CREATE PROCEDURE

This statement is used to register a stored procedure with an application server.

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 as least one of the following:

- SYSADM or DBADM authority
- IMPLICIT_SCHEMA authority on the database, if the implicit or explicit schema name of the procedure does not exist
- CREATEIN privilege on the schema, if the schema name of the procedure refers to an existing schema.

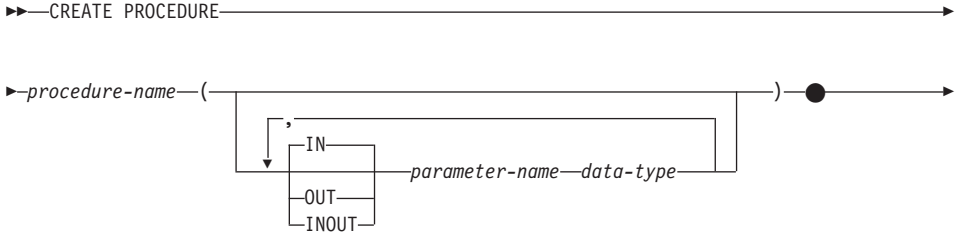
To create a not-fenced stored procedure, the privileges held by the authorization ID of the statement must also include at least one of the following:

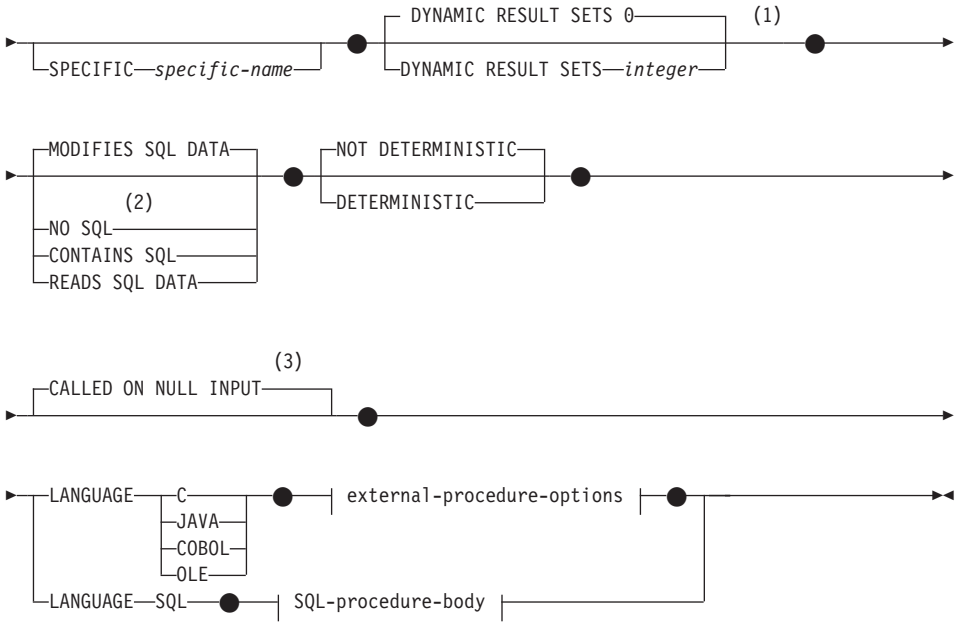
- CREATE_NOT_FENCED authority on the database
- SYSADM or DBADM authority.

To create a fenced stored procedure, no additional authorities or privileges are required.

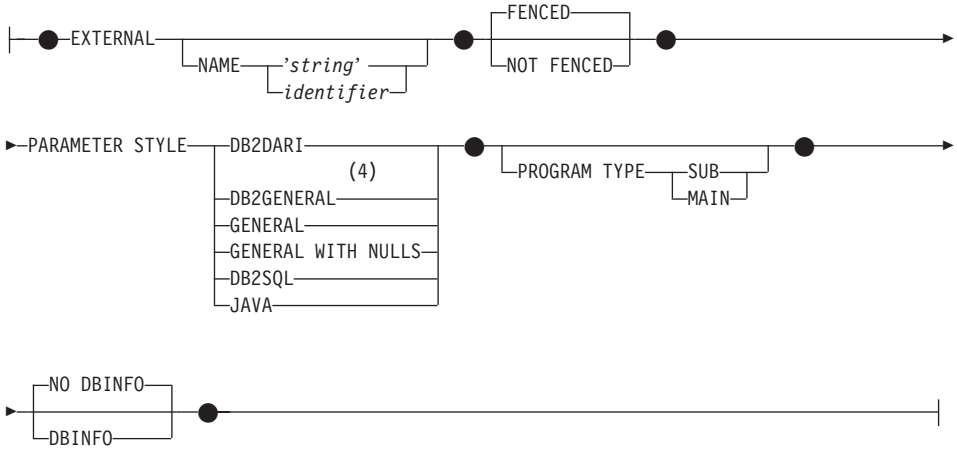
If the authorization ID has insufficient authority to perform the operation, an error (SQLSTATE 42502) is raised.

Syntax

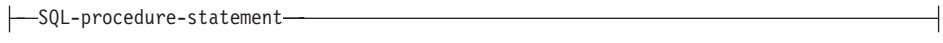




external-procedure-options:



SQL-procedure-body:



Notes:

1 RESULT SETS may be specified in place of DYNAMIC RESULT SETS.

2 NO SQL is not a valid choice for LANGUAGE SQL.

3 NULL CALL may be specified in place of CALLED ON NULL INPUT.

4 DB2GENRL may be specified in place of DB2GENERAL, SIMPLE CALL may be specified in place of GENERAL and SIMPLE CALL WITH NULLS may be specified in place of GENERAL WITH NULLS.

Description

procedure-name
Names the procedure being defined. It is a qualified or unqualified name that designates a procedure. The unqualified form of *procedure-name* is an SQL identifier (with a maximum length of 128). 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 qualified form is a *schema-name* followed by a period and an SQL identifier.

The name, including the implicit or explicit qualifiers, together with the number of parameters must not identify a procedure described in the catalog (SQLSTATE 42723). The unqualified name, together with the number of the parameters, while of course unique within its schema, need not be unique across schemas.

The a two-part name is specified, the *schema-name* cannot begin with "SYS". Otherwise, an error (SQLSTATE 42939) is raised.

(**IN** | **OUT** | **INOUT** *parameter-name data-type,...)*
Identifies the parameters of the procedure, and specifies the mode, name and data type of each parameter. One entry in the list must be specified for each parameter that the procedure will expect.

It is possible to register a procedure that has no parameters. In this case, the parentheses must still be coded, with no intervening data types. For example,

```
CREATE PROCEDURE SUBWOOFER() ...
```

No two identically-named procedures within a schema are permitted to have exactly the same number of parameters. Lengths, precisions and scales are not considered in this type comparison. Therefore CHAR(8) and CHAR(35) are considered to be the same type, as are DECIMAL(11,2) and DECIMAL (4,3). There is some further bundling of types that causes them to be treated as the same type for this purpose, such as DECIMAL and NUMERIC. A duplicate signature raises an SQL error (SQLSTATE 42723).

For example, given the statements:

```
CREATE PROCEDURE PART (IN NUMBER INT, OUT PART_NAME CHAR(35)) ...
CREATE PROCEDURE PART (IN COST DECIMAL(5,3), OUT COUNT INT) ...
```

the second statement will fail because the number of parameters of the procedure are the same even if the data types are not.

IN | OUT | INOUT
Specifies the mode of the parameter.

- IN - parameter is input only
- OUT - parameter is output only
- INOUT - parameter is both input and output

parameter-name
Specifies the name of the parameter.

data-type
Specifies the data type of the parameter.

- SQL data type specifications and abbreviations which may be specified in the *data-type* definition of a CREATE TABLE statement and have a correspondence in the language that is being used to write the procedure may be specified. See the language-specific sections of the *Application Development Guide* for details on the mapping between the SQL data types and host language data types with respect to stored procedures.
- User-defined data types are not supported (SQLSTATE 42601).

SPECIFIC *specific-name*
Provides a unique name for the instance of the procedure that is being defined. This specific name can be used when dropping the procedure or commenting on the procedure. It can never be used to invoke the procedure. The unqualified form of *specific-name* is an SQL identifier (with a maximum length of 18). The qualified form is a *schema-name* followed by a period and an SQL identifier. The name, including the implicit or explicit qualifier, must not identify another procedure instance that exists at the application server; otherwise an error (SQLSTATE 42710) is raised.

The *specific-name* may be the same as an existing *procedure-name*.

If no qualifier is specified, the qualifier that was used for *procedure-name* is used. If a qualifier is specified, it must be the same as the explicit or implicit qualifier of *procedure-name* or an error (SQLSTATE 42882) is raised.

If *specific-name* is not specified, a unique name is generated by the database manager. The unique name is SQL followed by a character timestamp, SQLyymmddhhmmsshhn.

DYNAMIC RESULT SETS *integer*
Indicates the estimated upper bound of returned result sets for the stored procedure. Refer to "Returning Result Sets from Stored Procedures" in the *SQL Reference* for more information.

The value RESULT SETS may be used as a synonym for DYNAMIC RESULT SETS for backwards and family compatibility.

NO SQL, CONTAINS SQL, READS SQL DATA, MODIFIES SQL DATA
Indicates whether the stored procedure issues any SQL statements and, if so, what type.

NO SQL
Indicates that the stored procedure cannot execute any SQL statements (SQLSTATE 38001).

CONTAINS SQL
Indicates that SQL statements that neither read nor modify SQL data can be executed by the stored procedure (SQLSTATE 38004 or 42985). Statements that are not supported in any stored procedure return a different error (SQLSTATE 38003 or 42985).

READS SQL DATA
Indicates that some SQL statements do not modify SQL data can be included in the stored procedure (SQLSTATE 38002 or 42985). Statements that are not supported in any stored procedure return a different error (SQLSTATE 38003 or 42985).

MODIFIES SQL DATA
Indicates that the stored procedure can execute any SQL statement except statements that are not supported in stored procedures (SQLSTATE 38003 or 42985).

The following table indicates whether or not an SQL statement (specified in the first column) is allowed to execute in a stored procedure with the specified SQL data access indication. If an executable SQL statement is encountered in a stored procedure defined with NO SQL, SQLSTATE 38001 is returned. For other executions contexts, SQL statements that are not supported in any context return SQLSTATE 38003. For other SQL statements not allowed in a CONTAINS SQL context, SQLSTATE 38004 is returned and in a READS SQL DATA context, SQLSTATE 38002 is returned. During creation of an SQL procedure, a statement that does not match the SQL data access indication will cause SQLSTATE 42895 to be returned.

Table 21. SQL Statement and SQL Data Access Indication

SQL Statement	NO SQL	CONTAINS SQL	READS SQL DATA	MODIFIES SQL DATA
ALTER...	N	N	N	Y
BEGIN DECLARE SECTION	Y(1)	Y	Y	Y
CALL	N	Y(4)	Y(4)	Y(4)
CLOSE CURSOR	N	N	Y	Y
COMMENT ON	N	N	N	Y
COMMIT	N	N	N	N
COMPOUND SQL	N	Y	Y	Y
CONNECT(2)	N	N	N	N
CREATE	N	N	N	Y
DECLARE CURSOR	Y(1)	Y	Y	Y
DECLARE GLOBAL TEMPORARY TABLE	N	Y	Y	Y
DELETE	N	N	N	Y
DESCRIBE	N	N	Y	Y
DISCONNECT(2)	N	N	N	N
DROP ...	N	N	N	Y
END DECLARE SECTION	Y(1)	Y	Y	Y
EXECUTE	N	Y(3)	Y(3)	Y
EXECUTE IMMEDIATE	N	Y(3)	Y(3)	Y
EXPLAIN	N	N	N	Y
FETCH	N	N	Y	Y
FREE LOCATOR	N	Y	Y	Y
FLUSH EVENT MONITOR	N	N	N	Y
GRANT ...	N	N	N	Y
INCLUDE	Y(1)	Y	Y	Y
INSERT	N	N	N	Y
LOCK TABLE	N	Y	Y	Y
OPEN CURSOR	N	N	Y	Y
PREPARE	N	Y	Y	Y
REFRESH TABLE	N	N	N	Y

Table 21. SQL Statement and SQL Data Access Indication (continued)

SQL Statement	NO SQL	CONTAINS SQL	READS SQL DATA	MODIFIES SQL DATA
RELEASE CONNECTION(2)	N	N	N	N
RELEASE SAVEPOINT	N	N	N	Y
RENAME TABLE	N	N	N	Y
REVOKE ...	N	N	N	Y
ROLLBACK	N	Y	Y	Y
ROLLBACK TO SAVEPOINT	N	N	N	Y
SAVEPOINT	N	N	N	Y
SELECT INTO	N	N	Y	Y
SET CONNECTION(2)	N	N	N	N
SET INTEGRITY	N	N	N	Y
SET special register	N	Y	Y	Y
UPDATE	N	N	N	Y
VALUES INTO	N	N	Y	Y
WHENEVER	Y(1)	Y	Y	Y

Notes:

1. Although the NO SQL option implies that no SQL statements can be specified, non-executable statements are not restricted.
2. Connection management statements are not allowed in any stored procedure execution contexts.
3. It depends on the statement being executed. The statement specified for the EXECUTE statement must be a statement that is allowed in the context of the particular SQL access level in effect. For example, if the SQL access level in effect is READS SQL DATA, the statement must not be an INSERT, UPDATE, or DELETE.
4. A CALL statement in a stored procedure can only refer to a stored procedure written in the same programming language as the calling stored procedure.

LANGUAGE

This mandatory clause is used to specify the language interface convention to which the stored procedure body is written.

C This means the database manager will call the stored procedure as

if it were a C procedure. The stored procedure must conform to the C language calling and linkage convention as defined by the standard ANSI C prototype.

JAVA This means the database manager will call the stored procedure as a method in a Java class.

COBOL
This means the database manager will call the procedure as if it were a COBOL procedure.

OLE This means the database manager will call the stored procedure as if it were a method exposed by an OLE automation object. The stored-procedure must conform with the OLE automation data types and invocation mechanism. Also, the OLE automation object needs to be implemented as an in-process server (DLL). These restrictions are outlined in the OLE Automation Programmer's Reference.

LANGUAGE OLE is only supported for stored procedures stored in DB2 for Windows 32-bit operating systems.

SQL The specified *SQL-procedure-body* includes the statements which define the processing of the stored procedure

EXTERNAL
This clause indicates that the CREATE PROCEDURE statement is being used to register a new procedure based on code written in an external programming language and adhering to the documented linkage conventions and interface.

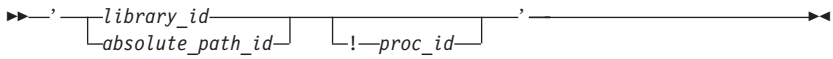
If NAME clause is not specified "NAME *procedure-name*" is assumed.

NAME *'string'*
This clause identifies the name of the user-written code which implements the procedure being defined.

The 'string' option is a string constant with a maximum of 254 characters. The format used for the string is dependent on the LANGUAGE specified.

- For LANGUAGE C:

  The *string* specified is the library name and procedure within the library, which the database manager invokes to execute the stored procedure being CREATEd. The library (and the procedure within the library) do not need to exist when the CREATE PROCEDURE statement is performed. However, when the procedure is called, the library and procedure within the library must exist and be accessible from the database server machine.



The name must be enclosed in single quotes. Extraneous blanks are not permitted within the single quotes.

library_id
Identifies the library name containing the procedure. The database manager will look for the library in the .../sqllib/function/unfenced directory and the .../sqllib/function directory (UNIX-based systems), or ...*instance_name*\function\unfenced directory and the ...*instance_name*\function directory (OS/2, Windows 32-bit operating systems as specified by the DB2INSTPROF registry variable), where the database manager will locate the controlling sqllib directory which is being used to run the database manager. For example, the controlling sqllib directory in UNIX-based systems is /u/$DB2INSTANCE/sqllib.

If 'myproc' were the *library_id* in a UNIX-based system it would cause the database manager to look for the procedure in library /u/production/sqllib/function/unfenced/myfunc and /u/production/sqllib/function/myfunc, provided the database manager is being run from /u/production.

For OS/2, Windows 32-bit operating systems, the database manager will look in the LIBPATH or PATH if the *library_id* is not found in the function directory, and will be run as fenced.

Stored procedures located in any of these directories do not use any of the registered attributes.

absolute_path_id
Identifies the full path name of the procedure.

In a UNIX-based system, for example, '/u/jchui/mylib/myproc' would cause the database manager to look in /u/jchui/mylib for the myproc procedure.

In OS/2, Windows 32-bit operating systems 'd:\mylib\myproc' would cause the database manager to load the myproc.dll file from the d:\mylib directory.

If an absolute path is specified, the procedure will run as fenced, ignoring the FENCED or NOT FENCED attribute.

! proc_id
Identifies the entry point name of the procedure to be invoked. The ! serves as a delimiter between the library id and the

procedure id. If *! proc_id* is omitted, the database manager will use the default entry point established when the library was linked.

In a UNIX-based system, for example, 'mymod!proc8' would direct the database manager to look for the library $inst_home_dir/sqllib/function/mymod and to use entry point proc8 within that library.

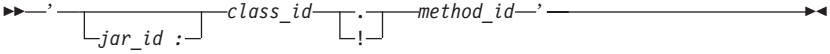
In OS/2, Windows 32-bit operating systems 'mymod!proc8' would direct the database manager to load the mymod.dll file and call the proc8() procedure in the dynamic link library (DLL).

If the string is not properly formed, an error (SQLSTATE 42878) is raised.

The body of every stored procedure should be in a directory which is mounted and available on every partition of the database.

- For LANGUAGE JAVA:

  The *string* specified contains the optional jar file identifier, class identifier and method identifier, which the database manager invokes to execute the stored procedure being CREATEd. The class identifier and method identifier do not need to exist when the CREATE PROCEDURE statement is performed. If a *jar_id* is specified, it must exist when the CREATE PROCEDURE statement is performed. However, when the procedure is called, the class identifier and the method identifier must exist and be accessible from the database server machine, otherwise an error (SQLSTATE 42884) is raised.



The name must be enclosed in single quotes. Extraneous blanks are not permitted within the single quotes.

jar_id
Identifies the jar identifier given to the jar collection when it was installed in the database. It can be either a simple identifier, or a schema qualified identifier. Examples are 'myJar' and 'mySchema.myJar'.

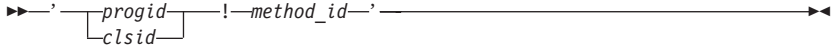
class_id
Identifies the class identifier of the Java object. If the class is part of a package, the class identifier part must include the complete package prefix, for example, 'myPacks.StoredProcs'.

The Java virtual machine will look in directory '../myPacks/StoredProcs/' for the classes. In OS/2 and Windows 32-bit operating systems, the Java virtual machine will look in directory '..\myPacks\StoredProcs\'.

method_id
Identifies the method name with the Java class to be invoked.

- For LANGUAGE OLE:

  The string specified is the OLE programmatic identifier (*progid*) or class identifier (*clsid*), and method identifier (*method_id*), which the database manager invokes to execute the stored procedure being created by the statement. The programmatic identifier or class identifier, and the method identifier do not need to exist when the CREATE PROCEDURE statement is executed. However, when the procedure is used in the CALL statement, the method identifier must exist and be accessible from the database server machine, otherwise an error results (SQLSTATE 42724).



The name must be enclosed in single quotes. Extraneous blanks are not permitted within the single quotes.

progid
Identifies the programmatic identifier of the OLE object.

A *progid* is not interpreted by the database manager, but only forwarded to the OLE automation controller at run time. The specified OLE object must be creatable and support late binding (also known as IDispatch-based binding). By convention, progids have the following format:

<program_name>.<component_name>.<version>

Since it is only a convention, and not a rule, *progids* may in fact have a different format.

clsid
Identifies the class identifier of the OLE object to create. It can be used as an alternative for specifying a progid in the case that an OLE object is not registered with a progid. The *clsid* has the form:

{nnnnnnnn-nnnn-nnnn-nnnn-nnnnnnnnnnnn}

where 'n' is an alphanumeric character. A *clsid* is not interpreted by the database manager but only forwarded to the OLE APIs at run time.

method_id
Identifies the method name of the OLE object to be invoked.

NAME *identifier*
This *identifier* specified is an SQL identifier. The SQL identifier is used as the *library-id* in the string. Unless it is a delimited identifier, the identifier is folded to upper case. If the identifier is qualified with a schema name, the schema name portion is ignored. This form of NAME can only be used with LANGUAGE C.

FENCED or **NOT FENCED**
This clause specifies whether or not the stored procedure is considered "safe" to run in the database manager operating environment's process or address space (NOT FENCED), or not (FENCED).

If a stored procedure is registered as FENCED, the database manager insulates its internal resources (e.g. data buffers) from access by the procedure. All procedures have the option of running as FENCED or NOT FENCED. In general, a procedure running as FENCED will not perform as well as a similar one running as NOT FENCED.

If the stored procedure is located in .../sqllib/function/unfenced directory and the .../sqllib/function directory (UNIX-based systems), or ...*instance_name*\function\unfenced directory and the ...*instance_name*\function directory (OS/2, Windows 32-bit operating systems), then the FENCED or NOT FENCED registered attribute (and every other registered attribute) will be ignored.

Note: Use of NOT FENCED for procedures not adequately checked out can compromise the integrity of DB2. DB2 takes some precautions against many of the common types of inadvertent failures that might occur, but cannot guarantee complete integrity when NOT FENCED stored procedures are used.

To change from FENCED to NOT FENCED, the procedure must be re-registered (by first dropping it and then re-creating it). Either SYSADM authority, DBADM authority or a special authority (CREATE_NOT_FENCED) is required to register a stored procedures as NOT FENCED. Only FENCED can be specified for a stored procedure with LANGUAGE OLE.

PARAMETER STYLE
This clause is used to specify the conventions used for passing parameters to and returning the value from stored procedures.

DB2DARI
This means that the stored procedure will use a parameter

passing convention that conforms to C language calling and linkage conventions. This can only be specified when LANGUAGE C is used.

DB2GENERAL
This means that the stored procedure will use a parameter passing convention that is defined for use with Java methods. This can only be specified when LANGUAGE JAVA is used.

The value DB2GENRL may be used as a synonym for DB2GENERAL.

GENERAL
This means that the stored procedure will use a parameter passing mechanism where the stored procedure receives the parameters specified on the CALL. The parameters are passed directly as expected by the language, the SQLDA structure is not used. This can only be specified when LANGUAGE C or COBOL is used.

Null indicators are NOT directly passed to the program.

The value SIMPLE CALL may be used as a synonym for GENERAL.

GENERAL WITH NULLS
In addition to the parameters on the CALL statement as specified in GENERAL, another argument is passed to the stored procedure. This additional argument contains a vector of null indicators for each of the parameters on the CALL statement. In C, this would be an array of short ints. This can only be specified when LANGUAGE C or COBOL is used.

The value SIMPLE CALL WITH NULLS may be used as a synonym for GENERAL WITH NULLLS.

DB2SQL
In addition to the parameters on the CALL statement, the following arguments are passed to the stored procedure:

- a NULL indicator for each parameter on the CALL statement
- the SQLSTATE to be returned to DB2
- the qualified name of the stored procedure
- the specific name of the stored procedure
- the SQL diagnostic string to be returned to DB2

This can only be specified when LANGUAGE C, COBOL or OLE is used.

JAVA This means that the stored procedure will use a parameter

passing convention that conforms to the Java language and SQLJ Routines specification. IN/OUT and OUT parameters will be passed as single entry arrays to facilitate returning values. This can only be specified when LANGUAGE JAVA is used.

PARAMETER STYLE JAVA procedures do not support the DBINFO or PROGRAM TYPE clauses.

Refer to *Application Development Guide* for details on passing parameters.

PROGRAM TYPE
Specifies whether the stored procedure expects parameters in the style of a main routine or a subroutine.

SUB
The stored procedure expects the parameters to be passed as separate arguments.

MAIN
The stored procedure expects the parameters to be passed as an argument counter, and a vector of arguments (argc, argv). The name of the stored procedure to be invoked must also be "main". Stored procedures of this type must still be built in the same fashion as a shared library as opposed to a stand-alone executable.

The default for PROGRAM TYPE is SUB. PROGRAM TYPE MAIN is only valid for LANGUAGE C or COBOL and PARAMETER STYLE GENERAL, GENERAL WITH NULLS or DB2SQL.

DETERMINISTIC or **NOT DETERMINISTIC**
This clause specifies whether the procedure always returns the same results for given argument values (DETERMINISTIC) or whether the procedure depends on some state values that affect the results (NOT DETERMINISTIC). That is, a DETERMINISTIC procedure must always return the same result from successive invocations with identical inputs.

This clause currently does not impact processing of the stored procedure.

CALLED ON NULL INPUT
CALLED ON NULL INPUT always applies to stored procedures. This means that regardless if any arguments are null, the stored procedure is called. It can return a null value or a normal (non-null) value. Responsibility for testing for null argument values lies with the stored procedure.

The value NULL CALL may be used as a synonym for CALLED ON NULL INPUT for backwards and family compatibility.

NO DBINFO or DBINFO
Specifies whether specific information known by DB2 is passed to the stored procedure when it is invoked as an additional invocation-time argument (DBINFO) or not (NO DBINFO). NO DBINFO is the default. DBINFO is not supported for LANGUAGE OLE (SQLSTATE 42613). It is also not supported for PARAMETER STYLE JAVA, DB2GENERAL, or DB2DARI.

If DBINFO is specified, then a structure is passed to the stored procedure which contains the following information:

- Data base name - the name of the currently connected database.
- Application ID - unique application ID which is established for each connection to the database.
- Application Authorization ID - the application run-time authorization ID.
- Code page - identifies the database code page.
- Schema name - not applicable to stored procedures.
- Table name - not applicable to stored procedures.
- Column name - not applicable to stored procedures.
- Database version/release - identifies the version, release and modification level of the database server invoking the stored procedure.
- Platform - contains the server's platform type.
- Table function result column numbers - not applicable to stored procedures.

Please see the *Application Development Guide* for detailed information on the structure and how it is passed to the stored procedure.

SQL-procedure-body
Specifies the SQL statement that is the body of the SQL procedure. Multiple SQL-procedure-statements may be specified within a compound statement. See "Chapter 7. SQL Procedures" on page 1059 for more information.

Notes

- For information on creating the programs for a stored procedure, see the *Application Development Guide*.
- Creating a procedure with a schema name that does not already exist will result in the implicit creation of that schema provided the authorization ID of the statement has IMPLICIT_SCHEMA authority. The schema owner is SYSIBM. The CREATEIN privilege on the schema is granted to PUBLIC.

- The settings of the special registers of the caller are inherited by the stored procedure on invocation and restored upon return to the caller. Special registers may be changed within a stored procedure, but these changes do not effect the caller. This is not true for legacy stored procedures (those defined with parameter style DB2DARI or stored in the default library), where the changes made to special registers in a procedure become the settings for the caller.

Examples

Example 1: Create the procedure definition for a stored procedure, written in Java, that is passed a part number and returns the cost of the part and the quantity that are currently available.

```
CREATE PROCEDURE PARTS_ON_HAND (IN  PARTNUM  INTEGER,
                                OUT COST     DECIMAL(7,2),
                                OUT QUANTITY INTEGER)
       EXTERNAL NAME 'parts.onhand'
       LANGUAGE JAVA PARAMETER STYLE JAVA
```

Example 2: Create the procedure definition for a stored procedure, written in C, that is passed an assembly number and returns the number of parts that make up the assembly, total part cost and a result set that lists the part numbers, quantity and unit cost of each part.

```
CREATE PROCEDURE ASSEMBLY_PARTS (IN  ASSEMBLY_NUM  INTEGER,
                                 OUT NUM_PARTS     INTEGER,
                                 OUT COST DOUBLE)
       EXTERNAL NAME 'parts!assembly'
       DYNAMIC RESULT SETS 1 NOT FENCED
       LANGUAGE C PARAMETER STYLE GENERAL
```

Example 3: Create an SQL procedure that returns the median staff salary. Return a result set containing the name, position, and salary of all employees who earn more than the median salary.

```
CREATE PROCEDURE MEDIAN_RESULT_SET
(OUT medianSalary DOUBLE)
     RESULT SETS 1
     LANGUAGE SQL
BEGIN
     DECLARE v_numRecords INT DEFAULT 1;
     DECLARE v_counter INT DEFAULT 0;

     DECLARE c1 CURSOR FOR
        SELECT CAST(salary AS DOUBLE)
           FROM staff
           ORDER BY salary;
     DECLARE c2 CURSOR WITH RETURN FOR
        SELECT name, job, CAST(salary AS INTEGER)
           FROM staff
           WHERE salary > medianSalary
           ORDER BY salary;
     DECLARE EXIT HANDLER FOR NOT FOUND
```

```
      SET medianSalary = 6666;
    SET medianSalary = 0;
    SELECT COUNT(*) INTO v_numRecords
      FROM STAFF;
    OPEN c1;
    WHILE v_counter < (v_numRecords / 2 + 1)
      DO FETCH c1 INTO medianSalary;
      SET v_counter = v_counter + 1;
    END WHILE;
    CLOSE c1;
    OPEN c2;
END
```

CREATE SCHEMA

The CREATE SCHEMA statement defines a schema. It is also possible to create some objects and grant privileges on objects within the statement.

Invocation

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

Authorization

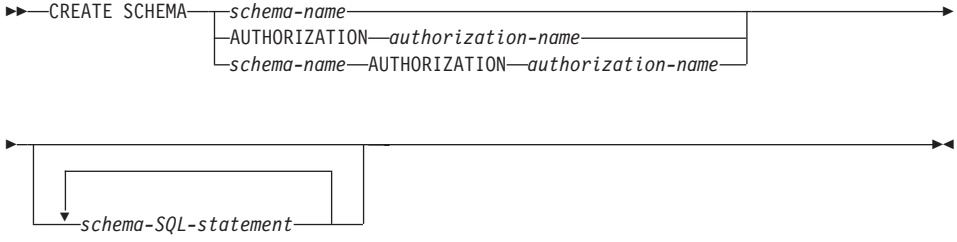
An authorization ID that holds SYSADM or DBADM authority can create a schema with any valid *schema-name* or *authorization-name*.

An authorization ID that does not hold SYSADM or DBADM authority can only create a schema with a *schema-name* or *authorization-name* that matches the authorization ID of the statement.

If the statement includes any *schema-SQL-statement*s the privileges held by the *authorization-name* (if not specified, it defaults to the authorization ID of the statement) must include at least one of the following:

- The privileges required to perform each of the *schema-SQL-statement*s
- SYSADM or DBADM authority.

Syntax



Description

schema-name
Names the schema. The name must not identify a schema already described in the catalog (SQLSTATE 42710). The name cannot begin with "SYS" (SQLSTATE 42939). The owner of the schema is the authorization ID that issued the statement.

AUTHORIZATION *authorization-name*
Identifies the user that is the owner of the schema. The value

authorization-name is also used to name the schema. The *authorization-name* must not identify a schema already described in the catalog (SQLSTATE 42710).

schema-name **AUTHORIZATION** *authorization-name*
Identifies a schema called *schema-name* with the user called *authorization-name* as the schema owner. The *schema-name* must not identify a schema-name for a schema already described in the catalog (SQLSTATE 42710). The *schema-name* cannot begin with "SYS" (SQLSTATE 42939).

schema-SQL-statement
SQL statements that can be included as part of the CREATE SCHEMA statement are:

- CREATE TABLE statement excluding typed tables and summary tables (see "CREATE TABLE" on page 712)
- CREATE VIEW statement excluding typed views (see "CREATE VIEW" on page 823)
- CREATE INDEX statement (see "CREATE INDEX" on page 662)
- COMMENT ON statement (see "COMMENT ON" on page 532)
- GRANT statement (see "GRANT (Table, View, or Nickname Privileges)" on page 926).

Notes

- The owner of the schema is determined as follows:
  - If an AUTHORIZATION clause is specified, the specified *authorization-name* is the schema owner
  - If an AUTHORIZATION clause is not specified, the authorization ID that issued the CREATE SCHEMA statement is the schema owner.
- The schema owner is assumed to be a user (not a group).
- When the schema is explicitly created with the CREATE SCHEMA statement, the schema owner is granted CREATEIN, DROPIN, and ALTERIN privileges on the schema with the ability to grant these privileges to other users.
- The definer of any object created as part of the CREATE SCHEMA statement is the schema owner. The schema owner is also the grantor for any privileges granted as part of the CREATE SCHEMA statement.
- Unqualified object names in any SQL statement within the CREATE SCHEMA statement are implicitly qualified by the name of the created schema.
- If the CREATE statement contains a qualified name for the object being created, the schema name specified in the qualified name must be the same

as the name of the schema being created (SQLSTATE 42875). Any other objects referenced within the statements may be qualified with any valid schema name.

- If the AUTHORIZATION clause is specified and DCE authentication is used, the group membership of the *authorization-name* specified will not be considered in evaluating the authorizations required to perform the statements that follow the clause. If the *authorization-name* specified is different than the authorization id creating the schema, an authorization failure may result during the execution of the CREATE SCHEMA statement.
- It is recommended not to use ″SESSION″ as a schema name. Since declared temporary tables must be qualified by ″SESSION″, it is possible to have an application declare a temporary table with a name identical to that of a persistent table. An SQL statement that references a table with the schema name ″SESSION″ will resolve (at statement compile time) to the declared temporary table rather than a persistent table with the same name. Since an SQL statement is compiled at different times for static embedded and dynamic embedded SQL statements, the results depend on when the declared temporary table is defined. If persistent tables, views or aliases are not defined with a schema name of ″SESSION″, these issues do not require consideration.

Examples

Example 1: As a user with DBADM authority, create a schema called RICK with the user RICK as the owner.

```
CREATE SCHEMA RICK AUTHORIZATION RICK
```

Example 2: Create a schema that has an inventory part table and an index over the part number. Give authority on the table to user JONES.

```
CREATE SCHEMA INVENTRY

   CREATE TABLE PART (PARTNO   SMALLINT NOT NULL,
                      DESCR    VARCHAR(24),
                      QUANTITY INTEGER)

   CREATE INDEX PARTIND ON PART (PARTNO)

   GRANT ALL ON PART TO JONES
```

Example 3: Create a schema called PERS with two tables that each have a foreign key that references the other table. This is an example of a feature of the CREATE SCHEMA statement that allows such a pair of tables to be created without the use of the ALTER TABLE statement.

```
CREATE SCHEMA PERS

   CREATE TABLE ORG (DEPTNUMB  SMALLINT NOT NULL,
                     DEPTNAME VARCHAR(14),
                     MANAGER  SMALLINT,
```

```
                        DIVISION VARCHAR(10),
                        LOCATION VARCHAR(13),
                        CONSTRAINT PKEYDNO
                           PRIMARY KEY (DEPTNUMB),
                        CONSTRAINT FKEYMGR
                           FOREIGN KEY (MANAGER)
                           REFERENCES STAFF (ID) )

CREATE TABLE STAFF (ID        SMALLINT NOT NULL,
                    NAME      VARCHAR(9),
                    DEPT      SMALLINT,
                    JOB       VARCHAR(5),
                    YEARS     SMALLINT,
                    SALARY    DECIMAL(7,2),
                    COMM      DECIMAL(7,2),
                    CONSTRAINT PKEYID
                       PRIMARY KEY (ID),
                    CONSTRAINT FKEYDNO
                       FOREIGN KEY (DEPT)
                       REFERENCES ORG (DEPTNUMB) )
```

CREATE SERVER

The CREATE SERVER statement[73] defines a data source to a federated database.

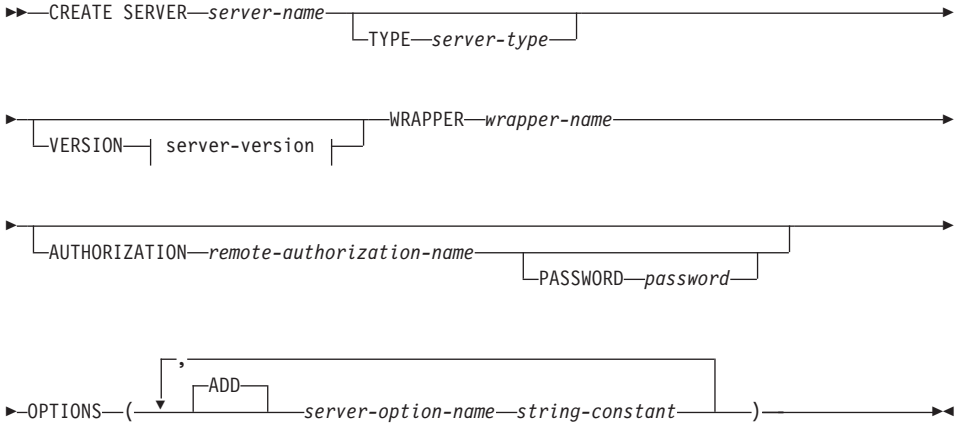
Invocation

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

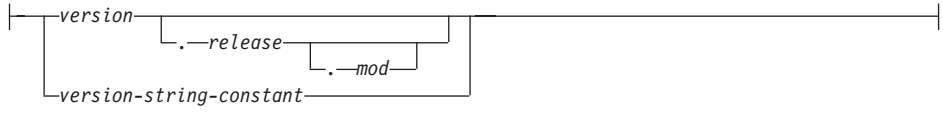
Authorization

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

Syntax



server-version:



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

Description

server-name
Names the data source that is being defined to the federated database. The name must not identify a data source that is described in the catalog. The *server-name* must not be the same as the name of any table space in the federated database.

TYPE *server-type*
Specifies the type of the data source denoted by *server-name*. This option is required with the DRDA, SQLNET, and NET8 wrappers. Refer to "Appendix F. Federated Systems" on page 1245 for a list of supported data source types.

VERSION
Specifies the version of the data source denoted by *server-name.*

version
Specifies the version number. *version* must be an integer.

release
Specifies the number of the release of the version denoted by *version. release* must be an integer.

mod
Specifies the number of the modification of the release denoted by *release. mod* must be an integer.

version-string-constant
Specifies the complete designation of the version. The *version-string-constant* can be a single value (for example, '8i'); or it can be the concatenated values of *version*, *release*, and, if applicable, *mod* (for example, '8.0.3').

WRAPPER *wrapper-name*
Names the wrapper that the federated server uses to interact with data sources of the type and version denoted by *server-type* and *'server-version'*.

AUTHORIZATION *remote-authorization-name*
Specifies the authorization ID under which any necessary actions are performed at the data source when the CREATE SERVER statement is processed. This ID must hold the authority (BINDADD or its equivalent) that the necessary actions require.

PASSWORD *password*
Specifies the password associated with the authorization ID represented by *remote-authorization-name.* If *password* is not specified, it will default to the password for the ID under which the user is connected to the federated database.

OPTIONS
Indicates what server options are to be enabled. Refer to "Server Options" on page 1249 for descriptions of *server-option-name*s and their settings.

ADD
Enables one or more server options.

server-option-name
Names a server option that will be used to either configure or provide information about the data source denoted by *server-name*.

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

Notes

- If *remote-authorization-name* is not specified, the authorization ID for the federated database will be used.
- The *password* should be specified in the case required by the data source; if any letters in *password* must be in lowercase, enclose *password* in quotation marks. If an *identifier* is specified but not *password*, the authentication type of the data source denoted by *server-name* is assumed to be CLIENT.
- If the CREATE SERVER statement is used to define a DB2 family instance as a data source, DB2 may need to bind certain packages to that instance. If a bind is required, the *remote-authorization-name* in the statement must have BIND authority. The time required for the bind to complete is dependent on data source speed and network connection speed.
- If a server option is set to one value for a type of data source, and this same option set to another value for an instance of this type, the second value overrides the first for the instance. For example, suppose that PASSWORD is set to 'Y' (yes, validate passwords at the data source) for a federated system's DB2 Universal Database for OS/390 data sources. Then later, this option's default ('N') is used for a specific DB2 Universal Database for OS/390 data source named SIBYL. As a result, passwords will be validated at all of the DB2 Universal Database for OS/390 data sources except SIBYL.

Examples

Example 1: Define a DB2 for MVS/ESA 4.1 data source that is accessible through a wrapper called DB2WRAP. Call the data source CRANDALL. In addition, specify that:

- MURROW and DROWSSAP will be the authorization ID and password under which packages are bound at CRANDALL when this statement is processed.
- CRANDALL is defined to the DB2 RDBMS as an instance called MYNODE.

- When the federated server accesses CRANDALL, it will be connected to a database called MYDB.
- The authorization IDs and passwords under which CRANDALL can be accessed are to be sent to CRANDALL in uppercase.
- MYDB and the federated database use the same collating sequence.

```
CREATE SERVER CRANDALL
   TYPE DB2/MVS
   VERSION 4.1
   WRAPPER DB2WRAP
   AUTHORIZATION MURROW
   PASSWORD DROWSSAP
   OPTIONS
       ( NODE 'MYNODE',
       DBNAME 'MYDB',
       FOLD_ID 'U',
       FOLD_PW 'U',
       COLLATING_SEQUENCE 'Y' )
```

Example 2: Define an Oracle 7.2 data source that's accessible through a wrapper called KLONDIKE. Call the data source CUSTOMERS. Specify that:

- CUSTOMERS is defined to the Oracle RDBMS as an instance called ABC.

Provide these statistics for the optimizer:

- The CPU for the federated server runs twice as fast as the CPU that supports CUSTOMERS.
- The I/O devices at the federated server process data one and a half times as fast as the I/O devices at CUSTOMERS.

```
CREATE SERVER CUSTOMERS
   TYPE ORACLE
   VERSION 7.2
   WRAPPER KLONDIKE
   OPTIONS
       ( NODE 'ABC',
       CPU_RATIO '2.0',
       IO_RATIO '1.5' )
```

CREATE TABLE

The CREATE TABLE statement defines a table. The definition must include its name and the names and attributes of its columns. The definition may include other attributes of the table, such as its primary key or check constraints.

Invocation

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

Authorization

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

- SYSADM or DBADM authority
- CREATETAB authority on the database and USE privilege on the table space as well as one of:
  - IMPLICIT_SCHEMA authority on the database, if the implicit or explicit schema name of the table does not exist
  - CREATEIN privilege on the schema, if the schema name of the table refers to an existing schema.

  If a subtable is being defined, the authorization ID must be the same as the definer of the root table of the table hierarchy.

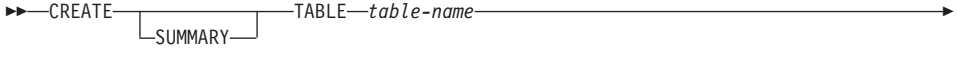
To define a foreign key, the privileges held by the authorization ID of the statement must include one of the following on the parent table:

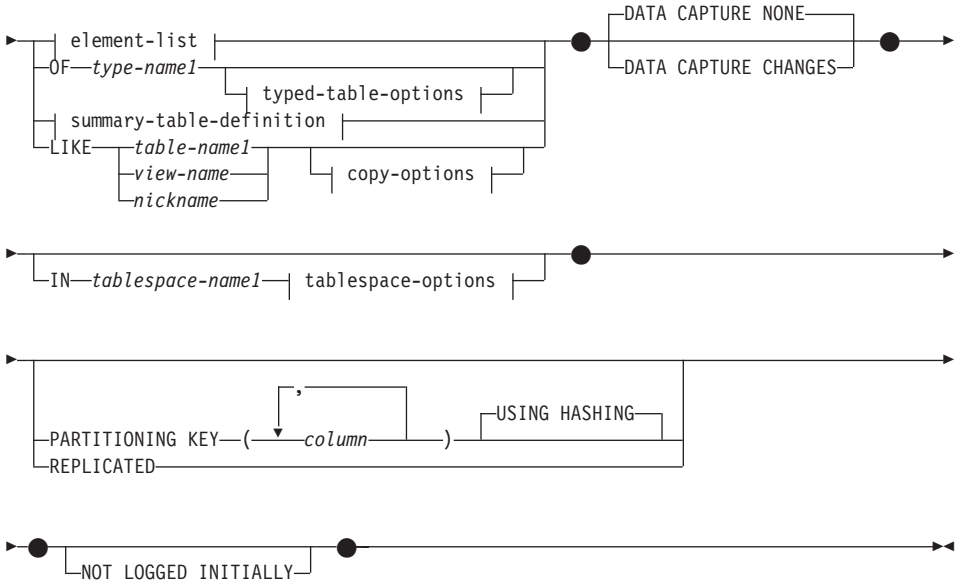
- REFERENCES privilege on the table
- REFERENCES privilege on each column of the specified parent key
- CONTROL privilege on the table
- SYSADM or DBADM authority.

To define a summary table (using a fullselect) the privileges held by the authorization ID of the statement must include at least one of the following on each table or view identified in the fullselect:

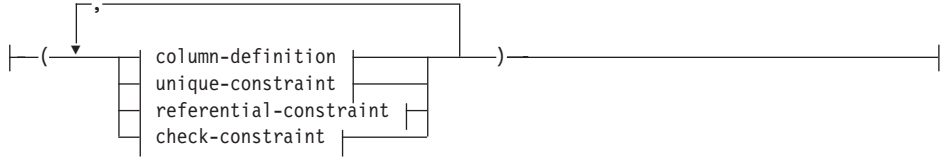
- SELECT privilege on the table or view and ALTER privilege if REFRESH DEFERRED or REFRESH IMMEDIATE is specified
- CONTROL privilege on the table or view
- SYSADM or DBADM authority.

Syntax

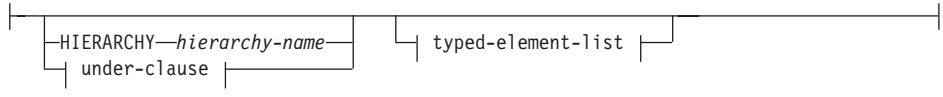




element-list:

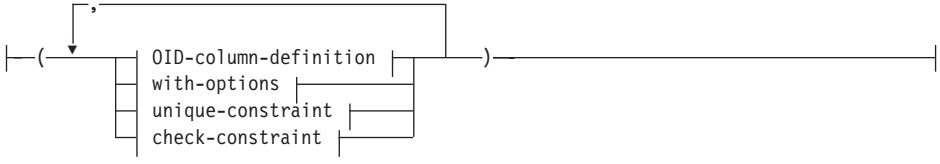


typed-table-options:

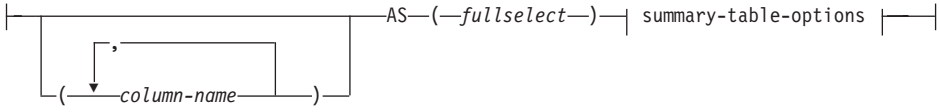


under-clause:

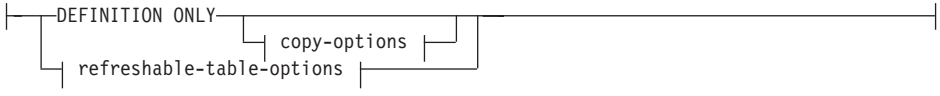
typed-element-list:



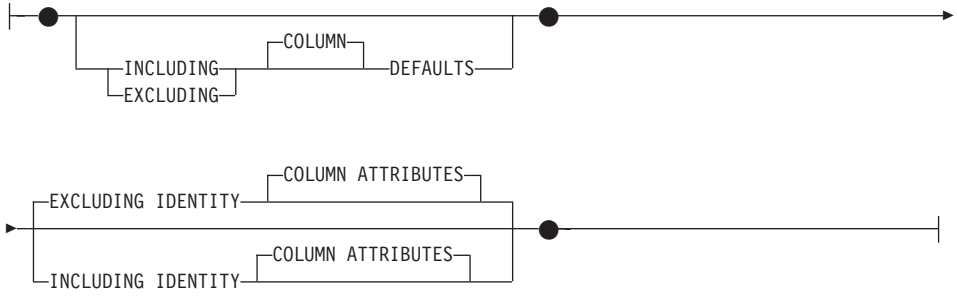
summary-table-definition:



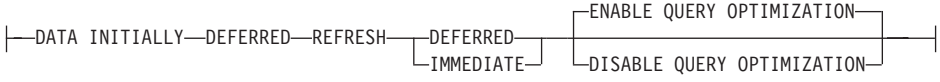
summary-table-options:



copy-options:

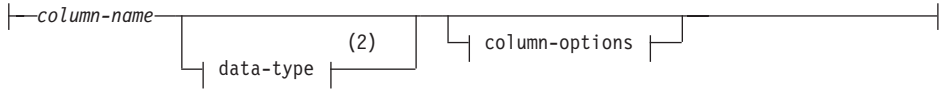


refreshable-table-options:

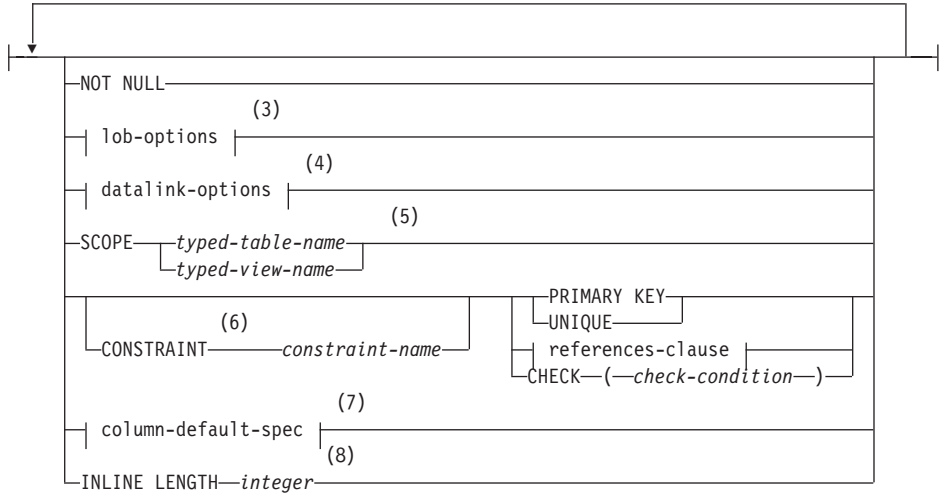


tablespace-options:

column-definition:



column-options:



Notes:

1 Specifying which table space will contain a table's index can only be done when the table is created.

2 If the first column-option chosen is a column-default-spec with a generation-expression, then the data-type can be omitted. It will be determined from the resulting data type of the generation-expression.

3 The lob-options clause only applies to large object types (BLOB, CLOB and DBCLOB) and distinct types based on large object types.

4 The datalink-options clause only applies to the DATALINK type and distinct types based on the DATALINK type. The LINKTYPE URL clause is required for these types.

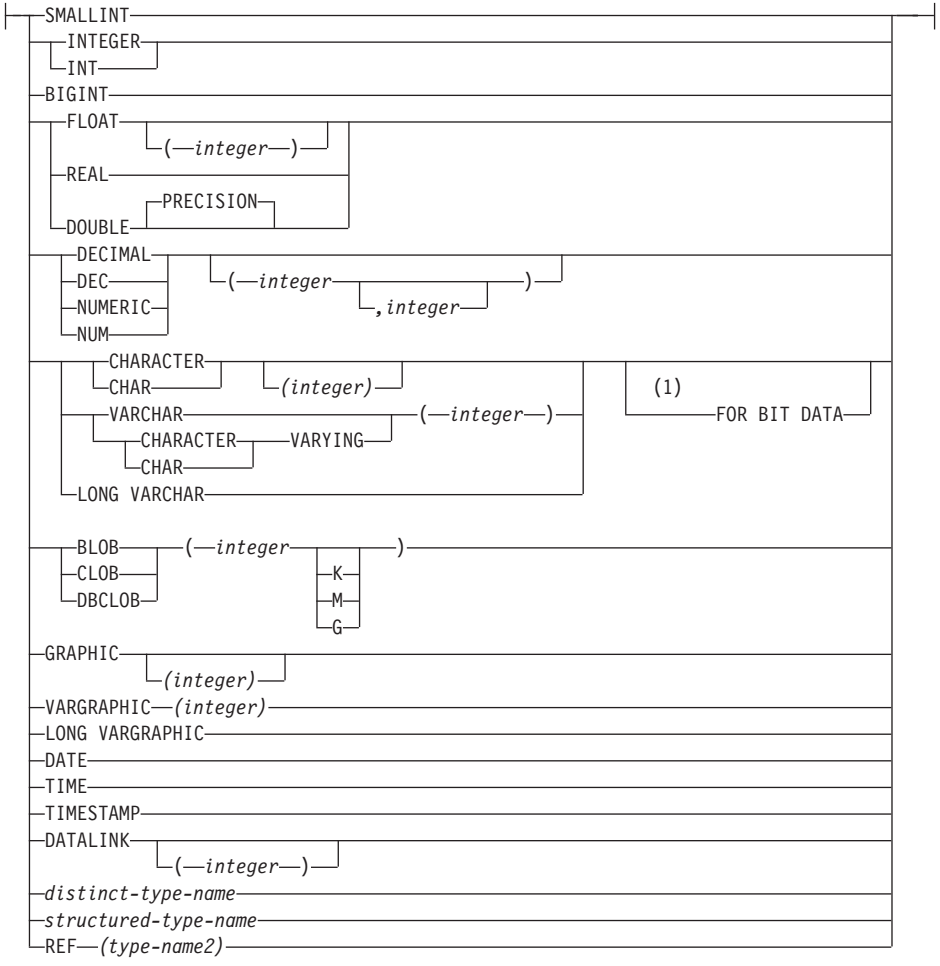
5 The SCOPE clause only applies to the REF type.

6 For compatibility with Version 1, the CONSTRAINT keyword may be omitted in a *column-definition* defining a references-clause.

7 IDENTITY column attributes are not supported in an Extended Enterprise Edition (EEE) database with more than one partition.

8 INLINE LENGTH only applies to columns defined as structured types.

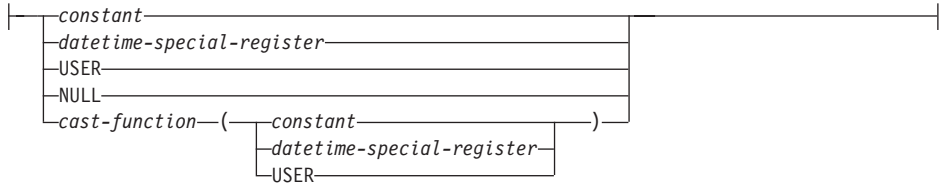
data-type:



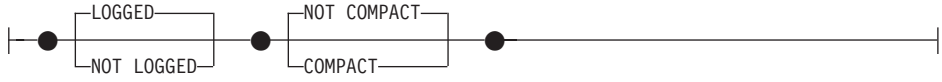
Notes:

1 The FOR BIT DATA clause may be specified in random order with the other column constraints that follow.

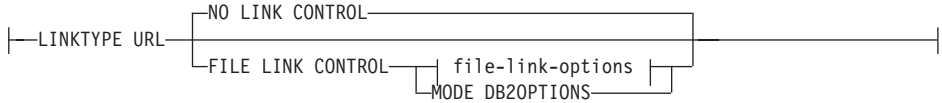
default-values:



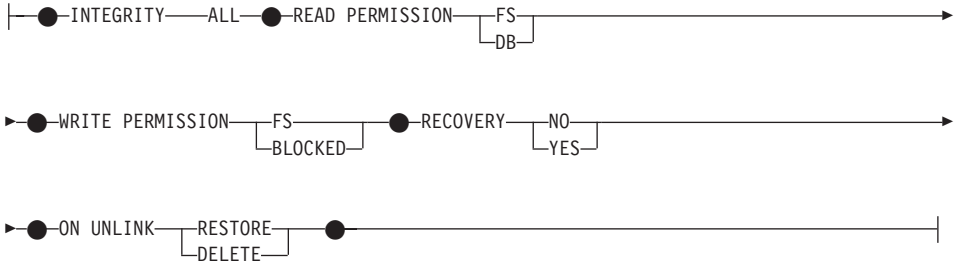
lob-options:



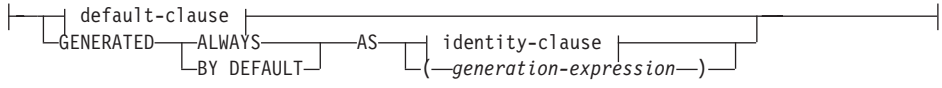
datalink-options:



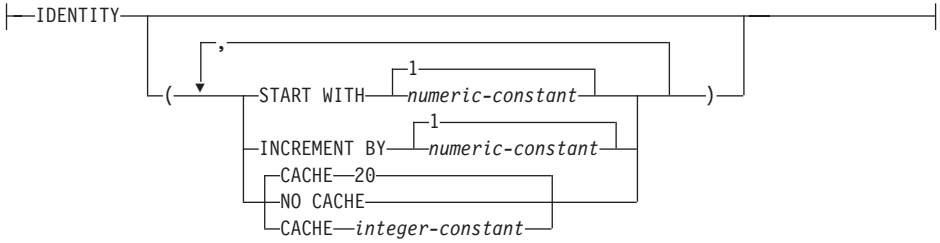
file-link-options:



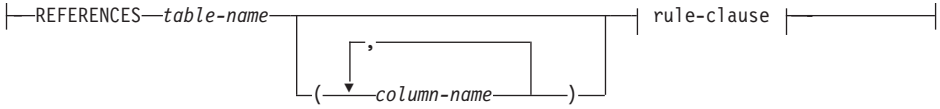
column-default-spec:



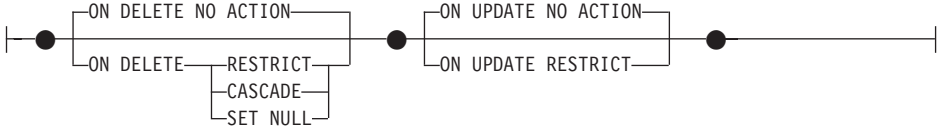
identity-clause:



references-clause:



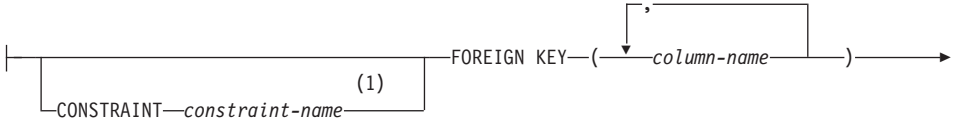
rule-clause:

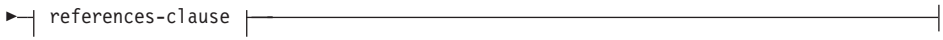


default-clause:

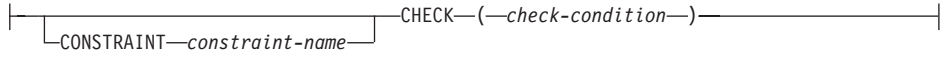
unique-constraint:

referential-constraint:



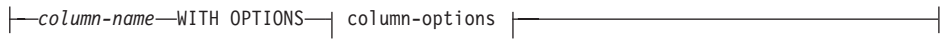


check-constraint:



OID-column-definition:

with-options:



Notes:

1 For compatibility with Version 1, constraint-name may be specified following FOREIGN KEY (without the CONSTRAINT keyword).

Description

SUMMARY
Indicates that a summary table is being defined. The keyword is optional, but when specified, the statement must include a *summary-table-definition* (SQLSTATE 42601).

table-name
Names the table. The name, including the implicit or explicit qualifier, must not identify a table, view, or alias described in the catalog. The schema name must not be SYSIBM, SYSCAT, SYSFUN, or SYSSTAT (SQLSTATE 42939).

OF *type-name1*
Specifies that the columns of the table are based on the attributes of the structured type identified by *type-name1*. If *type-name1* is specified without a schema name, the type name is resolved by searching the schemas on the SQL path (defined by the FUNCPATH preprocessing option for static SQL and by the CURRENT PATH register for dynamic SQL). The type name must be the name of an existing user-defined type (SQLSTATE 42704) and it must be an instantiable structured type (SQLSTATE 428DP) with at least one attribute (SQLSTATE 42997).

If UNDER is not specified, an object identifier column must be specified (refer to the *OID-column-definition*). This object identifier column is the first column of the table. The object ID column is followed by columns based on the attributes of *type-name1*.

HIERARCHY *hierarchy-name*
Names the hierarchy table associated with the table hierarchy. It is created at the same time as the root table of the hierarchy. The data for all subtables in the typed table hierarchy is stored in the hierarchy table. A hierarchy table cannot be directly referenced in SQL statements. A *hierarchy-name* is a *table-name*. The *hierarchy-name*, including the implicit or explicit schema name, must not identify a table, nickname, view, or alias described in the catalog. If the schema name is specified, it must be the same as the schema name of the table being created (SQLSTATE 428DQ). If this clause is omitted when defining the root table, a name is generated by the system consisting of the name of the table being created followed by a unique suffix such that the identifier is unique within the identifiers of the existing tables, views, aliases, and nicknames.

UNDER *supertable-name*
Indicates that the table is a subtable of *supertable-name*. The supertable must be an existing table (SQLSTATE 42704) and the table must be defined using a structured type that is the immediate supertype of *type-name1* (SQLSTATE 428DB). The schema name of *table-name* and *supertable-name* must be the same (SQLSTATE 428DQ). The table identified by *supertable-name* must not have any existing subtable already defined using *type-name1* (SQLSTATE 42742).

The columns of the table include the object identifier column of the supertable with its type modified to be REF(*type-name1*), followed by columns based on the attributes of *type-name1* (remember that the type includes the attributes of its supertype). The attribute names cannot be the same as the OID column name (SQLSTATE 42711).

Other table options including table space, data capture, not logged initially and partitioning key options cannot be specified. These options are inherited from the supertable (SQLSTATE 42613).

INHERIT SELECT PRIVILEGES
Any user or group holding a SELECT privilege on the supertable will be granted an equivalent privilege on the newly created subtable. The subtable definer is considered to be the grantor of this privilege.

element-list
Defines the elements of a table. This includes the definition of columns and constraints on the table.

typed-element-list
Defines the additional elements of a typed table. This includes the

additional options for the columns, the addition of an object identifier column (root table only), and constraints on the table.

summary-table-definition
If the table definition is based on the result of a query, then the table is a summary table based on the query.

column-name
Names the columns in the table. If a list of column names is specified, it must consist of as many names as there are columns in the result table of the fullselect. Each *column-name* must be unique and unqualified. If a list of column names is not specified, the columns of the table inherit the names of the columns of the result table of the fullselect.

A list of column names must be specified if the result table of the fullselect has duplicate column names of an unnamed column (SQLSTATE 42908). An unnamed column is a column derived from a constant, function, expression, or set operation that is not named using the AS clause of the select list.

AS
Introduces the query that is used for the definition of the table and to determine the data included in the table.

fullselect
Defines the query in which the table is based. The resulting column definitions are the same as those for a view defined with the same query.

Every select list element must have a name (use the AS clause for expressions - see "select-clause" on page 395 for details) . The *summary-table-options* specified define attributes of the summary table. The option chosen also defines the contents of the fullselect as follows.

When DEFINITION ONLY is specified, any valid fullselect that does not reference a typed table or typed view can be specified.

When REFRESH DEFERRED or REFRESH IMMEDIATE is specified, the fullselect cannot include (SQLSTATE 428EC):

- references to a nickname, summary table, declared temporary table, or typed table in any FROM clause
- references to a view where the fullselect of the view violates any of the listed restrictions on the fullselect of the summary table
- expressions that are a reference type or DATALINK type (or distinct type based on these types)
- functions that have external action
- functions written in SQL

- functions that depend on physical characteristics (for example NODENUMBER, PARTITION)
- table or view references to system objects (explain tables also should not be specified)
- expressions that are a structured type or LOB type (or a distinct type based on a LOB type)

When REFRESH IMMEDIATE is specified:

- the fullselect must be a subselect
- the subselect cannot include:
  - functions that are not deterministic
  - scalar fullselects
  - predicates with fullselects
  - special registers
- a GROUP BY clause must be included in the subselect unless the summary table is REPLICATED.
- The supported column functions are SUM, COUNT, COUNT_BIG and GROUPING (without DISTINCT). The select list must contain a COUNT(*) or COUNT_BIG(*) column. If the summary table select list contains SUM(X) where X is a nullable argument, then the summary table must also have COUNT(X) in its select list. These column functions cannot be part of any expressions.
- if the FROM clause references more than one table or view, it can only define an inner join without using the explicit INNER JOIN syntax
- all GROUP BY items must be included in the select list
- GROUPING SETS, CUBE and ROLLUP are supported. The GROUP BY items and associated GROUPING column functions in the select list must form a unique key of the result set. Thus, the following restrictions must be satisfied:
  - no grouping sets may be repeated. For example, `ROLLUP(X,Y), X` is not allowed because it is equivalent to `GROUPING SETS((X,Y),(X),(X))`
  - if X is a nullable GROUP BY item that appears within GROUPING SETS, CUBE, or ROLLUP, then GROUPING(X) must appear in the select list
  - grouping on constants is not allowed
- a HAVING clause is not allowed
- if in a multiple partition nodegroup, then a partitioning key must be a subset of the group by items, or the summary table must be replicated.

summary-table-options
Define the attributes of the summary table.

DEFINITION ONLY
The query is used only to define the table. The table is not populated using the results of query and the REFRESH TABLE statement cannot be used. When the CREATE TABLE statement is completed, the table is no longer considered a summary table.

The columns of the table are defined based on the definitions of the columns that result from the fullselect. If the fullselect references a single table in the FROM clause, select list items that are columns of that table are defined using the column name, data type, and nullability characteristic of the referenced table.

refreshable-table-options
Define the refreshable options of the summary table attributes.

DATA INITIALLY DEFERRED
Data is not inserted into the table as part of the CREATE TABLE statement. A REFRESH TABLE statement specifying the *table-name* is used to insert data into the table.

REFRESH
Indicates how the data in the table is maintained.

DEFERRED
The data in the table can be refreshed at any time using the REFRESH TABLE statement. The data in the table only reflects the result of the query as a snapshot at the time the REFRESH TABLE statement is processed. Summary tables defined with this attribute do not allow INSERT, UPDATE or DELETE statements (SQLSTATE 42807).

IMMEDIATE
The changes made to the underlying tables as part of a DELETE, INSERT, or UPDATE are cascaded to the summary table. In this case, the content of the table, at any point-in-time, is the same as if the specified *subselect* is processed. Summary tables defined with this attribute do not allow INSERT, UPDATE, or DELETE statements (SQLSTATE 42807).

ENABLE QUERY OPTIMIZATION
The summary table can be used for query optimization under appropriate circumstances.

DISABLE QUERY OPTIMIZATION
The summary table will not be used for query optimization. The table can still be queried directly.

LIKE *table-name1* **or** *view-name* **or** *nickname*
Specifies that the columns of the table have exactly the same name and description as the columns of the identified table (*table-name1*), view (*view-name*) or nickname (*nickname*). The name specified after LIKE must identify a table, view or nickname that exists in the catalog, or a declared temporary table. A typed table or typed view cannot be specified (SQLSTATE 428EC).

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

- If a table is identified, then the implicit definition includes the column name, data type and nullability characteristic of each of the columns of *table-name1*. If EXCLUDING COLUMN DEFAULTS is not specified, then the column default is also included.
- If a view is identified, then the implicit definition includes the column name, data type, and nullability characteristic of each of the result columns of the fullselect defined in *view-name*.
- If a nickname is identified, then the implicit definition includes the column name, data type, and nullability characteristic of each column of *nickname*.

Column default and identity column attributes may be included or excluded, based on the copy-attributes clauses. The implicit definition does not include any other attributes of the identified table, view or nickname. Thus the new table does not have any unique constraints, foreign key constraints, triggers, or indexes. The table is created in the table space implicitly or explicitly specified by the IN clause, and the table has any other optional clause only if the optional clause is specified.

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

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

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

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

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

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

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

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

column-definition
Defines the attributes of a column.

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

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 8101. Refer to "Row Size" on page 756 for more details.
- an 16K page size with maximum of 1 012 columns where the byte counts of the columns must not be greater than 16 293.
- an 32K page size with maximum of 1 012 columns where the byte counts of the columns must not be greater than 32 677.

data-type
Is one of the types in the following list. Use:

SMALLINT
For a small integer.

INTEGER or **INT**
For a large integer.

BIGINT
For a big integer.

FLOAT(*integer***)**
For a single or double precision floating-point number, depending on the value of the *integer*. The value of the integer must be in the range 1 through 53. The values 1 through 24 indicate single precision and the values 25 through 53 indicate double precision.

You can also specify:

REAL	For single precision floating-point.
DOUBLE	For double precision floating-point.
DOUBLE PRECISION	For double precision floating-point.
FLOAT	For double precision floating-point.

DECIMAL*(precision-integer, scale-integer)* or **DEC***(precision-integer, scale-integer)*
For a decimal number. The first integer is the precision of the number; that is, the total number of digits; it may range from 1 to 31. The second integer is the scale of the number; that is, the number of digits to the right of the decimal point; it may range from 0 to the precision of the number.

If precision and scale are not specified, the default values of 5,0 are used. The words **NUMERIC** and **NUM** can be used as synonyms for **DECIMAL** and **DEC**.

CHARACTER*(integer)* or **CHAR***(integer)* or **CHARACTER** or **CHAR**
For a fixed-length character string of length *integer*, which may range from 1 to 254. If the length specification is omitted, a length of 1 character is assumed.

VARCHAR*(integer)*, or **CHARACTER VARYING***(integer)*, or **CHAR VARYING***(integer)*
For a varying-length character string of maximum length *integer*, which may range from 1 to 32 672.

LONG VARCHAR
For a varying-length character string with a maximum length of 32700.

FOR BIT DATA
Specifies that the contents of the column are to be treated as bit

(binary) data. During data exchange with other systems, code page conversions are not performed. Comparisons are done in binary, irrespective of the database collating sequence.

BLOB*(integer [K | M | G])*
For a binary large object string of the specified maximum length in bytes.

The length may be in the range of 1 byte to 2 147 483 647 bytes.

If *integer* by itself is specified, that is the maximum length.

If *integer K* (in either upper or lower case) is specified, the maximum length is 1 024 times *integer*. The maximum value for *integer* is 2 097 152.

If *integer M* is specified, the maximum length is 1 048 576 times *integer*. The maximum value for *integer* is 2 048.

If *integer G* is specified, the maximum length is 1 073 741 824 times *integer*. The maximum value for *integer* is 2.

To create BLOB strings greater than 1 gigabyte, you must specify the NOT LOGGED option.

Any number of spaces is allowed between the integer and K, M, or G. Also, no space is required. For example, all the following are valid.

```
BLOB(50K)    BLOB(50 K)    BLOB (50   K)
```

CLOB*(integer [K | M | G])*[74]
For a character large object string of the specified maximum length in bytes.

The meaning of the *integer K | M | G* is the same as for BLOB.

To create CLOB strings greater than 1 gigabyte, you must specify the NOT LOGGED option.

DBCLOB*(integer [K | M | G])*
For a double-byte character large object string of the specified maximum length in double-byte characters.

The meaning of the *integer K | M | G* is similar to that for BLOB. The differences are that the number specified is the number of double-byte characters and that the maximum size is 1 073 741 823 double-byte characters.

74. Observe that it is not possible to specify the FOR BIT DATA clause for CLOB columns. However, a CHAR FOR BIT DATA string can be assigned to a CLOB column and a CHAR FOR BIT DATA string can be concatenated with a CLOB string.

To create DBCLOB strings greater than 1 gigabyte, you must specify the NOT LOGGED option.

GRAPHIC*(integer)*
For a fixed-length graphic string of length *integer* which may range from 1 to 127. If the length specification is omitted, a length of 1 is assumed.

VARGRAPHIC*(integer)*
For a varying-length graphic string of maximum length *integer*, which may range from 1 to 16 336.

LONG VARGRAPHIC
For a varying-length graphic string with a maximum length of 16 350.

DATE
For a date.

TIME
For a time.

TIMESTAMP
For a timestamp.

DATALINK or **DATALINK***(integer)*
For a link to data stored outside the database.

The column in the table consists of ″anchor values″ that contain the reference information that is required to establish and maintain the link to the external data as well as an optional comment.

The length of a DATALINK column is 200 bytes. If *integer* is specified, it must be 200. If the length specification is omitted, a length of 200 bytes is assumed.

A DATALINK value is an encapsulated value with a set of built-in scalar functions. There is a function called DLVALUE to create a DATALINK value. The following functions can be used to extract attributes from a DATALINK value.

- DLCOMMENT
- DLLINKTYPE
- DLURLCOMPLETE
- DLURLPATH
- DLURLPATHONLY
- DLURLSCHEME
- DLURLSERVER

A DATALINK column has the following restrictions:

- The column cannot be part of any index. Therefore, it cannot be included as a column of a primary key or unique constraint (SQLSTATE 42962).
- The column cannot be a foreign key of a referential constraint (SQLSTATE 42830).
- A default value (WITH DEFAULT) cannot be specified for the column. If the column is nullable, the default for the column is NULL (SQLSTATE 42894).

distinct-type-name
For a user-defined type that is a distinct type. If a distinct type name is specified without a schema name, the distinct type name is resolved by searching the schemas on the SQL path (defined by the FUNCPATH preprocessing option for static SQL and by the CURRENT PATH register for dynamic SQL).

If a column is defined using a distinct type, then the data type of the column is the distinct type. The length and the scale of the column are respectively the length and the scale of the source type of the distinct type.

If a column defined using a distinct type is a foreign key of a referential constraint, then the data type of the corresponding column of the primary key must have the same distinct type.

structured-type-name
For a user-defined type that is a structured type. If a structured type name is specified without a schema name, the structured type name is resolved by searching the schemas on the SQL path (defined by the FUNCPATH preprocessing option for static SQL, and by the CURRENT PATH register for dynamic SQL).

If a column is defined using a structured type, then the static data type of the column is the structured type. The column may include values with a dynamic type that is a subtype of *structured-type-name*.

A column defined using a structured type cannot be used in a primary key, unique constraint, foreign key, index key or partitioning key (SQLSTATE 42962).

If a column is defined using a structured type, and contains a reference-type attribute at any level of nesting, that reference-type attribute is unscoped. To use such an attribute in a dereference operation, it is necessary to specify a SCOPE explicitly, using a CAST specification.

If a column is defined using a structured type with an attribute of type DATALINK, or a distinct type sourced on DATALINK, this

column can only be null. An attempt to use the constructor function for this type will return an error (SQLSTATE 428ED) and so no instance of this type can be inserted into the column.

REF (*type-name2***)**
For a reference to a typed table. If *type-name2* is specified without a schema name, the type name is resolved by searching the schemas on the SQL path (defined by the FUNCPATH preprocessing option for static SQL and by the CURRENT PATH register for dynamic SQL). The underlying data type of the column is based on the representation data type specified in the REF USING clause of the CREATE TYPE statement for *type-name2* or the root type of the data type hierarchy that includes *type-name2*.

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

NOT NULL
Prevents the column from containing null values.

If NOT NULL is not specified, the column can contain null values, and its default value is either the null value or the value provided by the WITH DEFAULT clause.

lob-options
Specifies options for LOB data types.

LOGGED
Specifies that changes made to the column are to be written to the log. The data in such columns is then recoverable with database utilities (such as RESTORE DATABASE). LOGGED is the default.

LOBs greater than 1 gigabyte cannot be logged (SQLSTATE 42993) and LOBs greater than 10 megabytes should probably not be logged.

NOT LOGGED
Specifies that changes made to the column are not to be logged.

NOT LOGGED has no effect on a commit or rollback operation; that is, the database's consistency is maintained even if a transaction is rolled back, regardless of whether or not the LOB value is logged. The implication of not logging is that during a roll forward operation, after a backup or load operation, the LOB data will be replaced by zeros for those LOB values that would have had log records replayed during the roll forward. During crash recovery, all committed changes

and changes rolled back will reflect the expected results. See the *Administration Guide* for the implications of not logging LOB columns.

COMPACT
Specifies that the values in the LOB column should take up minimal disk space (free any extra disk pages in the last group used by the LOB value), rather than leave any leftover space at the end of the LOB storage area that might facilitate subsequent append operations. Note that storing data in this way may cause a performance penalty in any append (length-increasing) operations on the column.

NOT COMPACT
Specifies some space for insertions to assist in future changes to the LOB values in the column. This is the default.

datalink-options
Specifies the options associated with a DATALINK data type.

LINKTYPE URL
This defines the type of link as a Uniform Resource Locator (URL).

NO LINK CONTROL
Specifies that there will not be any check made to determine that the file exists. Only the syntax of the URL will be checked. There is no database manager control over the file.

FILE LINK CONTROL
Specifies that a check should be made for the existence of the file. Additional options may be used to give the database manager further control over the file.

file-link-options
Additional options to define the level of database manager control of the file link.

INTEGRITY
Specifies the level of integrity of the link between a DATALINK value and the actual file.

ALL
Any file specified as a DATALINK value is under the control of the database manager and may NOT be deleted or renamed using standard file system programming interfaces.

READ PERMISSION
Specifies how permission to read the file specified in a DATALINK value is determined.

FS The read access permission is determined by the file system permissions. Such files can be accessed without retrieving the file name from the column.

DB
The read access permission is determined by the database. Access to the file will only be allowed by passing a valid file access token, returned on retrieval of the DATALINK value from the table, in the open operation.

WRITE PERMISSION
Specifies how permission to write to the file specified in a DATALINK value is determined.

FS The write access permission is determined by the file system permissions. Such files can be accessed without retrieving the file name from the column.

BLOCKED
Write access is blocked. The file cannot be directly updated through any interface. An alternative mechanism must be used to cause updates to the information. For example, the file is copied, the copy updated, and then the DATALINK value updated to point to the new copy of the file.

RECOVERY
Specifies whether or not DB2 will support point in time recovery of files referenced by values in this column.

YES
DB2 will support point in time recovery of files referenced by values in this column. This value can only be specified when INTEGRITY ALL and WRITE PERMISSION BLOCKED are also specified.

NO
Specifies that point in time recovery will not be supported.

ON UNLINK
Specifies the action taken on a file when a DATALINK value is changed or deleted (unlinked). Note that this is not applicable when WRITE PERMISSION FS is used.

RESTORE
Specifies that when a file is unlinked, the DataLink File Manager will attempt to return the file to the owner with the permissions that existed at the time

the file was linked. In the case where the user is no longer registered with the file server, the result is product-specific.[75] This can only be specified when INTEGRITY ALL and WRITE PERMISSION BLOCKED are also specified.

DELETE
Specifies that the file will be deleted when it is unlinked. This can only be specified when READ PERMISSION DB and WRITE PERMISSION BLOCKED are also specified.

MODE DB2OPTIONS
This mode defines a set of default file link options. The defaults defined by DB2OPTIONS are:

- INTEGRITY ALL
- READ PERMISSION FS
- WRITE PERMISSION FS
- RECOVERY NO

ON UNLINK is not applicable since WRITE PERMISSION FS is used.

SCOPE
Identifies the scope of the reference type column.

A scope must be specified for any column that is intended to be used as the left operand of a dereference operator or as the argument of the DEREF function. Specifying the scope for a reference type column may be deferred to a subsequent ALTER TABLE statement to allow the target table to be defined, usually in the case of mutually referencing tables.

typed-table-name
The name of a typed table. The table must already exist or be the same as the name of the table being created (SQLSTATE 42704). The data type of *column-name* must be REF(*S*), where *S* is the type of *typed-table-name* (SQLSTATE 428DM). No checking is done of values assigned to *column-name* to ensure that the values actually reference existing rows in *typed-table-name*.

typed-view-name
The name of a typed view. The view must already exist or be the same as the name of the view being created (SQLSTATE 42704). The data type of *column-name* must be REF(*S*), where *S*

75. With DB2 Universal Database, the file is assigned to a special predefined "dfmunknown" user id.

is the type of *typed-view-name* (SQLSTATE 428DM). No checking is done of values assigned to *column-name* to ensure that the values actually reference existing rows in *typed-view-name*.

CONSTRAINT *constraint-name*
Names the constraint. A *constraint-name* must not identify a constraint that was already specified within the same CREATE TABLE statement. (SQLSTATE 42710).

If this clause is omitted, an 18-character identifier unique within the identifiers of the existing constraints defined on the table, is generated[76] by the system.

When used with a PRIMARY KEY or UNIQUE constraint, the *constraint-name* may be used as the name of an index that is created to support the constraint.

PRIMARY KEY
This provides a shorthand method of defining a primary key composed of a single column. Thus, if PRIMARY KEY is specified in the definition of column C, the effect is the same as if the PRIMARY KEY(C) clause is specified as a separate clause.

A primary key cannot be specified if the table is a subtable (SQLSTATE 429B3) since the primary key is inherited from the supertable.

See PRIMARY KEY within the description of the *unique-constraint* below.

UNIQUE
This provides a shorthand method of defining a unique key composed of a single column. Thus, if UNIQUE is specified in the definition of column C, the effect is the same as if the UNIQUE(C) clause is specified as a separate clause.

A unique constraint cannot be specified if the table is a subtable (SQLSTATE 429B3) since unique constraints are inherited from the supertable.

See UNIQUE within the description of the *unique-constraint* below.

references-clause
This provides a shorthand method of defining a foreign key composed of a single column. Thus, if a references-clause is specified in the definition of column C, the effect is the same as if

76. The identifier is formed of ″SQL″ followed by a sequence of 15 numeric characters generated by a timestamp-based function.

that references-clause were specified as part of a FOREIGN KEY clause in which C is the only identified column.

See *references-clause* under *referential-constraint* below.

CHECK (*check-condition***)**
This provides a shorthand method of defining a check constraint that applies to a single column. See CHECK (*check-condition*) below.

INLINE LENGTH *integer*
This option is only valid for a column defined using a structured type (SQLSTATE 42842) and indicates the maximum byte size of an instance of a structured type to store inline with the rest of the values in the row. Instances of structured types that cannot be stored inline are stored separately from the base table row, similar to the way that LOB values are handled. This takes place automatically.

The default INLINE LENGTH for a structured-type column is the inline length of its type (specified explicitly or by default in the CREATE TYPE statement). If INLINE LENGTH of the structured type is less than 292, the value 292 is used for the INLINE LENGTH of the column.

Note: The inline lengths of subtypes are not counted in the default inline length, meaning that instances of subtypes may not fit inline unless an explicit INLINE LENGTH is specified at CREATE TABLE time to account for existing and future subtypes.

The explicit INLINE LENGTH value must be at least 292 and cannot exceed 32672 (SQLSTATE 54010).

column-default-spec

default-clause
Specifies a default value for the column.

WITH
An optional keyword.

DEFAULT
Provides a default value in the event a value is not supplied on INSERT or is specified as DEFAULT on INSERT or UPDATE. If a default value is not specified following the DEFAULT keyword, the default value depends on the data type of the column as shown in Table 19 on page 487.

If a column is defined as a DATALINK, then a default value cannot be specified (SQLSTATE 42613). The only possible default is NULL.

If the column is based on a column of a typed table, a specific default value must be specified when defining a default. A default value cannot be specified for the object identifier column of a typed table (SQLSTATE 42997).

If a column is defined using a distinct type, then the default value of the column is the default value of the source data type cast to the distinct type.

If a column is defined using a structured type, the *default-clause* cannot be specified (SQLSTATE 42842).

Omission of DEFAULT from a *column-definition* results in the use of the null value as the default for the column. If such a column is defined NOT NULL, then the column does not have a valid default.

default-values
Specific types of default values that can be specified are as follows.

constant
Specifies the constant as the default value for the column. The specified constant must:

- represent a value that could be assigned to the column in accordance with the rules of assignment as described in Chapter 3
- not be a floating-point constant unless the column is defined with a floating-point data type
- not have non-zero digits beyond the scale of the column data type if the constant is a decimal constant (for example, 1.234 cannot be the default for a DECIMAL(5,2) column)
- be expressed with no more than 254 characters including the quote characters, any introducer character such as the X for a hexadecimal constant, and characters from the fully qualified function name and parentheses when the constant is the argument of a *cast-function*.

datetime-special-register
Specifies the value of the datetime special register (CURRENT DATE, CURRENT TIME, or CURRENT TIMESTAMP) at the time of INSERT or UPDATE as

the default for the column. The data type of the column must be the data type that corresponds to the special register specified (for example, data type must be DATE when CURRENT DATE is specified).

USER
Specifies the value of the USER special register at the time of INSERT or UPDATE as the default for the column. If USER is specified, the data type of the column must be a character string with a length not less than the length attribute of USER.

NULL
Specifies NULL as the default for the column. If NOT NULL was specified, DEFAULT NULL may be specified within the same column definition but will result in an error on any attempt to set the column to the default value.

cast-function
This form of a default value can only be used with columns defined as a distinct type, BLOB or datetime (DATE, TIME or TIMESTAMP) data type. For distinct type, with the exception of distinct types based on BLOB or datetime types, the name of the function must match the name of the distinct type for the column. If qualified with a schema name, it must be the same as the schema name for the distinct type. If not qualified, the schema name from function resolution must be the same as the schema name for the distinct type. For a distinct type based on a datetime type, where the default value is a constant, a function must be used and the name of the function must match the name of the source type of the distinct type with an implicit or explicit schema name of SYSIBM. For other datetime columns, the corresponding datetime function may also be used. For a BLOB or a distinct type based on BLOB, a function must be used and the name of the function must be BLOB with an implicit or explicit schema name of SYSIBM. For an example of using the *cast-function*, see 501.

constant
Specifies a constant as the argument. The constant must conform to the rules of a constant for the source type of the distinct type or for the data

type if not a distinct type. If the *cast-function* is BLOB, the constant must be a string constant.

datetime-special-register
Specifies CURRENT DATE, CURRENT TIME, or CURRENT TIMESTAMP. The source type of the distinct type of the column must be the data type that corresponds to the specified special register.

USER
Specifies the USER special register. The data type of the source type of the distinct type of the column must be a string data type with a length of at least 8 bytes. If the *cast-function* is BLOB, the length attribute must be at least 8 bytes.

If the value specified is not valid, an error (SQLSTATE 42894) is raised.

GENERATED
Indicates that DB2 generates values for the column. You must specify GENERATED if the column is to be considered a generated column or an IDENTITY column.

ALWAYS
Indicates that DB2 will always generate a value for the column when a row is inserted into the table or whenever the result value of the *generation-expression* may change. The result of the expression is stored in the table. GENERATED ALWAYS is the recommended value unless you are using data propagation, or doing unload and reload operations. GENERATED ALWAYS is the required value for generated columns.

BY DEFAULT
Indicates that DB2 will generate a value for the column when a row is inserted into the table, unless a value is specified. BY DEFAULT is the recommended value when using data propagation or doing unload/reload.

Although not explicitly required, a unique, single-column index should be defined on the generated column to ensure uniqueness of the values.

AS IDENTITY
Specifies that the column is to be the identity column for this

table.[77] A table can only have a single IDENTITY column (SQLSTATE 428C1). The IDENTITY keyword can only be specified if the *data-type* associated with the column is an exact numeric type[78] with a scale of zero, or a user-defined distinct type for which the source type is an exact numeric type with a scale of zero (SQLSTATE 42815).

An identity column is implicitly NOT NULL.

START WITH *numeric-constant*
Specifies the first value for the identity column. This value can be any positive or negative value that could be assigned to this column (SQLSTATE 42820) as long as there are no non-zero digits to the right of the decimal point (SQLSTATE 42894). The default is 1.

INCREMENT BY *numeric-constant*
Specifies the interval between consecutive values of the identity column. This value can be any positive or negative value that could be assigned to this column (SQLSTATE 42820). This value cannot be zero and cannot exceed the value of a large integer constant (SQLSTATE 428125), provided that there are no non-zero digits to the right of the decimal point (SQLSTATE 42894).

If this value is negative, then the sequence of values for this identity column descends. If this value is positive, then the sequence of values for this identity column ascends. The default is 1.

CACHE or NO CACHE
Specifies whether to keep some pre-allocated values in memory for faster access. If a new value is needed for the identity column, and there are none available in the cache, then the end of the new cache block must be logged. However, when a new value is needed for the identity column, and there is an unused value in the cache, then the allocation of that identity value is quicker, since no logging is necessary. This is a performance and tuning option.

77. Identity columns are not be supported in a database with multiple partitions (SQLSTATE 42997). An identity column cannot be created if more than one partition for the database exists. A database that includes any identity columns cannot be started with more than one partition.

78. SMALLINT, INTEGER, BIGINT, or DECIMAL with a scale of zero, or a distinct type based on one of these types are considered exact numeric types. By contrast, single and double precision floating points are considered approximate numeric data types. Reference types, even if represented by an exact numeric type cannot be defined as identity columns.

CACHE *integer-constant*
Specifies how many values of the identity sequence that DB2 pre-allocates and keeps in memory. Pre-allocating and storing values in the cache reduces logging when values are generated for the identity column.

If a new value is needed for the identity column and there are none available in the cache, then the allocation of the value involves waiting for the log. However, when a new value is needed for the identity column and there is an unused value in the cache, the allocation of that identity value can be made quicker by not performing the logging.

In the event of a database deactivation, either normally[79] or due to a system failure, all cached sequence values that have not been used in committed statements are lost. The value specified for the CACHE option is the maximum number of values for the identity column that could be lost in case of database deactivation.

The minimum value is 2 and the maximum value is 32767 (SQLSTATE 42815). The default is CACHE 20.

NO CACHE
Specifies that values for the identity column are not to be pre-allocated.

When this option is specified, the values of the identity column are not stored in the cache. In this case, every request for a new identity value results in logging.

AS (*generation-expression***)**
Specifies that the definition of the column is based on an expression.[80] The *generation-expression* cannot contain any of the following (SQLSTATE 42621):

- subqueries
- column functions
- dereference operations or DEREF functions

79. If a database is not explicitly activated (using the ACTIVATE command or API), when the last application is disconnected from the database, an implicit deactivation occurs.

80. If the expression for a GENERATED ALWAYS column includes a user-defined external function, changing the executable for the function (such that the results change for given arguments) can result in inconsistent data. This can be avoided by using the SET INTEGRITY statement to force the generation of new values.

- user-defined or built-in functions that are non-deterministic
- user-defined functions using the EXTERNAL ACTION option
- user-defined functions using the SCRATCHPAD option
- user-defined functions using the READS SQL DATA option
- host variables or parameter markers
- special registers
- references to columns defined later in the column list
- references to other generated columns

The data type for the column is based on the result data type of the *generation-expression*. A CAST specification can be used to force a particular data type and to provide a scope (for a reference type only). If *data-type* is specified, values are assigned to the column under the assignment rules described in "Chapter 3. Language Elements" on page 63. A generated column is implicitly considered nullable, unless the NOT NULL column option is used. The data type of a generated column must be one for which equality is defined. This excludes columns of LONG VARCHAR, LONG VARGRAPHIC, LOB data types, DATALINKs, structured types, and distinct types based on any of these types (SQLSTATE 42962).

OID-column-definition
Defines the object identifier column for the typed table.

REF IS *OID-column-name* **USER GENERATED**
Specifies that an object identifier (OID) column is defined in the table as the first column. An OID is required for the root table of a table hierarchy (SQLSTATE 428DX). The table must be a typed table (the OF clause must be present) that is not a subtable (SQLSTATE 42613). The name for the column is defined as *OID-column-name* and cannot be the same as the name of any attribute of the structured type *type-name1* (SQLSTATE 42711). The column is defined with type REF(*type-name1*), NOT NULL and a system required unique index (with a default index name) is generated. This column is referred to as the *object identifier column* or *OID column*. The keywords USER GENERATED indicate that the initial value for the OID column must be provided by the user when inserting a row. Once a row is inserted, the OID column cannot be updated (SQLSTATE 42808).

with-options
Defines additional options that apply to columns of a typed table.

column-name
Specifies the name of the column for which additional options are specified. The *column-name* must correspond to the name of a column of the table that is not also a column of a supertable (SQLSTATE 428DJ). A column name can only appear in one WITH OPTIONS clause in the statement (SQLSTATE 42613).

If an option is already specified as part of the type definition (in CREATE TYPE), the options specified here override the options in CREATE TYPE.

WITH OPTIONS *column-options*
Defines options for the specified column. See *column-options* described earlier. If the table is a subtable, primary key or unique constraints cannot be specified (SQLSTATE 429B3).

DATA CAPTURE
Indicates whether extra information for inter-database data replication is to be written to the log. This clause cannot be specified when creating a subtable (SQLSTATE 42613).

If the table is a typed table, then this option is not supported (SQLSTATE 428DH or 42HDR).

NONE
Indicates that no extra information will be logged.

CHANGES
Indicates that extra information regarding SQL changes to this table will be written to the log. This option is required if this table will be replicated and the Capture program is used to capture changes for this table from the log.

If the table is defined to allow data on a partition other than the catalog partition (multiple partition nodegroup or nodegroup with a partition other than the catalog partition), then this option is not supported (SQLSTATE 42997).

If the schema name (implicit or explicit) of the table is longer than 18 bytes, then this option is not supported (SQLSTATE 42997).

Further information about using replication can be found in the *Administration Guide* and the *Replication Guide and Reference*.

IN *tablespace-name1*
Identifies the table space in which the table will be created. The table space must exist, and be a REGULAR table space over which the authorization ID of the statement has USE privilege. If no

other table space is specified, then all table parts will be stored in this table space. This clause cannot be specified when creating a subtable (SQLSTATE 42613), since the table space is inherited from the root table of the table hierarchy. If this clause is not specified, a table space for the table is determined as follows:

```
IF table space IBMDEFAULTGROUP over which the user has USE privilege
  exists with sufficient page size
    THEN choose it
 ELSE IF a table space over which the user has USE privilege
  exists with sufficient page size
  (see below when multiple table spaces qualify)
    THEN choose it
 ELSE issue an error (SQLSTATE 42727).
```

If more than one table space is identified by the ELSE IF condition, then choose the table space with the smallest sufficient page size over which the authorization ID of the statement has USE privilege. When more than one table space qualifies, preference is given according to who was granted the USE privilege:

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

If more than one table space still qualifies, the final choice is made by the database manager.

Determination of the table space may change when:

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

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

tablespace-options:

Specifies the table space in which indexes and/or long column values will be stored. See "CREATE TABLESPACE" on page 764 for details on types of table spaces.

INDEX IN *tablespace-name2*

Identifies the table space in which any indexes on the table will be created. This option is allowed only when the primary table space specified in the IN clause is a DMS table space. The specified table space must exist, must be a REGULAR DMS table space over which the authorization ID of the statement has USE privilege, and must be in the same nodegroup as *tablespace-name1* (SQLSTATE 42838).

Note that specifying which table space will contain a table's index can only be done when the table is created. The checking of USE privilege over the table space for the index is only carried out at table creation time. The database manager will not require that the authorization ID of a CREATE INDEX statement have USE privilege on the table space when an index is created later.

LONG IN *tablespace-name3*
Identifies the table space in which the values of any long columns (LONG VARCHAR, LONG VARGRAPHIC, LOB data types, distinct types with any of these as source types, or any columns defined with user-defined structured types with values that cannot be stored inline) will be stored. This option is allowed only when the primary table space specified in the IN clause is a DMS table space. The table space must exist, must be a LONG DMS table space over which the authorization ID of the statement has USE privilege, and must be in the same nodegroup of *tablspace-name1* (SQLSTATE 42838).

Note that specifying which table space will contain a table's long and LOB columns can only be done when the table is created. The checking of USE privilege over the table space for the long and LOB columns is only carried out at table creation time. The database manager will not require that the authorization ID of an ALTER TABLE statement have USE privilege on the table space when a long or LOB column is added later.

PARTITIONING KEY (*column-name***,...)**
Specifies the partitioning key used when data in the table is partitioned. Each *column-name* must identify a column of the table and the same column must not be identified more than once. No column with data type that is a LONG VARCHAR, LONG VARGRAPHIC, BLOB, CLOB, DBCLOB, DATALINK, distinct type based on any of these types, or structured type may be used as part of a partitioning key (SQLSTATE 42962). A partitioning key cannot be specified for a table that is a subtable (SQLSTATE 42613), since the partitioning key is inherited from the root table in the table hierarchy.

If this clause is not specified, and this table resides in a multiple partition nodegroup, then the partitioning key is defined as follows:

- if the table is a typed table, the object identifier column

- if a primary key is specified, the first column of the primary key is the partitioning key
- otherwise, the first column whose data type is not a LOB, LONG VARCHAR, LONG VARGRAPHIC, DATALINK column, distinct type based on one of these types, or structured type column is the partitioning key.

If none of the columns satisfy the requirement of the default partitioning key, the table is created without one. Such tables are allowed only in table spaces defined on single-partition nodegroups.

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

For restrictions related to the partitioning key, see "Rules" on page 752.

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

REPLICATED
Specifies that the data stored in the table is physically replicated on each database partition of the nodegroup of the table space in which the table is defined. This means that a copy of all the data in the table exists on each of these database partitions. This option can only be specified for a summary table (SQLSTATE 42997).

NOT LOGGED INITIALLY
Any changes made to the table by an Insert, Delete, Update, Create Index, Drop Index, or Alter Table operation in the same unit of work in which the table is created are not logged. See "Notes" on page 753 for other considerations when using this option.

All catalog changes and storage related information are logged, as are all operations that are done on the table in subsequent units of work.

A foreign key constraint cannot be defined on a table that references a parent with the NOT LOGGED INITIALLY attribute. This clause cannot be specified when creating a subtable (SQLSTATE 42613).

Note: A rollback to savepoint request cannot be issued in the same unit of work as the creation of a NOT LOGGED INITIALLY table. This will result in an error (SQLSTATE 40506), and the entire unit of work will be rolled back.

unique-constraint

Defines a unique or primary key constraint. If the table has a partitioning key, then any unique or primary key must be a superset of the partitioning key. A unique or primary key constraint cannot be specified for a table that is a subtable (SQLSTATE 429B3). If the table is a root table, the constraint applies to the table and all its subtables.

CONSTRAINT *constraint-name*

Names the primary key or unique constraint. See page 734.

UNIQUE (*column-name***,...)**

Defines a unique key composed of the identified columns. The identified columns must be defined as NOT NULL. Each *column-name* must identify a column of the table and the same column must not be identified more than once.

The number of identified columns must not exceed 16 and the sum of their stored lengths must not exceed 1024 (refer to "Byte Counts" on page 757 for the stored lengths). The length of any individual column must not exceed 255 bytes. This length is for the data only and is not affected by the null byte, should it be present. The maximum data length of a column is 255 bytes, whether the column is nullable or not. No LOB, LONG VARCHAR, LONG VARGRAPHIC, DATALINK, distinct type based on one of these types, or structured type may be used as part of a unique key, even if the length attribute of the column is small enough to fit within the 255 byte limit (SQLSTATE 42962).

The set of columns in the unique key cannot be the same as the set of columns of the primary key or another unique key (SQLSTATE 01543).[81]

A unique constraint cannot be specified if the table is a subtable (SQLSTATE 429B3) since unique constraints are inherited from the supertable.

The description of the table as recorded in the catalog includes the unique key and its unique index. A unique index will automatically be created for the columns in the sequence specified with ascending order for each column. The name of the index will be the same as the *constraint-name* if this does not conflict with an existing index in the schema where the table is created. If the index name conflicts, the

81. If LANGLEVEL is SQL92E or MIA then an error is returned, SQLSTATE 42891.

name will be SQL, followed by a character timestamp (*yymmddhhmmssxxx*), with SYSIBM as the schema name.

PRIMARY KEY (*column-name***,...)**
Defines a primary key composed of the identified columns. The clause must not be specified more than once and the identified columns must be defined as NOT NULL. Each *column-name* must identify a column of the table and the same column must not be identified more than once.

The number of identified columns must not exceed 16 and the sum of their stored lengths must not exceed 1024 (refer to "Byte Counts" on page 757 for the stored lengths). The length of any individual column must not exceed 255 bytes. This length is for the data only and is not affected by the null byte, should it be present. The maximum data length of a column is 255 bytes, whether the column is nullable or not. No LOB, LONG VARCHAR, LONG VARGRAPHIC, DATALINK, distinct type based on one of these types, or structured type may be used as part of a primary key, even if the length attribute of the column is small enough to fit within the 255 byte limit (SQLSTATE 42962).

The set of columns in the primary key cannot be the same as the set of columns of a unique key (SQLSTATE 01543). [81]

Only one primary key can be defined on a table.

A primary key cannot be specified if the table is a subtable (SQLSTATE 429B3) since the primary key is inherited from the supertable.

The description of the table as recorded in the catalog includes the primary key and its primary index. A unique index will automatically be created for the columns in the sequence specified with ascending order for each column. The name of the index will be the same as the *constraint-name* if this does not conflict with an existing index in the schema where the table is created. If the index name conflicts, the name will be SQL, followed by a character timestamp (*yymmddhhmmssxxx*), with SYSIBM as the schema name.

If the table has a partitioning key, the columns of a *unique-constraint* must be a superset of the partitioning key columns; column order is unimportant.

referential-constraint
Defines a referential constraint.

CONSTRAINT *constraint-name*
Names the referential constraint. See page 734.