exists, an error occurs (SQLSTATE 42704).
- When binding a package that has static SQL statements that refer to tables implicitly or explicitly qualified by SESSION, those statements will not be bound statically. When these statements are invoked, they will be incrementally bound, regardless of the VALIDATE option chosen while binding the package. At runtime, each table reference will be resolved to a declared temporary table, if it exists, or a permanent table. If neither exists, an error will be raised (SQLSTATE 42704).
- **Privileges:** When a declared global temporary table is defined, the definer of the table is granted all table privileges on the table, including the ability to drop the table. Additionally, these privileges are granted to PUBLIC.[89]

89. None of the privileges are granted with the GRANT option, and none of the privileges appear in the catalog table.

This enables any SQL statement in the session to reference a declared global temporary table that has already been defined in that session.

- **Instantiation and Termination:** For the explanations below, P denotes a session and T is a declared global temporary table in the session P:
  - An empty instance of T is created as a result of the DECLARE GLOBAL TEMPORARY TABLE statement that is executed in P.
  - Any SQL statement in P can make reference to T; and any reference to T in P is a reference to that same instance of T.
  - If a DECLARE GLOBAL TEMPORARY TABLE statement is specified within the SQL procedure compound statement (defined by BEGIN and END), the scope of the declared global temporary table is the connection, not just the compound statement, and the table is known outside of the compound statement. The table is not implicitly dropped at the END of the compound statement. A declared global temporary table cannot be defined multiple times by the same name in other compound statements in that session, unless the table has been explicitly dropped.
  - Assuming that the ON COMMIT DELETE ROWS clause was specified implicitly or explicitly, then when a commit operation terminates a unit of work in P, and there is no open WITH HOLD cursor in P that is dependent on T, the commit includes the operation DELETE FROM SESSION.T.
  - When a rollback operation terminates a unit of work or a savepoint in P, and that unit of work or savepoint includes a modification to SESSION.T, then the rollback includes the operation DELETE from SESSION.T.

    When a rollback operation terminates a unit of work or a savepoint in P, and that unit of work or savepoint includes the declaration of SESSION.T, then the rollback includes the operation DROP SESSION.T.

    If a rollback operation terminates a unit of work or a savepoint in P, and that unit of work or savepoint includes the drop of a declared temporary table SESSION.T, then the rollback will undo the drop of the table, but the table will have been emptied.
  - When the application process that declared T terminates or disconnects from the database, T is dropped and its instantiated rows are destroyed.
  - When the connection to the server at which T was declared terminates, T is dropped and its instantiated rows are destroyed.
- **Restrictions on the Use of Declared Global Temporary Tables:** Declared Global Temporary tables cannot:
  - Be specified in an ALTER, COMMENT, GRANT, LOCK, RENAME or REVOKE statement (SQLSTATE 42995).
  - Be referenced in a CREATE ALIAS, CREATE FUNCTION (SQL Scalar, Table, or Row), CREATE INDEX, CREATE TRIGGER, or CREATE VIEW statement (SQLSTATE 42995).

– Be specified in referential constraints (SQLSTATE 42995).

DELETE

The DELETE statement deletes rows from a table or view. Deleting a row from a view deletes the row from the table on which the view is based.

There are two forms of this statement:

- The *Searched* DELETE form is used to delete one or more rows (optionally determined by a search condition).
- The *Positioned* DELETE form is used to delete exactly one row (as determined by the current position of a cursor).

Invocation

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

Authorization

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

- DELETE privilege on the table or view for which rows are to be deleted
- CONTROL privilege on the table or view for which rows are to be deleted
- SYSADM or DBADM authority.

To execute a Searched DELETE statement, the privileges held by the authorization ID of the statement must also include at least one of the following for each table or view referenced by a subquery:

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

When the package is precompiled with SQL92 rules [90] and the searched form of a DELETE includes a reference to a column of the table or view in the *search-condition*, the privileges held by the authorization ID of the statement must also include at least one of the following:

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

When the specified table or view is preceded by the ONLY keyword, the privileges held by the authorization ID of the statement must also include the SELECT privilege for every subtable or subview of the specified table or view.

90. The package used to process the statement is precompiled using option LANGLEVEL with value SQL92E or MIA.

Group privileges are not checked for static DELETE statements.

Syntax

Searched DELETE:

```
>>──DELETE FROM──┬─table-name──────────────────┬──┬──────────────────────────┬──────►
                 ├─view-name───────────────────┤  │ ┌─AS─┐                   │
                 └─ONLY──(──┬─table-name─┬──)──┘  └─┴────┴─correlation-name─┘
                            └─view-name──┘

►──┬────────────────────────────┬─────────────────────────────────────────────────►◄
   └─WHERE──search-condition────┘
```

Positioned DELETE:

```
>>──DELETE FROM──┬─table-name──────────────────┬──WHERE CURRENT OF──cursor-name──────►◄
                 ├─view-name───────────────────┤
                 └─ONLY──(──┬─table-name─┬──)──┘
                            └─view-name──┘
```

Description

FROM *table-name* or *view-name*
: Identifies the table or view from which rows are to be deleted. The name must identify a table or view that exists in the catalog, but it must not identify a catalog table, a catalog view, a summary table or a read-only view. (For an explanation of read-only views, see "CREATE VIEW" on page 823.)

  If *table-name* is a typed table, rows of the table or any of its proper subtables may get deleted by the statement.

  If *view-name* is a typed view, rows of the underlying table or underlying tables of the view's proper subviews may get deleted by the statement. If *view-name* is a regular view with an underlying table that is a typed table, rows of the typed table or any of its proper subtables may get deleted by the statement.

  Only the columns of the specified table may be referenced in the WHERE clause. For a positioned DELETE, the associated cursor must also have specified the table or view in the FROM clause without using ONLY.

FROM ONLY (*table-name***)**
: Applicable to typed tables, the ONLY keyword specifies that the statement should apply only to data of the specified table and rows of proper subtables cannot be deleted by the statement. For a positioned DELETE,

the associated cursor must also have specified the table in the FROM clause using ONLY. If *table-name* is not a typed table, the ONLY keyword has no effect on the statement.

FROM ONLY (*view-name***)**
Applicable to typed views, the ONLY keyword specifies that the statement should apply only to data of the specified view and rows of proper subviews cannot be deleted by the statement. For a positioned DELETE, the associated cursor must also have specified the view in the FROM clause using ONLY. If *view-name* is not a typed view, the ONLY keyword has no effect on the statement.

correlation-name
May be used within the search-condition to designate the table or view. (For an explanation of correlation-name, see "Chapter 3. Language Elements" on page 63.)

WHERE
Specifies a condition that selects the rows to be deleted. The clause can be omitted, a search condition specified, or a cursor named. If the clause is omitted, all rows of the table or view are deleted.

search-condition
Is any search condition as described in "Search Conditions" on page 205. Each *column-name* in the search condition, other than in a subquery must identify a column of the table or view.

The *search-condition* is applied to each row of the table or view and the deleted rows are those for which the result of the *search-condition* is true.

If the search condition contains a subquery, the subquery can be thought of as being executed each time the *search condition* is applied to a row, and the results used in applying the *search condition*. In actuality, a subquery with no correlated references is executed once, whereas a subquery with a correlated reference may have to be executed once for each row. If a subquery refers to the object table of a DELETE statement or a dependent table with a delete rule of CASCADE or SET NULL, the subquery is completely evaluated before any rows are deleted.

CURRENT OF *cursor-name*
Identifies a cursor that is defined in a DECLARE CURSOR statement of the program. The DECLARE CURSOR statement must precede the DELETE statement.

The table or view named must also be named in the FROM clause of the SELECT statement of the cursor, and the result table of the cursor must not be read-only. (For an explanation of read-only result tables, see "DECLARE CURSOR" on page 841.)

When the DELETE statement is executed, the cursor must be positioned on a row: that row is the one deleted. After the deletion, the cursor is positioned before the next row of its result table. If there is no next row, the cursor is positioned after the last row.

Rules

- If the identified table or the base table of the identified view is a parent, the rows selected for delete must not have any dependents in a relationship with a delete rule of RESTRICT, and the DELETE must not cascade to descendent rows that have dependents in a relationship with a delete rule of RESTRICT.

  If the delete operation is not prevented by a RESTRICT delete rule, the selected rows are deleted. Any rows that are dependents of the selected rows are also affected:

  - The nullable columns of the foreign keys of any rows that are their dependents in a relationship with a delete rule of SET NULL are set to the null value.
  - Any rows that are their dependents in a relationship with a delete rule of CASCADE are also deleted, and the above rules apply, in turn, to those rows.

  The delete rule of NO ACTION is checked to enforce that any non-null foreign key refers to an existing parent row after the other referential constraints have been enforced.

Notes

- If an error occurs during the execution of a multiple row DELETE, no changes are made to the database.
- Unless appropriate locks already exist, one or more exclusive locks are acquired during the execution of a successful DELETE statement. Issuing a COMMIT or ROLLBACK statement will release the locks. Until the locks are released by a commit or rollback operation, the effect of the delete operation can only be perceived by:

  - The application process that performed the deletion
  - Another application process using isolation level UR.

  The locks can prevent other application processes from performing operations on the table.
- If an application process deletes a row on which any of its cursors are positioned, those cursors are positioned before the next row of their result table. Let C be a cursor that is positioned before row R (as a result of an OPEN, a DELETE through C, a DELETE through some other cursor, or a searched DELETE). In the presence of INSERT, UPDATE, and DELETE operations that affect the base table from which R is derived, the next

FETCH operation referencing C does not necessarily position C on R. For example, the operation can position C on R′, where R′ is a new row that is now the next row of the result table.

- SQLERRD(3) in the SQLCA shows the number of rows deleted from the object table after the statement executes. It does not include rows that were deleted as a result of a CASCADE delete rule. SQLERRD(5) in the SQLCA shows the number of rows affected by referential constraints and by triggered statements. It includes rows that were deleted as a result of a CASCADE delete rule and rows in which foreign keys were set to NULL as the result of a SET NULL delete rule. With regards to triggered statements, it includes the number of rows that were inserted, updated, or deleted. (For a description of the SQLCA, see “Appendix B. SQL Communications (SQLCA)” on page 1107.)
- If an error occurs that prevents deleting all rows matching the search condition and all operations required by existing referential constraints, no changes are made to the table and the error is returned.
- For any deleted row that includes currently linked files through DATALINK columns, the files are unlinked, and will be either restored or deleted, depending on the datalink column definition.

  An error may occur when attempting to delete a DATALINK value if the file server of value is no longer registered with the database server (SQLSTATE 55022).

  An error may also occur when deleting a row that has a link to a server that is unavailable at the time of deletion (SQLSTATE 57050).

Examples

Example 1: Delete department (DEPTNO) ‘D11’ from the DEPARTMENT table.

```
DELETE FROM DEPARTMENT
WHERE DEPTNO = 'D11'
```

Example 2: Delete all the departments from the DEPARTMENT table (that is, empty the table).

```
DELETE FROM DEPARTMENT
```

DESCRIBE

The DESCRIBE statement obtains information about a prepared statement. For an explanation of prepared statements, see “PREPARE” on page 954.

Invocation

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

Authorization

None required.

Syntax

```
►►──DESCRIBE──statement-name──INTO──descriptor-name──────────────────────────►◄
```

Description

statement-name
Identifies the statement about which information is required. When the DESCRIBE statement is executed, the name must identify a prepared statement.

INTO *descriptor-name*
Identifies an SQL descriptor area (SQLDA), which is described in “Appendix C. SQL Descriptor Area (SQLDA)” on page 1113. Before the DESCRIBE statement is executed, the following variables in the SQLDA must be set:

SQLN Indicates the number of variables represented by SQLVAR. (SQLN provides the dimension of the SQLVAR array.) SQLN must be set to a value greater than or equal to zero before the DESCRIBE statement is executed.

When the DESCRIBE statement is executed, the database manager assigns values to the variables of the SQLDA as follows:

SQLDAID
The first 6 bytes are set to ’SQLDA ’ (that is, 5 letters followed by the space character).

The seventh byte, called SQLDOUBLED, is set to ’2’ if the SQLDA contains two SQLVAR entries for every select-list item (or, *column* of the result table). This technique is used in order to accommodate LOB, distinct type, structured type, or reference type result columns. Otherwise, SQLDOUBLED is set to the space character.

The doubled flag is set to space if there is not enough room in the SQLDA to contain the entire DESCRIBE reply.

The eighth byte is set to the space character.

SQLDABC
Length of the SQLDA.

SQLD If the prepared statement is a SELECT, the number of columns in its result table; otherwise, 0.

SQLVAR
If the value of SQLD is 0, or greater than the value of SQLN, no values are assigned to occurrences of SQLVAR.

If the value is *n*, where *n* is greater than 0 but less than or equal to the value of SQLN, values are assigned to the first *n* occurrences of SQLVAR so that the first occurrence of SQLVAR contains a description of the first column of the result table, the second occurrence of SQLVAR contains a description of the second column of the result table, and so on. The description of a column consists of the values assigned to SQLTYPE, SQLLEN, SQLNAME, SQLLONGLEN, and SQLDATATYPE_NAME.

Basic SQLVAR

SQLTYPE
A code showing the data type of the column and whether or not it can contain null values.

SQLLEN
A length value depending on the data type of the result columns. SQLLEN is 0 for LOB data types.

SQLNAME
If the derived column is not a simple column reference, then sqlname contains an ASCII numeric literal value, which represents the derived column's original position within the select list; otherwise, sqlname contains the name of the column.

Secondary SQLVAR

These variables are only used if the number of SQLVAR entries are doubled to accommodate LOB, distinct type, structured type, or reference type columns.

SQLLONGLEN
The length attribute of a BLOB, CLOB, or DBCLOB column.

SQLDATATYPE_NAME
For any user-defined type (distinct or structured) column, the database manager sets this to the fully

qualified user-defined type name. For a reference type column, the database manager sets this to the fully qualified user-defined type name of the target type of the reference. Otherwise, schema name is SYSIBM and the type name is the name in the TYPENAME column of the SYSCAT.DATATYPES catalog view.

Notes

- Before the DESCRIBE statement is executed, the value of SQLN must be set to indicate how many occurrences of SQLVAR are provided in the SQLDA and enough storage must be allocated to contain SQLN occurrences. To obtain the description of the columns of the result table of a prepared SELECT statement, the number of occurrences of SQLVAR must not be less than the number of columns.
- If a LOB of a large size is expected, then remember that manipulating this large object will affect application memory. Given this condition, consider using locators or file reference variables. Modify the SQLDA after the DESCRIBE statement is executed but prior to allocating storage so that an SQLTYPE of SQL_TYP_xLOB is changed to SQL_TYP_xLOB_LOCATOR or SQL_TYP_xLOB_FILE with corresponding changes to other fields such as SQLLEN. Then allocate storage based on SQLTYPE and continue.

  See the *Application Development Guide* for more information on using locators and file reference variables with the SQLDA.
- Code page conversions between extended Unix code (EUC) code pages and DBCS code pages can result in the expansion and contraction of character lengths. See the *Application Development Guide* for information on handling such situations.
- If a structured type is being selected, but no FROM SQL transform is defined (either because no TRANSFORM GROUP was specified using the CURRENT DEFAULT TRANSFORM GROUP special register (SQLSTATE 428EM), or because the named group does not have a FROM SQL transform function defined (SQLSTATE 42744)), the DESCRIBE will return an error.
- **Allocating the SQLDA:** Among the possible ways to allocate the SQLDA are the three described below.

  First Technique: Allocate an SQLDA with enough occurrences of SQLVAR to accommodate any select list that the application will have to process. If the table contains any LOB, distinct type, structured type, or reference type columns, the number of SQLVARs should be double the maximum number of columns; otherwise the number should be the same as the maximum number of columns. Having done the allocation, the application can use this SQLDA repeatedly.

  This technique uses a large amount of storage that is never deallocated, even when most of this storage is not used for a particular select list.

Second Technique: Repeat the following two steps for every processed select list:

1. Execute a DESCRIBE statement with an SQLDA that has no occurrences of SQLVAR; that is, an SQLDA for which SQLN is zero. The value returned for SQLD is the number of columns in the result table. This is either the required number of occurrences of SQLVAR or half the required number. Because there were no SQLVAR entries, a warning with SQLSTATE 01005 will be issued. If the SQLCODE accompanying that warning is equal to one of +237, +238 or +239, the number of SQLVAR entries should be double the value returned in SQLD. [91]
2. Allocate an SQLDA with enough occurrences of SQLVAR. Then execute the DESCRIBE statement again, using this new SQLDA.

This technique allows better storage management than the first technique, but it doubles the number of DESCRIBE statements.

Third Technique: Allocate an SQLDA that is large enough to handle most, and perhaps all, select lists but is also reasonably small. Execute DESCRIBE and check the SQLD value. Use the SQLD value for the number of occurrences of SQLVAR to allocate a larger SQLDA, if necessary.

This technique is a compromise between the first two techniques. Its effectiveness depends on a good choice of size for the original SQLDA.

Example

In a C program, execute a DESCRIBE statement with an SQLDA that has no occurrences of SQLVAR. If SQLD is greater than zero, use the value to allocate an SQLDA with the necessary number of occurrences of SQLVAR and then execute a DESCRIBE statement using that SQLDA.

```
EXEC SQL  BEGIN DECLARE SECTION;
  char  stmt1_str[200];
EXEC SQL  END DECLARE SECTION;
EXEC SQL  INCLUDE SQLDA;
EXEC SQL  DECLARE DYN_CURSOR CURSOR FOR STMT1_NAME;

... /* code to prompt user for a query, then to generate */
    /* a select-statement in the stmt1_str           */
EXEC SQL  PREPARE STMT1_NAME FROM :stmt1_str;

... /* code to set SQLN to zero and to allocate the SQLDA */
EXEC SQL  DESCRIBE STMT1_NAME INTO :sqlda;

... /* code to check that SQLD is greater than zero, to set */
    /* SQLN to SQLD, then to re-allocate the SQLDA         */
```

91. The return of these positive SQLCODEs assumes that the SQLWARN bind option setting was YES (return positive SQLCODEs). If SQLWARN was set to NO, +238 is still returned to indicate that the number of SQLVAR entries must be double the value returned in SQLD.

```
EXEC SQL  DESCRIBE STMT1_NAME INTO :sqlda;

... /* code to prepare for the use of the SQLDA             */
    /* and allocate buffers to receive the data             */
EXEC SQL  OPEN DYN_CURSOR;

... /* loop to fetch rows from result table                 */
EXEC SQL  FETCH DYN_CURSOR USING DESCRIPTOR :sqlda;
.
.
.
```

DISCONNECT

The DISCONNECT statement destroys one or more connections when there is no active unit of work (that is, after a commit or rollback operation).[92]

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

```
                               (1)
>>─DISCONNECT──┬─server-name────┬────────────────────────────────────────────><
               ├─host-variable──┤
               ├─CURRENT────────┤
               │      ┌─SQL─┐   │
               └─ALL──┴─────┴───┘
```

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.

92. If a single connection is the target of the DISCONNECT statement, then the connection is destroyed only if the database has participated in any existing unit of work, not whether there is an active unit of work. For example, if several other databases have done work but the target in question has not, it can still be disconnected without destroying the connection.

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
Indicates that all existing connections of the application process are to be destroyed. An error or warning does not occur if no connections exist when the statement is executed. The optional keyword SQL is included to be consistent with the syntax of the RELEASE statement.

Rules

- Generally, the DISCONNECT statement cannot be executed while within a unit of work. If attempted, an error (SQLSTATE 25000) is raised. The exception to this rule is if a single connection is specified to be disconnected and the database has not participated in an existing unit of work. In this case, it does not matter if there is an active unit of work when the DISCONNECT statement is issued.
- The DISCONNECT statement cannot be executed at all in the Transaction Processing (TP) Monitor environment (SQLSTATE 25000). It is used when the SYNCPOINT precompiler option is set to TWOPHASE.

Notes

- If the DISCONNECT statement is successful, each identified connection is destroyed.

  If the DISCONNECT statement is unsuccessful, the connection state of the application process and the states of its connections are unchanged.
- If DISCONNECT is used to destroy the current connection, the next executed SQL statement should be CONNECT or SET CONNECTION.
- Type 1 CONNECT semantics do not preclude the use of DISCONNECT. However, though DISCONNECT CURRENT and DISCONNECT ALL can be used, they will not result in a commit operation like a CONNECT RESET statement would do.

  If *server-name* or *host-variable* is specified in the DISCONNECT statement, it must identify the current connection because Type 1 CONNECT only supports one connection at a time. Generally, DISCONNECT will fail if within a unit of work with the exception noted in “Rules”.
- Resources are required to create and maintain remote connections. Thus, a remote connection that is not going to be reused should be destroyed as soon as possible.

- Connections can also be destroyed during a commit operation because the connection option is in effect. The connection option could be AUTOMATIC, CONDITIONAL, or EXPLICIT, which can be set as a precompiler option or through the SET CLIENT API at run time. See "Options that Govern Distributed Unit of Work Semantics" on page 39 for information about the specification of the DISCONNECT option.

Examples

Example 1: The SQL connection to IBMSTHDB is no longer needed by the application. The following statement should be executed after a commit or rollback operation to destroy the connection.

```
EXEC SQL DISCONNECT IBMSTHDB;
```

Example 2: The current connection is no longer needed by the application. The following statement should be executed after a commit or rollback operation to destroy the connection.

```
EXEC SQL DISCONNECT CURRENT;
```

Example 3: The existing connections are no longer needed by the application. The following statement should be executed after a commit or rollback operation to destroy all the connections.

```
EXEC SQL DISCONNECT ALL;
```

DROP

The DROP statement deletes an object. Any objects that are directly or indirectly dependent on that object are either deleted or made inoperative. (See "Inoperative Trigger" on page 786 and "Inoperative views" on page 833 for details.) Whenever an object is deleted, its description is deleted from the catalog and any packages that reference the object are invalidated.

Invocation

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

Authorization

The privileges that must be held by the authorization ID of the DROP statement when dropping objects that allow two-part names must include one of the following or an error will result (SQLSTATE 42501):

- SYSADM or DBADM authority
- DROPIN privilege on the schema for the object
- definer of the object as recorded in the DEFINER column of the catalog view for the object
- CONTROL privilege on the object (applicable only to indexes, index specifications, nicknames, packages, tables, and views).
- definer of the user-defined type as recorded in the DEFINER column of the catalog view SYSCAT.DATATYPES (applicable only when dropping a method associated with a user-defined type)

The authorization ID of the DROP statement when dropping a table or view hierarchy must hold one of the above privileges for each of the tables or views in the hierarchy.

The authorization ID of the DROP statement when dropping a schema must have SYSADM or DBADM authority or be the schema owner as recorded in the OWNER column of SYSCAT.SCHEMATA.

The authorization ID of the DROP statement when dropping a buffer pool, nodegroup, or table space must have SYSADM or SYSCTRL authority.

The authorization ID of the DROP statement when dropping an event monitor, server definition, data type mapping, function mapping or a wrapper must have SYSADM or DBADM authority.

The authorization ID of the DROP statement when dropping a user mapping must have SYSADM or DBADM authority, if this authorization ID is different from the federated database authorization name within the mapping. Otherwise, if the authorization ID and the authorization name match, no authorities or privileges are required.

The authorization ID of the DROP statement when dropping a transform must hold SYSADM or DBADM authority, or must be the DEFINER of *type-name*.

Syntax

```
►►──DROP─────────────────────────────────────────────────────────────►
```

```
           (1)
►►──┬─ALIAS──────alias-name──────────────────────────────────────────────┬─────►◄
    ├─BUFFERPOOL──bufferpool-name────────────────────────────────────────┤
    ├─EVENT──MONITOR──event-monitor-name─────────────────────────────────┤
    ├─FUNCTION──────function-name──┬──────────────────────────┬──────────┤
    │                              └─(─┬────────────────┬─)─┘          │
    │                                  │  ┌─,────────┐  │              │
    │                                  │  ▼          │  │              │
    │                                  └──data-type──┴──┘              │
    ├─SPECIFIC FUNCTION──specific-name───────────────────────────────────┤
    ├─FUNCTION MAPPING──function-mapping-name────────────────────────────┤
    │                  (2)                                               │
    ├─INDEX──index-name──────────────────────────────────────────────────┤
    ├─INDEX EXTENSION──index-extension-name──RESTRICT────────────────────┤
    ├─METHOD──method-name──┬──────────────────────────┬──FOR──type-name──┤
    │                      └─(─┬────────────────┬─)─┘                    │
    │                          │  ┌─,────────┐  │                        │
    │                          │  ▼          │  │                        │
    │                          └──datatype───┴──┘                        │
    ├─SPECIFIC METHOD──specific-name─────────────────────────────────────┤
    ├─NICKNAME──nickname─────────────────────────────────────────────────┤
    ├─NODEGROUP──nodegroup-name──────────────────────────────────────────┤
    │             (3)                                                    │
    ├─PACKAGE──────package-name──────────────────────────────────────────┤
    ├─PROCEDURE──procedure-name──┬──────────────────────────┬────────────┤
    │                            └─(─┬────────────────┬─)─┘            │
    │                                │  ┌─,────────┐  │                │
    │                                │  ▼          │  │                │
    │                                └──data-type──┴──┘                │
    ├─SPECIFIC PROCEDURE──specific-name──────────────────────────────────┤
    ├─SCHEMA──schema-name──RESTRICT──────────────────────────────────────┤
    ├─SERVER──server-name────────────────────────────────────────────────┤
    ├─TABLE──table-name──────────────────────────────────────────────────┤
    ├─TABLE HIERARCHY──root-table-name───────────────────────────────────┤
    │                      ┌─,──────────────────┐                        │
    │                      ▼                    │                        │
    ├──┬─TABLESPACE──┬──────tablespace-name─────┴────────────────────────┤
    │  └─TABLESPACES─┘                                                   │
    ├──┬─TRANSFORM──┬──┬─ALL────────┬──FOR──type-name────────────────────┤
    │  └─TRANSFORMS─┘  └─group-name─┘                                    │
    ├─TRIGGER──trigger-name──────────────────────────────────────────────┤
    ├─┬──────────────────┬─TYPE──type-name───────────────────────────────┤
    │ │             (4)  │                                               │
    │ └─DISTINCT─────────┘                                               │
    ├─TYPE MAPPING──type-mapping-name────────────────────────────────────┤
    ├─USER MAPPING FOR──┬─authorization-name─┬──SERVER──server-name──────┤
    │                   └─USER───────────────┘                           │
    ├─VIEW──view-name────────────────────────────────────────────────────┤
    ├─VIEW HIERARCHY──root-view-name─────────────────────────────────────┤
    └─WRAPPER──wrapper-name──────────────────────────────────────────────┘
```

Notes:

1 SYNONYM can be used as a synonym for ALIAS.

2 *Index-name* can be the name of either an index or an index specification.

3 PROGRAM can be used as a synonym for PACKAGE.

4 DATA can also be used when dropping any user-defined type.

Description

ALIAS *alias-name*
Identifies the alias that is to be dropped. The *alias-name* must identify an alias that is described in the catalog (SQLSTATE 42704). The specified alias is deleted.

All tables, views and triggers[93] that reference the alias are made inoperative.

BUFFERPOOL *bufferpool-name*
Identifies the buffer pool that is to be dropped. The *bufferpool-name* must identify a buffer pool that is described in the catalog (SQLSTATE 42704). There can be no table spaces assigned to the buffer pool (SQLSTATE 42893). The IBMDEFAULTBP buffer pool cannot be dropped (SQLSTATE 42832). The storage for the buffer pool will not be released until the database is stopped.

EVENT MONITOR *event-monitor-name*
Identifies the event monitor that is to be dropped. The *event-monitor-name* must identify an event monitor that is described in the catalog (SQLSTATE 42704).

If the identified event monitor is ON, an error (SQLSTATE 55034) is raised. Otherwise, the event monitor is deleted.

If there are event files in the target path of the event monitor when the event monitor is dropped, the event files are not deleted. However, if a new event monitor is created which specifies the same target path, then the event files are deleted.

FUNCTION
Identifies an instance of a user-defined function (either a complete function or a function template) that is to be dropped. The function instance specified must be a user-defined function described in the catalog. Functions implicitly generated by the CREATE DISTINCT TYPE statement cannot be dropped.

There are several different ways available to identify the function instance:

93. This includes both the table referenced in the ON clause of the CREATE TRIGGER statement and all tables referenced within the triggered SQL statements.

FUNCTION *function-name*
Identifies the particular function, and is valid only if there is exactly one function instance with the *function-name*. The function thus identified may have any number of parameters defined for it. In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified object names. If no function by this name exists in the named or implied schema, an error (SQLSTATE 42704) is raised. If there is more than one specific instance of the function in the named or implied schema, an error (SQLSTATE 42854) is raised.

FUNCTION *function-name* **(***data-type***,...)**
Provides the function signature, which uniquely identifies the function to be dropped. The function selection algorithm is not used.

function-name
Gives the function name of the function to be dropped. In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified object names.

(data-type,...)
Must match the data types that were specified on the CREATE FUNCTION statement in the corresponding position. The number of data types, and the logical concatenation of the data types is used to identify the specific function instance which is to be dropped.

If the *data-type* is unqualified, the type name is resolved by searching the schemas on the SQL path. This also applies to data type names specified for a REFERENCE type.

It is not necessary to specify the length, precision or scale for the parameterized data types. Instead, an empty set of parentheses may be coded to indicate that these attributes are to be ignored when looking for a data type match.

FLOAT() cannot be used (SQLSTATE 42601) since the parameter value indicates different data types (REAL or DOUBLE).

However, if length, precision, or scale is coded, the value must exactly match that specified in the CREATE PROCEDURE statement.

A type of FLOAT(n) does not need to match the defined value for n since 0<n<25 means REAL and 24<n<54 means DOUBLE. Matching occurs based on whether the type is REAL or DOUBLE.

If no function with the specified signature exists in named or implied schema, an error (SQLSTATE 42883) is raised.

SPECIFIC FUNCTION *specific-name*
Identifies the particular user-defined function that is to be dropped, using the specific name either specified or defaulted to at function creation time. In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified object names. The *specific-name* must identify a specific function instance in the named or implied schema; otherwise, an error (SQLSTATE 42704) is raised.

It is not possible to drop a function that is in either the SYSIBM schema or the SYSFUN schema (SQLSTATE 42832).

Other objects can be dependent upon a function. All such dependencies must be removed before the function can be dropped, with the exception of packages which are marked inoperative. An attempt to drop a function with such dependencies will result in an error (SQLSTATE 42893). See page 884 for a list of these dependencies.

If the function can be dropped, it is dropped.

Any package dependent on the specific function being dropped is marked as inoperative. Such a package is not implicitly rebound. It must either be rebound by use of the BIND or REBIND command or it must be reprepared by use of the PREP command. See the *Command Reference* for information on these commands.

FUNCTION MAPPING *function-mapping-name*
Identifies the function mapping to be dropped. The *function-mapping-name* must identify a user-defined function mapping that is described in the catalog (SQLSTATE 42704). The function mapping is deleted from the database.

Default function mappings cannot be dropped. However, they can be disabled. For an example, see Example 3 in “CREATE FUNCTION MAPPING” on page 657.

Packages having a dependency on a dropped function mapping are invalidated.

INDEX *index-name*
Identifies the index or index specification that is to be dropped. The *index-name* must identify an index or index specification that is described in the catalog (SQLSTATE 42704). It cannot be an index required by the

system for a primary key or unique constraint or for a replicated summary table (SQLSTATE 42917). The specified index or index specification is deleted.

Packages having a dependency on a dropped index or index specification are invalidated.

INDEX EXTENSION *index-extension-name* **RESTRICT**
Identifies the index extension that is to be dropped. The *index-extension-name* must identify an index extension that is described in the catalog (SQLSTATE 42704). The RESTRICT keyword enforces the rule that no index can be defined that depends on this index extension definition (SQLSTATE 42893).

METHOD
Identifies a method body that is to be dropped. The method body specified must be a method described in the catalog (SQLSTATE 42704). Method bodies that are implicitly generated by the CREATE TYPE statement cannot be dropped.

DROP METHOD deletes the body of a method, but the method specification (signature) remains as a part of the definition of the subject type. After dropping the body of a method, the method specification can be removed from the subject type definition by ALTER TYPE DROP METHOD.

There are several ways available to identify the method body to be dropped:

METHOD *method-name*
Identifies the particular method dropped, and is valid only if there is exactly one method instance with name *method-name* and subject type *type-name*. Thus, the method identified may have any number of parameters. If no method by this name exists for the type *type-name*, an error is raised (SQLSTATE 42704). If there is more than one specific instance of the method for the named data type, an error is raised (SQLSTATE 42854).

METHOD *method-name* **(***data-type***,...)**
Provides the method signature, which uniquely identifies the method to be dropped. The method selection algorithm is not used.

method-name
The method name of the method to be dropped for the specified type. The name must be an unqualified identifier.

(*data-type***, ...)**
Must match the data types that were specified in the corresponding positions of the method-specification of the CREATE TYPE or ALTER TYPE statement. The number of data

types and the logical concatenation of the data types are used to identify the specific method instance which is to be dropped.

If the data-type is unqualified, the type name is resolved by searching the schemas on the SQL path.

It is not necessary to specify the length, precision or scale for the parameterized data types. Instead, an empty set of parentheses may be coded to indicate that these attributes are to be ignored when looking for a data type match.

FLOAT() cannot be used (SQLSTATE 42601) since the parameter value indicates different data types (REAL or DOUBLE).

However, if length, precision, or scale is coded, the value must exactly match that specified in the CREATE TYPE statement.

A type of FLOAT(n) does not need to match the defined value for n since 0<n<25 means REAL and 24<n<54 means DOUBLE. Matching occurs based on whether the type is REAL or DOUBLE.

If no method with the specified signature exists for the named data type, an error is raised (SQLSTATE 42883).

FOR *type-name*
: Names the type for which the specified method is to be dropped. The name must identify a type already described in the catalog (SQLSTATE 42704). In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified type name. In static SQL statements, the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified type names.

SPECIFIC METHOD *specific-name*
: Identifies the particular method that is to be dropped, using a name either specified or defaulted to at CREATE TYPE or ALTER TYPE time. If the specific name is unqualified, the CURRENT SCHEMA special register is used as a qualifier for an unqualified specific name in dynamic SQL. In static SQL statements the QUALIFIER precompile/bind option implicitly specifies the qualifier for an unqualified specific name. The specific-name must identify a method; otherwise, an error is raised (SQLSTATE 42704).

Other objects can be dependent upon a method. All such dependencies must be removed before the method can be dropped, with the exception of packages which will be marked inoperative if the drop is successful. An attempt to drop a method with such dependencies will result in an error (SQLSTATE 42893).

If the method can be dropped, it will be dropped.

Any package dependent on the specific method being dropped is marked as inoperative. Such a package is not implicitly re-bound. Either it must be re-bound by use of the BIND or REBIND command, or it must be re-prepared by use of the PREP command. See *Command Reference* for information on these commands.

NICKNAME *nickname*
Identifies the nickname to be dropped. The nickname must be listed in the catalog (SQLSTATE 42704). The nickname is deleted from the database.

All information about the columns and indexes associated with the nickname is deleted from the catalog. Any index specifications that are dependent on the nickname are dropped. Any views dependent on the nickname are marked inoperative. Any packages depending on the dropped index specifications or inoperative views are invalidated. The data source table that the nickname references is not affected.

NODEGROUP *nodegroup-name*
Identifies the nodegroup that is to be dropped. *nodegroup-name* must identify a nodegroup that is described in the catalog (SQLSTATE 42704). This is a one-part name.

Dropping a nodegroup drops all table spaces defined in the nodegroup. All existing database objects with dependencies on the tables in the table spaces (such as packages, referential constraints, etc.) are dropped or invalidated (as appropriate), and dependent views and triggers are made inoperative.

System defined nodegroups cannot be dropped (SQLSTATE 42832).

If a DROP NODEGROUP is issued against a nodegroup that is currently undergoing a data redistribution, the DROP NODEGROUP operation fails an error is returned (SQLSTATE 55038). However, a partially redistributed nodegroup can be dropped. A nodegroup can become partially redistributed if a REDISTRIBUTE NODEGROUP command does not execute to completion. This can happen if it gets interrupted by either an error or a force application all command.[94]

PACKAGE *package-name*
Identifies the package that is to be dropped. The *package-name* must identify a package that is described in the catalog (SQLSTATE 42704). The specified package is deleted. All privileges on the package are also deleted.

94. For a partially redistributed nodegroup, the REBALANCE_PMAP_ID in the SYSCAT.NODEGROUPS catalog is not −1.

PROCEDURE
Identifies an instance of a stored procedure that is to be dropped. The procedure instance specified must be a stored procedure described in the catalog.

There are several different ways available to identify the procedure instance:

PROCEDURE *procedure-name*
Identifies the particular procedure, and is valid only if there is exactly one procedure instance with the *procedure-name* in the schema. The procedure thus identified may have any number of parameters defined for it. If no procedure by this name exists in the named or implied schema, an error (SQLSTATE 42704) is raised. In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified object names. If there is more than one specific instance of the procedure in the named or implied schema, an error (SQLSTATE 42854) is raised.

PROCEDURE *procedure-name* **(***data-type***,...)**
Provides the procedure signature, which uniquely identifies the procedure to be dropped. The procedure selection algorithm is not used.

procedure-name
Gives the procedure name of the procedure to be dropped. In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified object names.

(data-type,...)
Must match the data types that were specified on the CREATE PROCEDURE statement in the corresponding position. The number of data types, and the logical concatenation of the data types is used to identify the specific procedure instance which is to be dropped.

If the *data-type* is unqualified, the type name is resolved by searching the schemas on the SQL path. This also applies to data type names specified for a REFERENCE type.

It is not necessary to specify the length, precision or scale for the parameterized data types. Instead, an empty set of parentheses may be coded to indicate that these attributes are to be ignored when looking for a data type match.

FLOAT() cannot be used (SQLSTATE 42601) since the parameter value indicates different data types (REAL or DOUBLE).

However, if length, precision, or scale is coded, the value must exactly match that specified in the CREATE FUNCTION statement.

A type of FLOAT(n) does not need to match the defined value for n since 0<n<25 means REAL and 24<n<54 means DOUBLE. Matching occurs based on whether the type is REAL or DOUBLE.

If no procedure with the specified signature exists in named or implied schema, an error (SQLSTATE 42883) is raised.

SPECIFIC PROCEDURE *specific-name*
Identifies the particular stored procedure that is to be dropped, using the specific name either specified or defaulted to at procedure creation time. In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified object names. The *specific-name* must identify a specific procedure instance in the named or implied schema; otherwise, an error (SQLSTATE 42704) is raised.

SCHEMA *schema-name* **RESTRICT**
Identifies the schema that is to be dropped. The *schema-name* must identify a schema that is described in the catalog (SQLSTATE 42704). The RESTRICT keyword enforces the rule that no objects can be defined in the specified schema for the schema to be deleted from the database (SQLSTATE 42893).

SERVER *server-name*
Identifies the data source whose definition is to be dropped from the catalog. The *server-name* must identify a data source that is described in the catalog (SQLSTATE 42704). The definition of the data source is deleted.

All nicknames for tables and views residing at the data source are dropped. Any index specifications dependent on these nicknames are dropped. Any user-defined function mappings, user-defined type mappings, and user mappings that are dependent on the dropped server definition are also dropped. All packages dependent on the dropped server definition, function mappings, nicknames, and index specifications are invalidated.

TABLE *table-name*
Identifies the base table, declared temporary table, or summary table that is to be dropped. The *table-name* must identify a table that is described in the catalog or, if it is a declared temporary table, then the *table-name* must

be qualified by the schema name SESSION and exist in the application (SQLSTATE 42704). The subtables of a typed table are dependent on their supertables. All subtables must be dropped before a supertable can be dropped (SQLSTATE 42893). The specified table is deleted from the database.

All indexes, primary keys, foreign keys, check constraints, and summary tables referencing the table are dropped. All views and triggers[95] that reference the table are made inoperative. All packages depending on any object dropped or marked inoperative will be invalidated. This includes packages dependent on any supertables above the subtable in the hierarchy. Any reference columns for which the dropped table is defined as the scope of the reference become unscoped.

Packages are not dependent on declared temporary tables, and therefore are not invalidated when such a table is dropped.

All files that are linked through any DATALINK columns are unlinked. The unlink operation is performed asynchronously so the files may not be immediately available for other operations.

When a subtable is dropped from a table hierarchy, the columns associated with the subtable are no longer accessible although they continue to be considered with respect to limits on the number of columns and size of the row. Dropping a subtable has the effect of deleting all the rows of the subtable from the supertables. This may result in activation of triggers or referential integrity constraints defined on the supertables.

When a declared temporary table is dropped, and its creation preceded the active unit of work or savepoint, then the table will be functionally dropped and the application will not be able to access the table. However, the table will still reserve some space in its table space and will prevent that USER TEMPORARY table space from being dropped or the nodegroup of the USER TEMPORARY table space from being redistributed until the unit of work is committed or savepoint is ended. Dropping a declared temporary table causes the data in the table to be destroyed, regardless of whether DROP is committed or rolled back.

TABLE HIERARCHY *root-table-name*
Identifies the typed table hierarchy that is to be dropped. The *root-table-name* must identify a typed table that is the root table in the typed table hierarchy (SQLSTATE 428DR). The typed table identified by *root-table-name* and all of its subtables are deleted from the database.

All indexes, summary tables, primary keys, foreign keys, and check constraints referencing the dropped tables are dropped. All views and

95. This includes both the table referenced in the ON clause of the CREATE TRIGGER statement and all tables referenced within the triggered SQL statements.

triggers that reference the dropped tables are made inoperative. All packages depending on any object dropped or marked inoperative will be invalidated. Any reference columns for which one of the dropped tables is defined as the scope of the reference become unscoped.

All files that are linked through any DATALINK columns are unlinked. The unlink operation is performed asynchronously so the files may not be immediately available for other operations.

Unlike dropping a single subtable, dropping the table hierarchy does not result in the activation of delete triggers of any tables in the hierarchy nor does it log the deleted rows.

TABLESPACE or **TABLESPACES** *tablespace-name*
Identifies the table spaces that are to be dropped. *tablespace-name* must identify a table space that is described in the catalog (SQLSTATE 42704). This is a one-part name.

The table spaces will not be dropped (SQLSTATE 55024) if there is any table that stores at least one of its parts in a table space being dropped and has one or more of its parts in another table space that is not being dropped (these tables would need to be dropped first). System table spaces cannot be dropped (SQLSTATE 42832). A SYSTEM TEMPORARY table space cannot be dropped (SQLSTATE 55026) if it is the only temporary table space that exists in the database. A USER TEMPORARY table space cannot be dropped if there is a declared temporary table created in it (SQLSTATE 55039). Even if a declared temporary table has been dropped, the USER TEMPORARY table space will still be considered to be in use until the unit of work containing the DROP TABLE has been committed.

Dropping a table space drops all objects defined in the table space. All existing database objects with dependencies on the table space, such as packages, referential constraints, etc. are dropped or invalidated (as appropriate), and dependent views and triggers are made inoperative.

Containers created by the user are not deleted. Any directories in the path of the container name that were created by the database manager on CREATE TABLESPACE will be deleted. All containers that are below the database directory are deleted. For SMS table spaces, the deletions occur after all connections are disconnected or the DEACTIVATE DATABASE command is issued.

TRANSFORM ALL FOR *type-name*
Indicates that all transforms groups defined for the user-defined data type *type-name* are to be dropped. The transform functions referenced in these groups are not dropped. In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements, the QUALIFIER precompile/bind option

implicitly specifies the qualifier for unqualified object names. The *type-name* must identify a user-defined type described in the catalog (SQLSTATE 42704).

If there are not transforms defined for *type-name*, an error is raised (SQLSTATE 42740).

DROP TRANSFORM is the inverse of CREATE TRANSFORM. It causes the transform functions associated with certain groups, for a given datatype, to become undefined. The functions formerly associated with these groups still exist and can still be called explicitly, but they no longer have the transform property, and are no longer invoked implicitly for exchanging values with the host language environment.

The transform group is not dropped if there is a user-defined function (or method) written in a language other than SQL that has a dependency on one of the group's transform functions defined for the user-defined type *type-name* (SQLSTATE 42893). Such a function has a dependency on the transform function associated with the referenced transform group defined for type *type-name*. Packages that depend on a transform function associated with the named transform group are marked inoperative.

TRANSFORMS *group-name* **FOR** *type-name*
Indicates that the specified transform group for the user-defined data type *type-name* is to be dropped. The transform functions referenced in this group are not dropped. In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements, the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified object names. The *type-name* must identify a user-defined type described in the catalog (SQLSTATE 42704), and the *group-name* must identify an existing transform group for *type-name*.

TRIGGER *trigger-name*
Identifies the trigger that is to be dropped. The *trigger-name* must identify a trigger that is described in the catalog (SQLSTATE 42704). The specified trigger is deleted.

Dropping triggers causes certain packages to be marked invalid. See the "Notes" section in "CREATE TRIGGER" on page 780 concerning the creation of triggers (which follows the same rules).

TYPE *type-name*
Identifies the user-defined type to be dropped. In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified object names. For a structured type, the associated reference type is also dropped. The *type-name* must identify a user-defined type described in the

catalog. If DISTINCT is specified, then the *type-name* must identify a distinct type described in the catalog. The type is not dropped (SQLSTATE 42893) if any of the following are true.

- The type is used as the type of a column of a table or view.
- The type has a subtype.
- The type is a structured type used as the data type of a typed table or a typed view.
- The type is an attribute of another structured type.
- There exists a column of a table whose type might contain an instance of *type-name*. This can occur if *type-name* is the type of the column or is used elsewhere in the column's associated type hierarchy. More formally, for any type T, T cannot be dropped if there exists a column of a table whose type directly or indirectly uses *type-name*.
- The type is the target type of a reference-type column of a table or view, or a reference-type attribute of another structured type.
- The type or a reference to the type is a parameter type or a return value type of a function or method that cannot be dropped.
- The type, or a reference to the type, is used in the body of an SQL function or method, but it is not a parameter type or a return value type.
- The type is used in a check constraint, trigger, view definition, or index extension.

Functions that use the type: If the user-defined type can be dropped, then for every function, F (with specific name SF), that has parameters or a return value of the type being dropped or a reference to the type being dropped, the following DROP FUNCTION statement is effectively executed:

```
DROP SPECIFIC FUNCTION SF
```

It is possible that this statement also would cascade to drop dependent functions. If all of these functions are also in the list to be dropped because of a dependency on the user-defined type, the drop of the user-defined type will succeed (otherwise it fails with SQLSTATE 42893).

Methods that use the type: If the user-defined type can be dropped, then for every method, M of type T1 (with specific name SM), that has parameters or a return value of the type being dropped or a reference to the type being dropped, the following statements are effectively executed:

```
DROP SPECIFIC METHOD SM
ALTER TYPE T1 DROP SPECIFIC METHOD SM
```

The existence of objects that are dependent on these methods may cause the DROP TYPE to fail.

TYPE MAPPING *type-mapping-name*
Identifies the user-defined data type mapping to be dropped. The *type-mapping-name* must identify a data type mapping that is described in the catalog (SQLSTATE 42704). The data type mapping is deleted from the database.

No additional objects are dropped.

USER MAPPING FOR *authorization-name* | **USER SERVER** *server-name*
Identifies the user mapping to be dropped. This mapping associates an authorization name that is used to access the federated database with an authorization name that is used to access a data source. The first of these two authorization names is either identified by the *authorization-name* or referenced by the special register USER. The *server-name* identifies the data source that the second authorization name is used to access.

The *authorization-name* must be listed in the catalog (SQLSTATE 42704). The *server-name* must identify a data source that is described in the catalog (SQLSTATE 42704). The user mapping is deleted.

No additional objects are dropped.

VIEW *view-name*
Identifies the view that is to be dropped. The *view-name* must identify a view that is described in the catalog (SQLSTATE 42704). The subviews of a typed view are dependent on their superviews. All subviews must be dropped before a superview can be dropped (SQLSTATE 42893).

The specified view is deleted. The definition of any view or trigger that is directly or indirectly dependent on that view is marked inoperative. Any summary table that is dependent on any view that is marked inoperative is dropped. Any packages dependent on a view that is dropped or marked inoperative will be invalidated. This includes packages dependent on any superviews above the subview in the hierarchy. Any reference columns for which the dropped view is defined as the scope of the reference become unscoped.

VIEW HIERARCHY *root-view-name*
Identifies the typed view hierarchy that is to be dropped. The *root-view-name* must identify a typed view that is the root view in the typed view hierarchy (SQLSTATE 428DR). The typed view identified by *root-view-name* and all of its subviews are deleted from the database.

The definition of any view or trigger that is directly or indirectly dependent on any of the dropped views is marked inoperative. Any packages dependent on any view or trigger that is dropped or marked inoperative will be invalidated. Any reference columns for which a dropped view or view marked inoperative is defined as the scope of the reference become unscoped.

WRAPPER *wrapper-name*
: Identifies the wrapper to be dropped. The *wrapper-name* must identify a wrapper that is described in the catalog (SQLSTATE 42704). The wrapper is deleted.

  All server definitions, user-defined function mappings, and user-defined data type mappings that are dependent on the wrapper are dropped. All user-defined function mappings, nicknames, user-defined data type mappings, and user mappings that are dependent on the dropped server definitions are also dropped. Any index specifications dependent on the dropped nicknames are dropped, and any views dependent on these nicknames are marked inoperative. All packages dependent on the dropped objects and inoperative views are invalidated.

Rules

Dependencies: Table 27 on page 885 shows the dependencies[96] that objects have on each other.Four different types of dependencies are shown:

R Restrict semantics. The underlying object cannot be dropped as long as the object that depends on it exists.

C Cascade semantics. Dropping the underlying object causes the object that depends on it (the depending object) to be dropped as well. However, if the depending object cannot be dropped because it has a Restrict dependency on some other object, the drop of the underlying object will fail.

X Inoperative semantics. Dropping the underlying object causes the object that depends on it to become inoperative. It remains inoperative until a user takes some explicit action.

A Automatic Invalidation/Revalidation semantics. Dropping the underlying object causes the object that depends on it to become invalid. The database manager attempts to revalidate the invalid object.

Some DROP statement parameters and objects are not shown in Table 27 on page 885 because they would result in blank rows or columns:

- EVENT MONITOR, PACKAGE, PROCEDURE, SCHEMA, TYPE MAPPING, and USER MAPPING DROP statements do not have object dependencies.
- Alias, bufferpool, partitioning key, privilege, and procedure object types do not have DROP statement dependencies.
- A DROP SERVER, DROP FUNCTION MAPPING, or DROP TYPE MAPPING statement in a given unit of work (UOW) cannot be processed under either of the following conditions:

96. Not all dependencies are explicitly recorded in the catalog. For example, there is no record of which constraints a package has a dependency on.

- The statement references a single data source, and the UOW already includes a SELECT statement that references a nickname for a table or view within this data source (SQLSTATE 55006).
- The statement references a category of data sources (for example, all data sources of a specific type and version), and the UOW already includes a SELECT statement that references a nickname for a table or view within one of these data sources (SQLSTATE 55006).

Table 27. Dependencies

Object Type → / **Statement ↓**	**CONSTRAINT**	**FUNCTION**	**FUNC MAPPING**	**INDEX**	**INDEX EXTENSION**	**METHOD**	**NICKNAME**	**NODEGROUP**	**PACKAGE**	**SERVER**	**TABLE**	**TABLESPACE**	**TRIGGER**	**TYPE**	**TYPE MAPPING**	**USER MAPPING**	**VIEW**
ALTER NICKNAME	-	-	-	-	-	-	-	-	A	-	-	-	-	-	-	-	-
ALTER SERVER	-	-	-	-	-	-	-	-	A	-	-	-	-	-	-	-	-
ALTER TABLE DROP CONSTRAINT	C	-	-	-	-	-	-	-	A[1]	-	-	-	-	-	-	-	-
ALTER TABLE DROP PARTITIONING KEY	-	-	-	-	-	-	-	R[20]	A[1]	-	-	-	-	-	-	-	-
ALTER TYPE ADD ATTRIBUTE	-	-	-	-	R	-	-	-	A[23]	-	R[24]	-	-	-	-	-	R[14]
ALTER TYPE DROP ATTRIBUTE	-	-	-	-	R	-	-	-	A[23]	-	R[24]	-	-	-	-	-	R[14]
ALTER TYPE ADD METHOD	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
ALTER TYPE DROP METHOD	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
DROP ALIAS	-	R	-	-	-	-	-	-	A[3]	-	R[3]	-	X[3]	-	-	-	X[3]
DROP BUFFERPOOL	-	-	-	-	-	-	-	-	-	-	-	R	-	-	-	-	-

Table 27. Dependencies (continued)

Statement ↓ / Object Type →	CONSTRAINT	FUNCTION	FUNC MAPPING	INDEX	INDEX EXTENSION	METHOD	NICKNAME	NODEGROUP	PACKAGE	SERVER	TABLE	TABLESPACE	TRIGGER	TYPE	TYPE MAPPING	USER MAPPING	VIEW
DROP FUNCTION	R	R[7]	R	-	R	R[7]	-	-	X	-	R	-	R	-	-	-	R
DROP FUNCTION MAPPING	-	-	-	-	-	-	-	-	A	-	-	-	-	-	-	-	-
DROP INDEX	R	-	-	-	-	-	-	-	A	-	-	-	-	-	-	-	R[17]
DROP INDEX EXTENSION	-	R	-	R	-	-	-	-	-	-	-	-	-	-	-	-	-
DROP METHOD	R	R[7]	R	-	R	R	-	-	X	-	R	-	R	-	-	-	R
DROP NICKNAME	-	R	-	C	-	-	-	-	A	-	-	-	-	-	-	-	X[16]
DROP NODEGROUP	-	-	-	-	-	-	-	-	-	-	-	C	-	-	-	-	-
DROP SERVER	-	C[21]	C[19]	-	-	-	C	-	A	-	-	-	-	-	C[19]	C	-
DROP TABLE	C	R	-	C	-	-	-	-	A[9]	-	RC[11]	-	X[16]	-	-	-	X[16]
DROP TABLE HIERARCHY	C	R	-	C	-	-	-	-	A[9]	-	RC[11]	-	X[16]	-	-	-	X[16]
DROP TABLESPACE	-	-	-	C[6]	-	-	-	-	-	-	CR[6]	-	-	-	-	-	-
DROP TRANSFORM	-	R	-	-	-	-	-	-	X	-	-	-	-	-	-	-	-
DROP TRIGGER	-	-	-	-	-	-	-	-	A[1]	-	-	-	-	-	-	-	-
DROP TYPE	R[13]	R[5]	-	-	R	-	-	-	A[12]	-	R[18]	-	R[13]	R[4]	-	-	R[14]
DROP VIEW	-	R	-	-	-	-	-	-	A[2]	-	-	-	X[16]	-	-	-	X[15]
DROP VIEW HIERARCHY	-	R	-	-	-	-	-	-	A[2]	-	-	-	X[16]	-	-	-	X[16]
DROP WRAPPER	-	-	C	-	-	-	-	-	-	C	-	-	-	-	C	-	-
REVOKE a privilege[10]	-	CR[25]	-	-	-	-	-	-	A[1]	-	CX[8]	-	X	-	-	-	X[8]

1 This dependency is implicit in depending on a table with these constraints, triggers, or a partitioning key.

2 If a package has an INSERT, UPDATE, or DELETE statement acting upon a view, then the package has an insert, update or delete usage on the underlying base table of the view. In the case of UPDATE, the package has an update usage on each column of the underlying base table that is modified by the UPDATE.

If a package has a statement acting on a typed view, creating or dropping any view in the same view hierarchy will invalidate the package.

3 If a package, summary table, view, or trigger uses an alias, it becomes dependent both on the alias and the object that the alias references. If the alias is in a chain, then a dependency is created on each alias in the chain.

Aliases themselves are not dependent on anything. It is possible for an alias to be defined on an object that does not exist.

4 A user-defined type T can depend on another user-defined type B, if T:

- names B as the data type of an attribute
- has an attribute of REF(B)
- has B as a supertype.

5 Dropping a data type cascades to drop the functions and methods that use that data type as a parameter or a result type, and methods defined on the data type. Dropping of these functions and methods will not be prevented by the fact that they depend on each other. However, for functions or methods using the datatype within their bodies, restrict semantics apply.

6 Dropping a table space or a list of table spaces causes all the tables that are completely contained within the given table space or list to be dropped. However, if a table spans table spaces (indexes or long columns in different table spaces) and those table spaces are not in the list being dropped then the table space(s) cannot be dropped as long as the table exists.

7 A function can depend on another specific function if the depending function names the base function in a SOURCE clause. A function or method can also depend on another specific function or method if the depending routine is written in SQL and uses the base routine in its body. An external method, or an external function with a structured type parameter or returns type will also depend on one or more transform functions.

8 Only loss of SELECT privilege will cause a summary table to be

dropped or a view to become inoperative. If the view that is made inoperative is included in a typed view hierarchy, all of its subviews also become inoperative.

9 If a package has an INSERT, UPDATE, or DELETE statement acting on table T, then the package has an insert, update or delete usage on T. In the case of UPDATE, the package has an update usage on each column of T that is modified by the UPDATE.

If a package has a statement acting on a typed table, creating or dropping any table in the same table hierarchy will invalidate the package.

10 Dependencies do not exist at the column level because privileges on columns cannot be revoked individually.

If a package, trigger or view includes the use of OUTER(*Z*) in the FROM clause, there is a dependency on the SELECT privilege on every subtable or subview of *Z*. Similarly, if a package, trigger, or view includes the use of DEREF(*Y*) where *Y* is a reference type with a target table or view *Z*, there is a dependency on the SELECT privilege on every subtable or subview of *Z*.

11 A summary table is dependent on the underlying table or tables specified in the fullselect of the table definition.

Cascade semantics apply to dependent summary tables.

A subtable is dependent on its supertables up to the root table. A supertable cannot be dropped until all its subtables are dropped.

12 A package can depend on structured types as a result of using the TYPE predicate or the subtype-treatment expression (TREAT *expression* AS *data-type*). The package has a dependency on the subtypes of each structured type specified in the right side of the TYPE predicate, or the right side of the TREAT expression. Dropping or creating a structured type that alters the subtypes on which the package is dependent causes invalidation.

13 A check constraint or trigger is dependent on a type if the type is used anywhere in the constraint or trigger. There is no dependency on the subtypes of a structured type used in a TYPE predicate within a check constraint or trigger.

14 A view is dependent on a type if the type is used anywhere in the view definition (this includes the type of typed view). There is no dependency on the subtypes of a structured type used in a TYPE predicate within a view definition.

15 A subview is dependent on its superview up to the root view. A

superview cannot be dropped until all its subviews are dropped. Refer to [16] for additional view dependencies.

16 A trigger or view is also dependent on the target table or target view of a dereference operation or DEREF function. A trigger or view with a FROM clause that includes OUTER(*Z*) is dependent on all the subtables or subviews of *Z* that existed at the time the trigger or view was created.

17 A typed view can depend on the existence of a unique index to ensure the uniqueness of the object identifier column.

18 A table may depend on a user defined data type (distinct or structured) because the type is:

- used as the type of a column
- used as the type of the table
- used as an attribute of the type of the table
- used as the target type of a reference type that is the type of a column of the table or an attribute of the type of the table
- directly or indirectly used by a type that is the column of the table.

19 Dropping a server cascades to drop the function mappings and type mappings created for that named server.

20 If the partitioning key is defined on a table in a multiple partition nodegroup, the partitioning key is required.

21 If a dependent OLE DB table function has ″R″ dependent objects (see DROP FUNCTION), then the server cannot be dropped.

22 An SQL function or method can depend on the objects referenced by its body.

23 When an attribute A of type TA of *type-name* T is dropped, the following DROP statements are effectively executed:

```
Mutator method: DROP METHOD A (TA) FOR T
Observer method: DROP METHOD A () FOR T
ALTER TYPE T
   DROP METHOD A(TA)
   DROP METHOD A()
```

24 A table may depend on an attribute of a user-defined structured data type in the following cases:

1. The table is a typed table that is based on *type-name* or any of its subtypes.
2. The table has an existing column of a type that directly or indirectly refers to *type-name*.

25 A REVOKE of SELECT privilege on a table or view that is used in the body of an SQL function causes an attempt to drop the function, if the