CREATE PROCEDURE (SQL - native)

The CREATE PROCEDURE statement defines an SQL procedure, or a version of a procedure, at the current server and specifies the source statements for the procedure.

If the OR REPLACE clause is specified and the procedure already exists:

- If the VERSION clause is not specified, the procedure definition is replaced.
- If the VERSION clause is specified and the identified version exists, the version is replaced. Otherwise the version is added to the procedure definition.

Native SQL procedures can contain SQL control statements. For information about the SQL control statements that are supported in native SQL procedures, refer to Chapter 8, “SQL procedural language (SQL PL),” on page 2189.

Invocation

This statement can only be dynamically prepared, and the DYNAMICRULES run behavior must be specified implicitly or explicitly.

Authorization

To create a new procedure in the implicit or explicit schema, the privilege set that is defined below must include at least one of the following:

- The CREATIN privilege on the schema
- System DBADM authority
- SYSCTRL authority
- SYSADM authority

- Installation SYSOPR authority (when the current SQLID of the process is set to SYSINSTL)

The authorization ID that matches the schema name implicitly has the CREATEIN privilege on the schema.

To replace a procedure, the privilege set that is defined below must include at least one of the following:

- Ownership of the procedure
- Both the DROPIN and CREATEIN privilege on the schema
- System DBADM authority
- SYSCTRL authority
- SYSADM authority
- Installation SYSOPR authority (when the current SQLID of the process is set to SYSINSTL)

To add or replace a version of a procedure, the privilege set that is defined below must include at least one of the following:

- Ownership of the procedure
- The ALTERIN privilege on the schema
- System DBADM authority
- SYSCTRL authority
- SYSADM authority
- Installation SYSOPR authority (when the current SQLID of the process is set to SYSINSTL)

If a user-defined type is referenced (as the data type of a parameter or SQL variable), the privilege set must also include at least one of the following privileges or authorities:

- Ownership of the user-defined type
- The USAGE privilege on the user-defined type
- System DBADM authority
- DATAACCESS authority
- SYSADM authority

If the procedure uses a table as a parameter, the privilege set must also include at least one of the following privileges or authorities:

- Ownership of the table
- The SELECT privilege on the table
- DATAACCESS authority
- SYSADM authority

If you specify the WLM ENVIRONMENT FOR DEBUG MODE clause, RACF or an external security product is invoked to check the required authority for defining programs in the WLM environment. If the WLM environment access is protected in RACF, the privilege set must include the required authority.

To create or replace a procedure, or add or replace a version of a procedure, the privilege set must include the required authorization to add a new package or a new version of an existing package depending on the value of the BIND NEW PACKAGE field on installation panel DSNTIPP, or the privilege set must include SYSADM or SYSCTRL or system DBADM authority. The owner of the procedure package must have the privileges that are required to execute the statements in *SQL-routine-body*.

If the SECURED option is specified, at least one of the following privileges is required:

- SECADM authority
- CREATE_SECURE_OBJECT privilege

If the SEPARATE SECURITY subsystem parameter is set to NO, SYSADM authority has implicit SECADM authority.

Additional authorization might be required on the SYSDUMMY*x* tables depending on the content of the procedure definition. For more information, see SYSDUMMYx tables (Introduction to Db2 for z/OS).

If the authorization ID that is used to create the procedure has the installation SYSADM authority or the installation SYSOPR authority and if the current SQLID is set to SYSINSTL, the procedure is identified as system-defined procedure.

Privilege set:

The privilege set is the privileges that are held by the SQL authorization ID of the process unless the process is within a trusted context and the ROLE AS OBJECT OWNER clause is specified. In that case, the privileges set is the privileges that are held by the role that is associated with the primary authorization ID of the process.

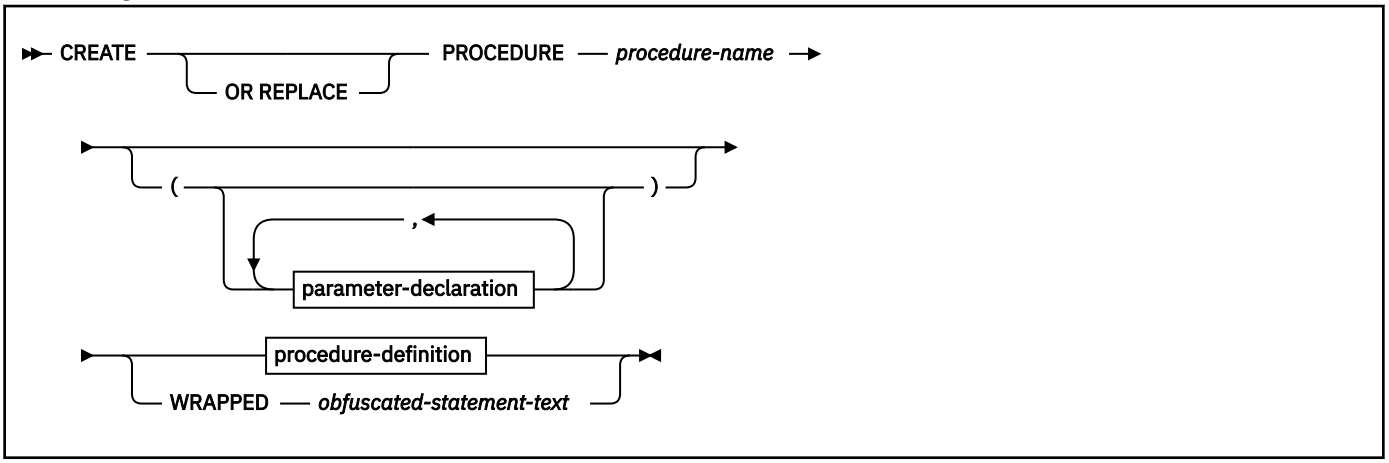
When a CREATE PROCEDURE statement is issued in a trusted context that is defined to use the role as the object owner and the type of the owner is not specified, the default package owner is determined as follows:

- If the PACKAGE OWNER option is not specified, the role associated with the user becomes the package owner.
- If the PACKAGE OWNER option is specified, the role specified in the PACKAGE OWNER option becomes the package owner. In a trusted context, the specified PACKAGE OWNER must be a role.

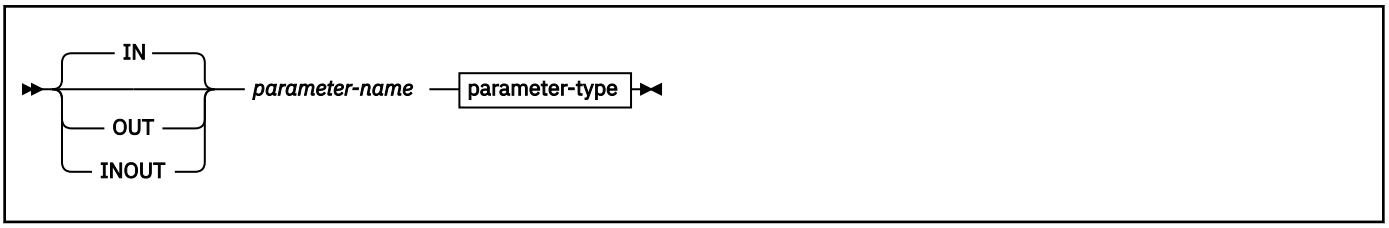
Otherwise, if the type of the owner is not specified, the default package owner is determined as follows:

- If the PACKAGE OWNER option is not specified, the procedure owner becomes the package owner.
- If the PACKAGE OWNER option is specified, the *authorization-name* specified in the PACKAGE OWNER option becomes the package owner.

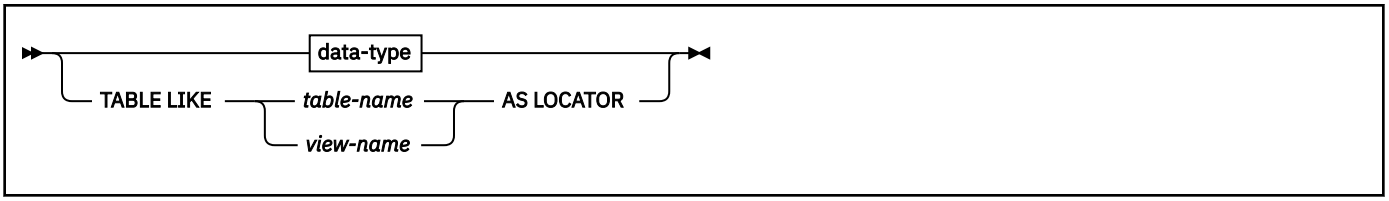
Syntax



parameter-declaration:

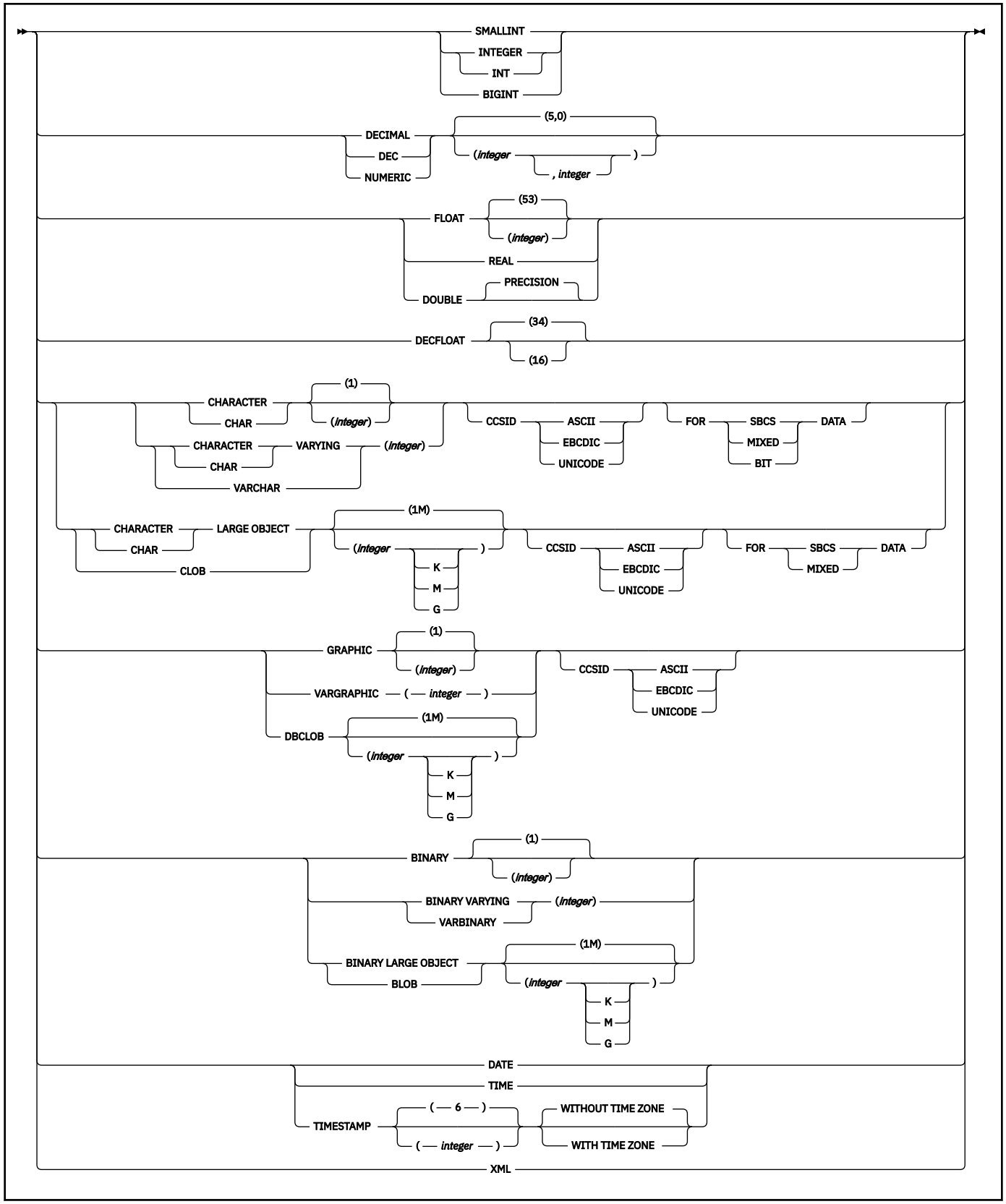


parameter-type:

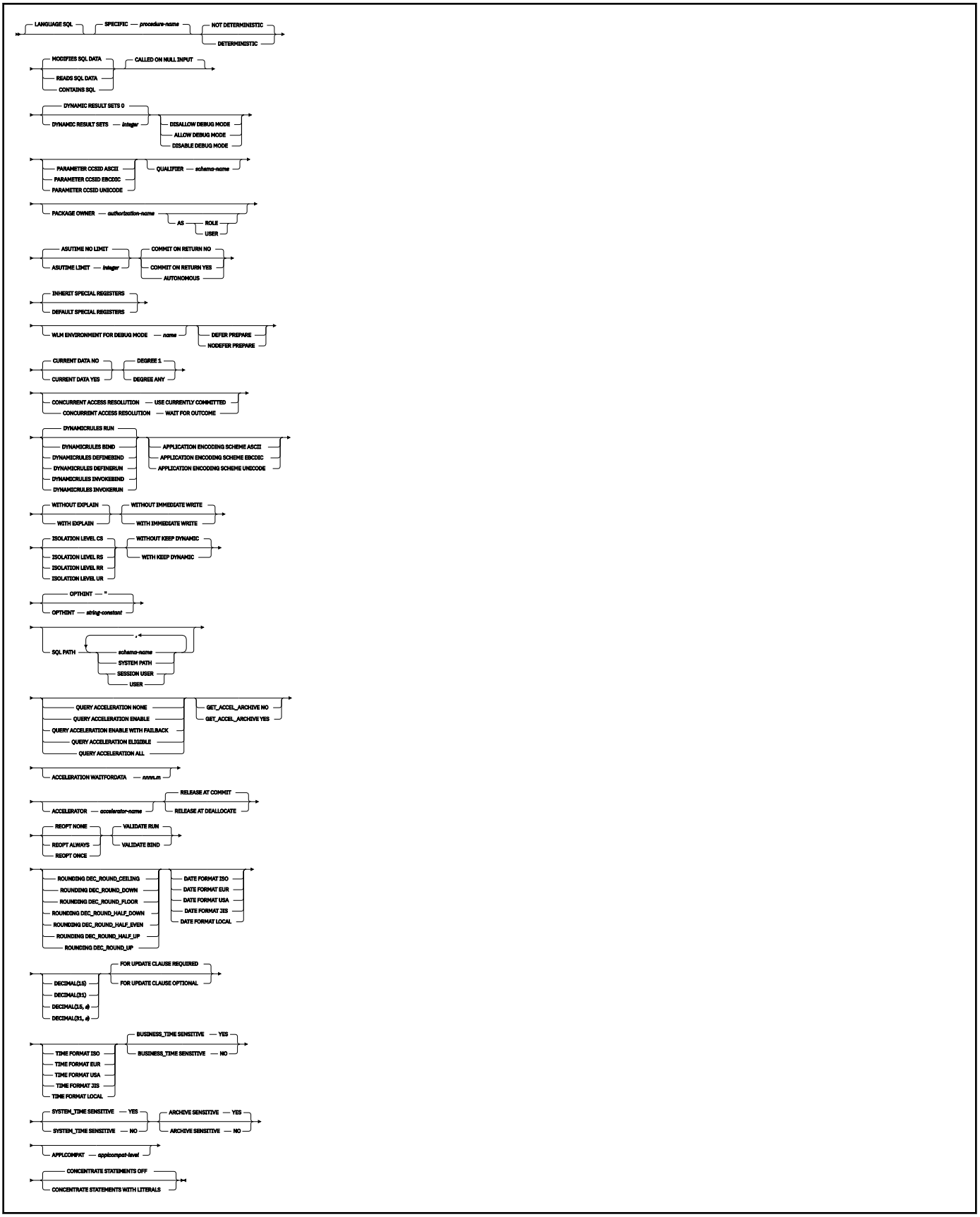


data-type:

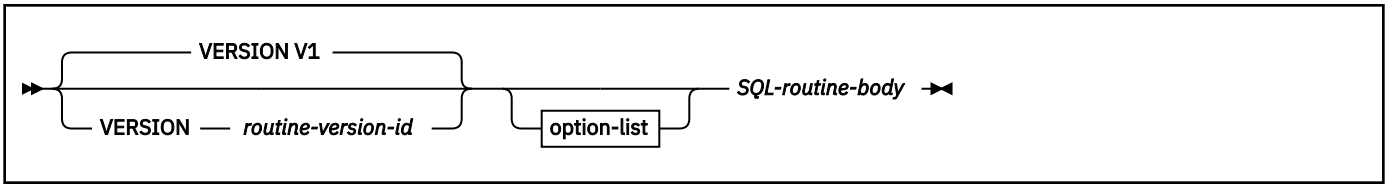
built-in-type:



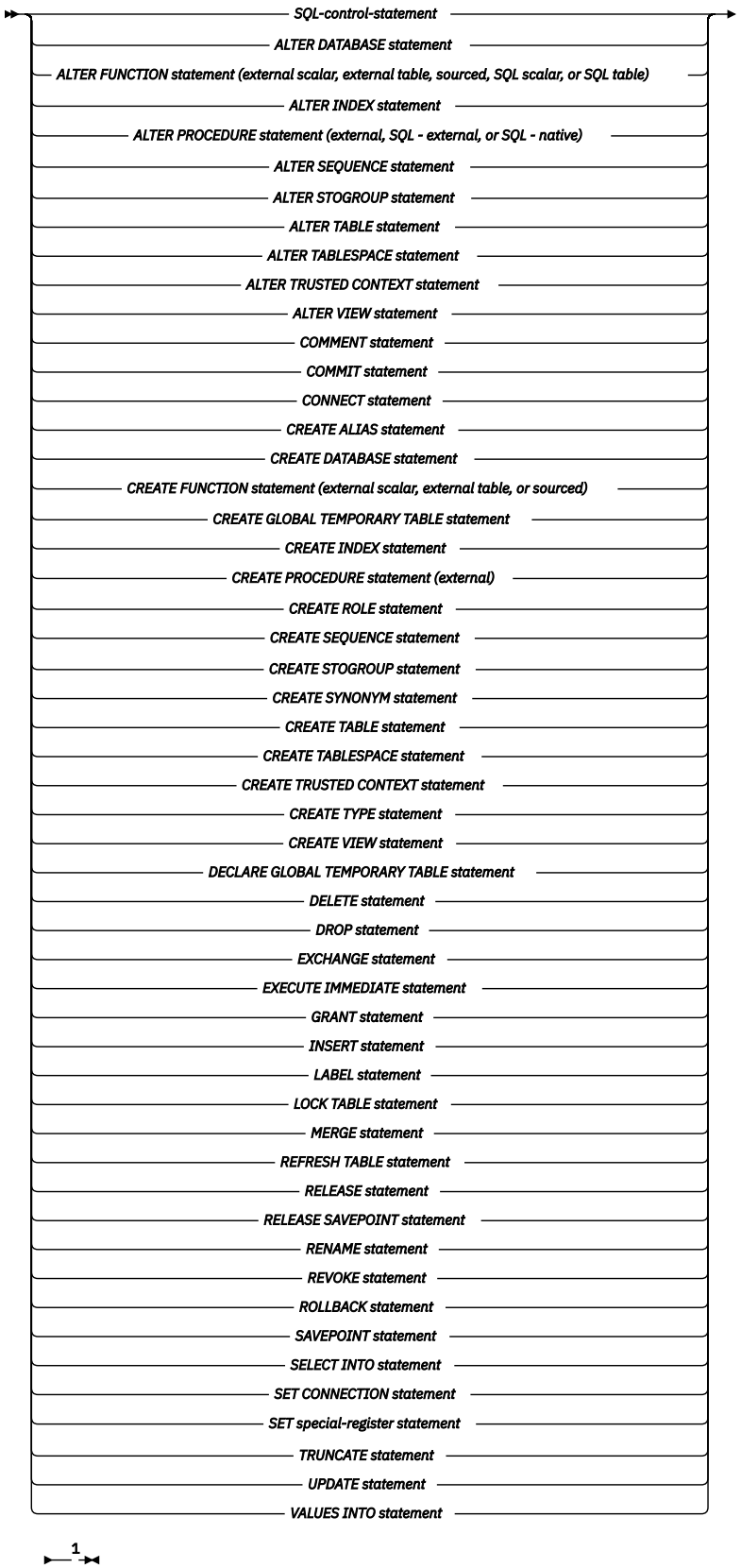
option-list: (The options can be specified in any order, but each one can only be specified one time.)



procedure-definition:



SQL-routine-body:



Notes:

[1] An ALTER FUNCTION statement (SQL scalar) or an ALTER PROCEDURE statement (SQL native) with an ADD VERSION or REPLACE clause are not allowed in an *SQL-routine-body*.

Description

OR REPLACE

Specifies that if the procedure exists at the current server, the existing definition is replaced, or that a version of the procedure is added or replaced. This option is ignored if a definition for the procedure does not exist at the current server. If a procedure exists with the specified name, the existing procedure must be a native SQL procedure.

If the procedure does not yet exist and the VERSION keyword is not specified, the procedure is created with the initial version of the procedure (V1).

If the procedure exists and the VERSION keyword is not specified, the procedure is replaced. The existing definition is dropped before the new definition is replaced in the catalog with the exception that the privileges that were granted on the procedure are not affected. The procedure is re-created as follows:

- Any existing comments are discarded
- The definition of the procedure can change
- The timestamp that is associated with the procedure definition is updated
- The owner of the procedure can change
- System default values are used for options that are not explicitly specified, even if those options were explicitly specified when the procedure that is being replaced was originally defined.

To replace an existing procedure, the procedure must not be defined with more than a single version, or with a single version for which the version id is other than V1. Additionally, one of the following conditions must be met:

- The SPECIFIC clause must be specified with the procedure name
- The signature of the new definition must match the signature of the existing procedure definition, except for parameter names

If the procedure exists with the specified version of the procedure, and the VERSION keyword is specified, the version is replaced in the catalog as if an ALTER PROCEDURE statement had been issued with the REPLACE VERSION clause. The procedure owner is not changed. Binding the replaced version of the procedure might result in a new access path even if the routine body is not changed.

When you replace a version of a procedure, the data types, CCSID specifications, and character data attributes (FOR BIT/SBCS/MIXED DATA) of the parameters must be the same as the attributes of the corresponding parameters for the currently active version of the procedure. The parameter list must not include a table parameter. For options that are not explicitly specified, the system default values for those options are used, even if those options were explicitly specified when the version of the procedure that is being replaced was originally defined. This is not the case for versions of the procedure that specified DISABLE DEBUG MODE. If DISABLE DEBUG MODE is specified for a version of a procedure, the option cannot be changed using a CREATE statement with the OR REPLACE clause.

If the procedure exists but version *routine-version-id* does not exist, and the VERSION keyword is specified, the specified version is created. *routine-version-id* is the version identifier for the new version of the procedure. The new version is defined as if an ALTER PROCEDURE statement had been issued with an ADD VERSION clause. The procedure owner is not changed. When a procedure definition is replaced, any existing comments in the catalog for that definition of the procedure are removed.

When you add a new version of a procedure, the data types, CCSID specifications, and character data attributes (FOR BIT/SBCS/MIXED DATA) of the parameters must be the same as the attributes of the corresponding parameters for the currently active version of the procedure. The parameter list must not include a table parameter. The parameter names can differ from the other versions of the procedure. For options that are not explicitly specified, the system default values are used.

procedure-name

Names the procedure. The name, including the implicit or explicit schema name, must not identify a procedure that exists at the current server. However, you can specify an existing procedure name when the OR REPLACE keyword is also specified.

The name must not identify an existing wrapped procedure if the CREATE statement adds or replaces a version of that procedure.

The schema name can be 'SYSTOOLS' or 'SYSFUN' if the privilege set includes the SYSADM or SYSCTRL privilege. Otherwise, the schema name must not begin with 'SYS' unless the schema name is 'SYSADM', 'SYSIBMADM', or 'SYSPROC'.

(*parameter-declaration,...*)
Specifies the number of parameters of the procedure, the data type and usage of each parameter, and the name of each parameter for the version of the procedure that is being defined. The number of parameters and the specified data type and usage of each parameter must match the data types in the corresponding position of the parameter for all other versions of this procedure. Synonyms for data types are considered to be a match. All parameters are nullable.

IN, OUT, and INOUT specify the usage of the parameter. The usage of the parameters must match the implicit or explicit usage of the parameters of other versions of the same procedure.

IN
Identifies the parameter as an input parameter to the procedure. The value of the parameter on entry to the procedure is the value that is returned to the calling SQL application, even if changes are made to the parameter within the procedure.

IN is the default.

OUT
Identifies the parameter as an output parameter that is returned by the procedure. If the parameter is not set within the procedure, the null value is returned.

INOUT
Identifies the parameter as both an input and output parameter for the procedure. If the parameter is not set within the procedure, its input value is returned.

parameter-name
Names the parameter for use as an SQL variable. A parameter name cannot be the same as the name of any other parameter for this version of the procedure. The name of the parameter in this version of the procedure can be different than the name of the corresponding parameter for other versions of this procedure.

built-in-type
The data type of the parameter is a built-in data type.

For more information on the data types, including the subtype of character data types (the FOR *subtype* DATA clause), see "*built-in-type*" in “CREATE TABLE” on page 1639. However, the varying length string data types have different maximum lengths than for the CREATE TABLE statement. The maximum lengths for parameters (and SQL variables) are as follows: 32704 for VARCHAR or VARBINARY, and 16352 for VARGRAPHIC.

For parameters with a character or graphic data type, the PARAMETER CCSID clause or CCSID clause indicates the encoding scheme of the parameter. If you do not specify either of these clauses, the encoding scheme is the value of field DEF ENCODING SCHEME on installation panel DSNTIPF.

Although an input parameter with a character data type has an implicitly or explicitly specified subtype (BIT, SBCS, or MIXED), the value that is actually passed in the input parameter can have any subtype. Therefore, conversion of the input data to the subtype of the parameter might occur when the procedure is called. With ASCII or EBCDIC, an error occurs if mixed data that actually contains DBCS characters is used as the value for an input parameter that is declared with an SBCS subtype.

Parameters with a datetime data type or a distinct type are passed to the function as a different data type:

- A datetime type parameter is passed as a character data type, and the data is passed in ISO format. The encoding scheme for a datetime type parameter is the same as the implicitly or explicitly specified encoding scheme of any character or graphic string parameters. If no character or graphic string parameters are passed, the encoding scheme is the value of field DEF ENCODING SCHEME on installation panel DSNTIPF.
- A distinct type parameter is passed as the source type of the distinct type.

AS LOCATOR
Specifies that a locator to the value of the parameter is passed to the procedure instead of the actual value. Specify AS LOCATOR only for parameters with a LOB data type or a distinct type based on a LOB data type. Passing locators instead of values can result in fewer bytes being passed to the procedure, especially when the value of the parameter is very large.

The AS LOCATOR clause has no effect on determining whether data types can be promoted.

distinct-type-name
The data type of the input parameter is a distinct type. Any length, precision, scale, subtype, or encoding scheme attributes for the parameter are those of the source type of the distinct type. The distinct type must not be based on a LOB data type.

array-type-name
The data type of the input parameter is a user-defined array type.

If you specify *array-type-name* without a schema name, Db2 resolves the array type by searching the schemas in the SQL path.

TABLE LIKE *table-name* AS LOCATOR
Specifies that the parameter is a transition table. However, when the procedure is called, the actual values in the transition table are not passed to the procedure. A single value is passed instead. This single value is a locator to the table, which the procedure uses to access the columns of the transition table. The table that is identified can contain XML columns; however, the procedure cannot reference those XML columns. A procedure with a table parameter can only be invoked from the triggered action of a trigger.

The TABLE LIKE clause must not be specified when the CREATE statement adds or replaces a version of an existing procedure.

VERSION *routine-version-id*
Specifies the version identifier for the version of the procedure that is to be defined. See "Naming conventions" on page 79 for information about specifying *routine-version-id*. You can use an ALTER PROCEDURE statement with the ADD VERSION clause or the BIND DEPLOY command to create additional versions of the procedure.

V1 is the default version identifier.

Important: Do not create additional versions of procedures that are supplied with Db2 by specifying the VERSION keyword. Only versions that are supplied with Db2 are supported. Additional versions of such routines cause the installation and configuration of the supplied routines to fail.

LANGUAGE SQL
Specifies that the procedure is written in the Db2 SQL procedural language.

SPECIFIC *procedure-name*

Specifies the procedure name as the specific name for the procedure. The name must be the same as the procedure name.

If you do not specify a schema name, it is the same as the explicit or implicit schema name of the procedure name (*procedure-name*). If you specify a schema name, it must be the same as the explicit or implicit schema name of the procedure name.

If you do not specify the SPECIFIC clause, the specific name is the name of the procedure. The specific name is stored in the SPECIFIC column of the SYSROUTINES catalog table.

Specify the SPECIFIC clause when replacing an existing procedure in the following situations:

- The parameter list of the existing procedure includes a table parameter.
- The CREATE statement specifies changes to the parameter list other than for parameter names.

DETERMINISTIC or NOT DETERMINISTIC
Specifies whether the procedure returns the same results each time it is called with the same IN and INOUT arguments.

DETERMINISTIC
The procedure always returns the same results each time it is called with the same IN and INOUT arguments if the data that is referenced in the database has not changed.

NOT DETERMINISTIC
The procedure might not return the same result each time it is called with the same IN and INOUT arguments, even when the data that is referenced in the database has not changed.

NOT DETERMINISTIC is the default.

Db2 does not verify that the procedure code is consistent with the specification of DETERMINISTIC or NOT DETERMINISTIC.

MODIFIES SQL DATA, READS SQL DATA, or CONTAINS SQL
Specifies the classification of SQL statements and nested routines that this routine can execute or invoke. The database manager verifies that the SQL statements issued by the procedure, and all routines locally invoked by the routine, are consistent with this specification; the verification is not performed when nested remote routines are invoked. For the classification of each statement, see "SQL statement data access classification for routines" on page 2253.

MODIFIES SQL DATA
Specifies that the procedure can execute any SQL statement except statements that are not supported in procedures.

MODIFIES SQL DATA is the default.

READS SQL DATA
Specifies that the procedure can execute statements with a data access indication of READS SQL DATA or CONTAINS SQL. The procedure cannot execute SQL statements that modify data.

CONTAINS SQL
Specifies that the procedure can execute only SQL statements with a data access indication of CONTAINS SQL. The procedure cannot execute statements that read or modify data.

CALLED ON NULL INPUT
Specifies that the procedure will be called if any, or even if all parameter values are null.

DYNAMIC RESULT SETS *integer*
Specifies the maximum number of query result sets that the procedure can return. The default is DYNAMIC RESULT SETS 0, which indicates that there are no result sets. The value must be between 0 and 32767.

ALLOW DEBUG MODE, DISALLOW DEBUG MODE, or DISABLE DEBUG MODE
Specifies whether this version of the routine can be run in debugging mode. The default is determined using the value of the CURRENT DEBUG MODE special register.

ALLOW DEBUG MODE
Specifies that this version of the routine can be run in debugging mode. When this version of the routine is invoked and debugging is attempted, a WLM environment must be available.

DISALLOW DEBUG MODE
Specifies that this version of the routine cannot be run in debugging mode.

You can use an ALTER statement to change this option to ALLOW DEBUG MODE for this initial version of the routine.

DISABLE DEBUG MODE
Specifies that this version of the routine can never be run in debugging mode.

This version of the routine cannot be changed to specify ALLOW DEBUG MODE or DISALLOW DEBUG MODE after this version of the routine has been created or altered to use DISABLE DEBUG MODE. To change this option, drop the routine and create it again using the option that you want. An alternative to dropping and recreating the routine is to create a version of the routine that uses the option that you want and making that version the active version.

When DISABLE DEBUG MODE is in effect, the WLM ENVIRONMENT FOR DEBUG MODE is ignored.

PARAMETER CCSID
Indicates whether the encoding scheme for character or graphic string parameters is ASCII, EBCDIC, or UNICODE. The default encoding scheme is the value that is specified in the CCSID clauses of the parameter list, or in the field DEF ENCODING SCHEME on installation panel DSNTIPF.

This clause provides a convenient way to specify the encoding scheme for character or graphic string parameters. If individual CCSID clauses are specified for individual parameters in addition to this PARAMETER CCSID clause, the value that is specified in all of the CCSID clauses must be the same value that is specified in this clause.

If the data type for a parameter is a user-defined distinct type that is defined as a character or graphic type string, the CCSID of the distinct type must be the same as the value that is specified in this clause.

If the data type for a parameter is a user-defined array type that is defined with character or graphic string array elements, or a character string array index, the CCSID of these array attributes must be the same as the value that is specified in this clause.

This clause also specifies the encoding scheme that will be used for system-generated parameters of the routine.

QUALIFIER *schema-name*
Specifies the implicit qualifier that is used for unqualified object names that are referenced in the procedure body. See "Unqualified alias, index, JAR file, mask, permission, sequence, table, trigger, and view names" on page 86 for information about how the default for this option is determined.

PACKAGE OWNER *authorization-name*

FL 500

Specifies the owner of the package that is associated with the version of the routine. The default is that the *authorization-name* is a role if the process is running in a trusted context defined with the role as object owner and qualifier attributes. Otherwise, the default is the SQL authorization ID of the process.

AS ROLE
Specifies that *authorization-name* is a role that exists at the current server.

AS USER
Specifies that *authorization-name* is an authorization ID.

ASUTIME
Specifies the total amount of processor time, in CPU service units, that a single invocation of a routine can run. The value is unrelated to the ASUTIME column of the resource limit specification table.

When you are debugging a routine, setting a limit can be helpful in case the routine gets caught in a loop. For information on service units, see z/OS MVS Initialization and Tuning Guide.

NO LIMIT
Specifies that there is no limit on the service units.

NO LIMIT is the default.

LIMIT *integer*
The limit on the number of CPU service units is a positive *integer* in the range 1 - 2 147 483 647. If the procedure uses more service units than the specified value, Db2 cancels the procedure. The CPU cycles that are consumed by parallel tasks in a procedure do not contribute towards the specified ASUTIME LIMIT.

COMMIT ON RETURN NO, COMMIT ON RETURN YES, or AUTONOMOUS
Indicates whether Db2 commits the transaction immediately on return from the procedure.

COMMIT ON RETURN NO
Db2 does not issue a commit when the procedure returns. NO is the default.

COMMIT ON RETURN YES,
Db2 issues a commit when the procedure returns if the following statements are true:

- The SQLCODE that is returned by the CALL statement is not negative.
- The procedure is not in a must-abort state.

The commit operation includes the work that is performed by the calling application process and by the procedure.

If the procedure returns result sets, the cursors that are associated with the result sets must have been defined as WITH HOLD to be usable after the commit.

AUTONOMOUS
Db2 executes the SQL procedure in a unit of work that is independent from the calling application. When this option is specified the procedure follows the rules of the COMMIT ON RETURN YES option before returning to the calling application. However, it does not commit changes in the calling application. When autonomous is specified:

- DYNAMIC RESULT SETS 0 must be in effect.
- Stored procedure parameters must not be defined as:
  - A LOB type
  - The XML data type
  - A distinct data type that is based on a LOB or XML value
  - An array type that is defined with array elements that are a LOB type

A value must not be assigned to a global variable when an autonomous procedure is executing.

INHERIT SPECIAL REGISTERS or DEFAULT SPECIAL REGISTERS
Specifies how special registers are set on entry to the routine.

INHERIT SPECIAL REGISTERS
Specifies that the values of special registers are inherited, according to the rules that are listed in the table for characteristics of special registers in a routine in Table 47 on page 212.

INHERIT SPECIAL REGISTERS is the default.

DEFAULT SPECIAL REGISTERS
Specifies that special registers are initialized to the default values, as indicated by the rules in the table for characteristics of special registers in a routine in Table 47 on page 212.

WLM ENVIRONMENT FOR DEBUG MODE *name*
Specifies the WLM (workload manager) application environment that is used by Db2 when debugging the routine. The *name* of the WLM environment is an SQL identifier.

If you do not specify WLM ENVIRONMENT FOR DEBUG MODE, Db2 uses the default WLM-established stored procedure address space specified at installation time.

You must have the appropriate authority for the WLM application environment.

The WLM ENVIRONMENT FOR DEBUG MODE value is ignored when DISABLE DEBUG MODE is in effect.

DEFER PREPARE or NODEFER PREPARE
Specifies whether to defer preparation of dynamic SQL statements that refer to remote objects, or to prepare them immediately.

The default depends on the value in effect for the REOPT option. If REOPT NONE is in effect, the default is inherited from the plan at run time. Otherwise, the default is DEFER PREPARE.

DEFER PREPARE
Specifies that the preparation of dynamic SQL statements that refer to remote objects will be deferred.

For considerations for distributed processing, see DEFER and NODEFER bind options (Db2 Commands).

NODEFER PREPARE
Specifies that the preparation of dynamic SQL statements that refer to remote objects will not be deferred.

CURRENT DATA YES or CURRENT DATA NO
Specifies whether to require data currency for read-only and ambiguous cursors when the isolation level of cursor stability is in effect. CURRENT DATA also determines whether block fetch can be used for distributed, ambiguous cursors.

CURRENT DATA YES
Specifies that data currency is required for read-only and ambiguous cursors. Db2 acquired page or row locks to ensure data currency. Block fetch is ignored for distributed, ambiguous cursors.

CURRENT DATA NO
Specifies that data currency is not required for read-only and ambiguous cursors. Block fetch is allowed for distributed, ambiguous cursors. Use of CURRENT DATA NO is not recommended if the routine attempts to dynamically prepare and execute a DELETE WHERE CURRENT OF statement against an ambiguous cursor after that cursor is opened. You receive a negative SQLCODE if your routine attempts to use a DELETE WHERE CURRENT OF statement for any of the following cursors:

- A cursor that is using block fetch
- A cursor that is using query parallelism
- A cursor that is positioned on a row that is modified by this or another application process

CURRENT DATA NO is the default.

DEGREE
Specifies whether to attempt to run a query using parallel processing to maximize performance.

1
Specifies that parallel processing should not be used.

1 is the default.

ANY
Specifies that parallel processing can be used.

CONCURRENT ACCESS RESOLUTION
Specifies the whether processing uses only committed data or whether it will wait for commit or rollback of data that is in the process of being updated.

WAIT FOR OUTCOME
Specifies that processing will wait for the commit or rollback of data that is in the process of being updated.

USE CURRENTLY COMMITTED
Specifies that processing use the currently committed version of the data when data that is in the process of being updated is encountered. USE CURRENTLY COMMITTED is applicable on scans that access tables that are defined in universal table spaces with row or page level lock size.

When there is lock contention between a read transaction and an insert transaction, USE CURRENTLY COMMITTED is applicable to scans with isolation level CS or RS. Applicable scans include intent read scans for read-only and ambiguous queries and for updatable cursors. USE CURRENTLY COMMITTED is also applicable to scans initiated from WHERE predicates of UPDATE or DELETE statements and the subselect of INSERT statements.

When there is lock contention is between a read transaction and a delete transaction, USE CURRENTLY COMMITTED is applicable to scans with isolation level CS and when CURRENT DATA NO is specified.

DYNAMICRULES
Specifies the values that apply, at run time, for the following dynamic SQL attributes:

- The authorization ID that is used to check authorization
- The qualifier that is used for unqualified objects
- The source for application programming options that Db2 uses to parse and semantically verify dynamic SQL statements

DYNAMICRULES also specifies whether dynamic SQL statements can include GRANT, REVOKE, ALTER, CREATE, DROP, and RENAME statements.

In addition to the value of the DYNAMICRULES clause, the run time environment of a routine controls how dynamic SQL statements behave at run time. The combination of the DYNAMICRULES value and the run time environment determines the value for the dynamic SQL attributes. That set of attribute values is called the dynamic SQL statement behavior. The following values can be specified:

RUN
Specifies that dynamic SQL statements are to be processed using run behavior.

RUN is the default.

BIND
Specifies that dynamic SQL statements are to be processed using bind behavior.

DEFINEBIND
Specifies that dynamic SQL statements are to be processed using either define behavior or bind behavior.

DEFINERUN
Specifies that dynamic SQL statements are to be processed using either define behavior or run behavior.

INVOKEBIND
Specifies that dynamic SQL statements are to be processed using either invoke behavior or bind behavior.

INVOKERUN
Specifies that dynamic SQL statements are to be processed using either invoke behavior or run behavior.

See For information on the effects of these options, see "Authorization IDs and dynamic SQL" on page 94.

APPLICATION ENCODING SCHEME
Specifies the default encoding scheme for SQL variables in static SQL statements in the routine body. The value is used for defining an SQL variable in a compound statement if the CCSID clause is not specified as part of the data type, and the PARAMETER CCSID routine option is not specified.

ASCII
Specifies that the data is encoded using the ASCII CCSIDs of the server.

EBCDIC
Specifies that the data is encoded using the EBCDIC CCSIDs of the server.

UNICODE
Specifies that the data is encoded using the Unicode CCSIDs of the server.

See the ENCODING bind option in ENCODING bind option (Db2 Commands) for information about how the default for this option is determined.

WITH EXPLAIN or WITHOUT EXPLAIN
Specifies whether information will be provided about how SQL statements in the routine will execute.

WITHOUT EXPLAIN
Specifies that information will not be provided about how SQL statements in the routine will execute.

You can get EXPLAIN output for a statement that is embedded in a routine that is specified using WITHOUT EXPLAIN by embedding the SQL statement EXPLAIN in the routine body. Otherwise, the value of the EXPLAIN option applies to all explainable SQL statements in the routine body, and to the fullselect portion of any DECLARE CURSOR statements.

WITHOUT EXPLAIN is the default.

WITH EXPLAIN
Specifies that information will be provided about how SQL statements in the routine will execute. Information is inserted into the table *owner*.PLAN_TABLE. *owner* is the authorization ID of the owner of the routine. Alternatively, the authorization ID of the owner of the routine can have an alias as *owner*.PLAN_TABLE that points to the base table, PLAN_TABLE. *owner* must also have the appropriate SELECT and INSERT privileges on that table. WITH EXPLAIN does not obtain information for statements that access remote objects. PLAN_TABLE must have a base table and can have multiple aliases with the same table name, PLAN_TABLE, but have different schema qualifiers. It cannot be a view or a synonym and should exist before the CREATE statement is processed. In all inserts to *owner*.PLAN_TABLE, the value of QUERYNO is the statement number that is assigned by Db2.

The WITH EXPLAIN option also populates two optional tables, if they exist: DSN_STATEMNT_TABLE and DSN_FUNCTION_TABLE. DSN_STATEMNT_TABLE contains an estimate of the processing cost for an SQL statement and DSN_FUNCTION_TABLE contains information about function resolution. For more information, see EXPLAIN tables (Db2 Performance).

For more information about the EXPLAIN statement, including a description of the tables that are populated by the WITH EXPLAIN option, see "EXPLAIN" on page 1904.

WITH IMMEDIATE WRITE or WITHOUT IMMEDIATE WRITE
Specifies whether immediate writes are to be done for updates that are made to group buffer pool dependent page sets or partitions. This option is only applicable for data sharing environments. The IMMEDWRITE subsystem parameter has no affect of this option. IMMEDWRITE bind option (Db2 Commands) shows the implied hierarchy of the IMMEDWRITE bind option (which is similar to this routine option) as it affects run time.

WITHOUT IMMEDIATE WRITE
Specifies that normal write activity is performed. Updated pages that are group buffer pool dependent are written at or before phase one of commit or at the end of abort for transactions that have been rolled back.

WITHOUT IMMEDIATE WRITE is the default.

WITH IMMEDIATE WRITE
Specifies that updated pages that are group buffer pool dependent are immediately written as soon as the buffer update completes. Updated pages are written immediately even if the buffer is updated during forward progress or during the rollback of a transaction. WITH IMMEDIATE WRITE might impact performance.

ISOLATION LEVEL RR, RS, CS, or UR
Specifies how far to isolate the routine from the effects of other running applications. For information about isolation levels, see Choosing an ISOLATION option (Db2 Performance).

RR
Specifies repeatable read.

RS
Specifies read stability.

CS
Specifies cursor stability. CS is the default.

UR
Specifies uncommitted read.

WITH KEEP DYNAMIC or WITHOUT KEEP DYNAMIC
Specifies whether Db2 keeps dynamic SQL statements after commit points.

WITHOUT KEEP DYNAMIC
Specifies that Db2 does not keep dynamic SQL statements after commit points.

WITHOUT KEEP DYNAMIC is the default.

WITH KEEP DYNAMIC
Specifies that Db2 keeps dynamic SQL statements after commit points. If you specify WITH KEEP DYNAMIC, the application does not need to prepare an SQL statement after every commit point. Db2 keeps the dynamic SQL statement until one of the following occurs:

- The application process ends
- A rollback operations occurs
- The application executes an explicit PREPARE statement with the same statement identifier as the dynamic SQL statement

If you specify WITH KEEP DYNAMIC, and the prepared statement cache is active, the Db2 subsystem keeps a copy of the prepared statement in the cache. If the prepared statement cache is not active, the subsystem keeps only the SQL statement string past a commit point. If the application executes an OPEN, EXECUTE, or DESCRIBE operation for that statement, the statement is implicitly prepared.

If you specify WITH KEEP DYNAMIC, DDF server threads that are used to execute procedures or packages that have this option in effect will remain active. Active DDF server threads are subject to idle thread timeout. For more information see IDLE THREAD TIMEOUT field (IDTHTOIN subsystem parameter) (Db2 Installation and Migration).

If you specify WITH KEEP DYNAMIC, you must not specify REOPT ALWAYS. WITH KEEP DYNAMIC and REOPT ALWAYS are mutually exclusive. However, you can specify WITH KEEP DYNAMIC and REOPT ONCE.

Use WITH KEEP DYNAMIC to improve performance if your DRDA client application uses a cursor that is defined as WITH HOLD. The Db2 subsystem automatically closes a held cursor when there are no more rows to retrieve, which eliminates an extra network message.

OPTHINT *string-constant*
Specifies whether query optimization hints are used for static SQL statements that are contained within the body of the routine.

string-constant is a character string of up to 128 bytes in length, which is used by the Db2 subsystem when searching the PLAN_TABLE for rows to use as input. The default value is an empty string, which indicates that the Db2 subsystem does not use optimization hints for static SQL statements.

Optimization hints are only used if optimization hints are enabled for your system. For more information, see OPTIMIZATION HINTS field (OPTHINTS subsystem parameter) (Db2 Installation and Migration).

SQL PATH
Specifies the SQL path that Db2 uses to resolve unqualified user-defined type, function, and procedure names in the procedure body. The default value is "SYSIBM", "SYSFUN", "SYSPROC", "SYSIBMADM", and the value of the QUALIFIER option, which is the qualifier for the procedure that is the target of the statement.

Schemas "SYSIBM", "SYSFUN", "SYSPROC", "SYSIBMADM" do not need to be explicitly specified. If any of these schemas is not explicitly specified, it is implicitly assumed at the beginning the SQL path.

Db2 calculates the length by taking each *schema-name* specified and removing any trailing blanks from it, adding two delimiters around it, and adding one comma after each schema name except for the last one. The length of the resulting string cannot exceed the length of the CURRENT SCHEMA special register. If you do not specify the "SYSIBM", "SYSFUN", "SYSPROC", "SYSIBMADM", schemas, they are not included in the length of the SQL path. If the total length of the SQL path exceeds the length of the CURRENT PATH special register, Db2 returns an error for the CREATE statement.

For more information, see:

"SQL path" on page 85
"CURRENT SCHEMA" on page 203
"CURRENT PATH" on page 197

schema-name
Specifies a schema. Db2 does not validate that the specified schema actually exists when the CREATE statement is processed.

SYSPUBLIC must not be specified for the SQL path.

schema-name-list
Specifies a comma separated list of schema names. The same schema name should not appear more than one time in the list of schema names. The number of schema names that you can specify is limited by the maximum length of the resulting SQL path.

SYSPUBLIC must not be specified for the SQL path.

SYSTEM PATH
Specifies the schema names "SYSIBM", "SYSFUN", "SYSPROC", "SYSIBMADM".

SESSION_USER or USER
Specifies the value of the SESSION_USER or USER special register, which represents a maximum 8 byte (in EBCDIC) *schema-name*. At the time the CREATE statement is processed, this length is included in the total length of the list of schema names that is specified for the PATH bind option.

RELEASE AT
Specifies when to release resources that the procedure uses: either at each commit point or when the procedure terminates.

COMMIT
Specifies that resources will be released at each commit point.

COMMIT is the default.

DEALLOCATE
Specifies that resources will be released only when the thread terminates. DEALLOCATE has no effect on dynamic SQL statements, which always use RELEASE AT COMMIT, with this exception: When you use the RELEASE AT DEALLOCATE clause and the WITH KEEP DYNAMIC clause, and the subsystem is installed with a value of YES for the field CACHE DYNAMIC SQL on installation panel DSNTIP8, the RELEASE AT DEALLOCATE option is honored for dynamic SELECT and data change statements.

Locks that are acquired for dynamic statements are held unit one of the following events occurs:

- The application process ends.
- The application process issues a PREPARE statement with the same statement identifier. (Locks are released at the next commit point).
- The statement is removed from the prepared statement cache because the statement has not been used. (Locks are released at the next commit point).
- An object that the statement is dependent on is dropped or altered, or a privilege that the statement needs is revoked. (Locks are released at the next commit point).

RELEASE AT DEALLOCATE can increase the package or plan size because additional items become resident in the package or plan.

For more information, see Choosing a RELEASE option (Db2 Performance).

REOPT
Specifies if Db2 will determine the access path at run time by using the values of SQL variables or SQL parameters, parameter markers, and special registers.

NONE
Specifies that Db2 does not determine the access path at run time by using the values of SQL variables or SQL parameters, parameter markers, and special registers.

NONE is the default.

ALWAYS
Specifies that Db2 always determines the access path at run time each time an SQL statement is run. Do not specify REOPT ALWAYS with the WITH KEEP DYNAMIC or NODEFER PREPARE clauses.

ONCE
Specifies that Db2 determine the access path for any dynamic SQL statements only once, at the first time the statement is opened. This access path is used until the prepared statement is invalidated or removed from the dynamic statement cache and need to be prepared again.

QUERY ACCELERATION
Specifies whether a static SQL query is bound for acceleration, and if so, with what behavior.

NONE
Specifies that no static SQL query in the application is bound for acceleration or will be accelerated when the application is run.

ENABLE
Specifies that a static SQL query is bound for acceleration if it satisfies the acceleration criteria, including the cost and heuristics criteria. The query is routed to an accelerator when the application runs. Otherwise, if the static query does not satisfy the acceleration criteria, the query is bound for execution in Db2.

If an error condition, such as one of the following examples, occurs while executing the accelerated static query when the application is run, Db2 fails the static query and returns a negative SQL code to the application:

- A failure occurs while running the static query on the accelerator.
- The accelerator returns an error for the query.
- The accelerator is not started and Db2 cannot route the static query to the accelerator for execution.

ENABLE WITH FAILBACK
Results in the same behavior as ENABLE, except if one of the error conditions occurs on the first OPEN of the accelerated static query when the application is run. In this case, instead of failing the static query and returning a negative SQL code to the application, Db2 performs a temporary *statement-level* incremental bind of the query and runs the query in Db2. The application does not see the acceleration failure. Failback to Db2 is not possible after the application does a successful OPEN for the query on the accelerator.

ELIGIBLE
Specifies that a static SQL query is bound for acceleration if the query meets the basic acceleration criteria, regardless of the cost or heuristics criteria. The query is routed to the accelerator when the application runs.

Like the behavior for ENABLE, if an error condition occurs while executing the accelerated static query when the application is run, Db2 fails the static query and returns a negative SQL code to the application.

ALL
Specifies that all of the static SQL queries in the application are to be bound for acceleration and routed to the accelerator when the application runs. If Db2 determines that a static query cannot be bound to run on the accelerator and the query references a user base table or view, the BIND or REBIND PACKAGE operation fails with an error message for that query. (A failure exception is made for declared global temporary tables (DGTTs) and created global temporary tables and (CGTTs) because these tables cannot be accelerated.)

Like the behavior for ENABLE, if an error condition occurs while executing the accelerated static query when the application is run, Db2 fails the static query and returns a negative SQL code to the application.

This bind option does not apply to a *fullselect* or WITH *common-table-expression* that is specified in a RETURN statement for the routine, or in a SET *host-variable-assignment* that is used in the routine. The queries that are specified in these cases cannot be accelerated.

GET_ACCEL_ARCHIVE

Specifies whether a static SQL query that is bound for acceleration retrieves archived data on the accelerator, instead of active data.

NO

Specifies that no static SQL query is bound to retrieve archived data from the accelerator. If the static query also is not bound for acceleration, the query is bound to run in Db2.

If the static query is bound for acceleration because the QUERYACCELERATION bind option was specified, the query is routed to the accelerator when the application runs; however, the query does not retrieve any archived data.

YES

Specifies that if all of the following criteria are met, the query is bound for acceleration and retrieves the archived data on the accelerator when the application runs:

- The QUERYACCELERATION bind option is also specified.
- The static SQL query references an accelerated table that has partitioned data archived on an accelerator.
- The static query satisfies the acceleration criteria that is specified by the QUERYACCELERATION bind option.

If the static query does not satisfy the acceleration criteria that is specified by the QUERYACCELERATION bind option, the BIND or REBIND PACKAGE operation fails with an error message for that query.

This bind option does not apply to a *fullselect* or WITH *common-table-expression* that is specified in a RETURN statement for the routine, or in a SET *host-variable-assignment* that is used in the routine. The queries that are specified in these cases cannot be accelerated.

ACCELERATION WAITFORDATA

Specifies the maximum amount of time, if any, that an accelerator will delay a query while the accelerator waits for the replication of committed Db2 data changes that occurred prior to Db2 running the query.

For static accelerated queries, you must also set the QUERYACCELERATION bind option for this function or procedure to a valid value other than NONE to request that static queries be accelerated. If the QUERYACCELERATION bind option value is set to NONE, the ACCELERATIONWAITFORDATA bind option is accepted and the package is bound with the option value; however, the option will not apply to static SQL queries because no static queries will be accelerated.

For dynamic accelerated queries, specifying the ACCELERATION WAITFORDATA bind option also initializes the CURRENT QUERY ACCELERATION WAITFORDATA special register, which is used for the dynamic queries in the Db2 function or procedure if the function or procedure option DEFAULT SPECIAL REGISTERS is also used. Initializing CURRENT QUERY ACCELERATION WAITFORDATA to a value greater than 0 specifies that Db2 and the accelerator will apply WAITFORDATA delay behavior and restrictions to all dynamic SQL queries to be accelerated from the Db2 function or procedure. The CURRENT QUERY ACCELERATION special register must also have a valid value other than NONE to request that dynamic queries be accelerated.

nnnn.m

Specifies a DECIMAL(5,1) numeric-constant value that specifies the maximum number of seconds that the accelerator will delay a query while the accelerator waits for the replication of committed Db2 data changes that occurred prior to Db2 running the query.

You can specify a value in the range of 0.0 - 3600.0 seconds. For example, a value of 20.0 represents 20.0 seconds (or 20000 milliseconds), and a value of 30.5 represents 30.5 seconds (or 30500 milliseconds). The maximum value of 3600.0 means they the query is delayed for 3600 seconds.

You can also specify the value as an INTEGER numeric-constant value ranging from 0 - 3600 seconds, which Db2 will convert to a DECIMAL(5,1) value.

Important: When a non-zero value is specified for the ACCELERATIONWAITFORDATA bind option, Db2 and the accelerator will apply other WAITFORDATA delay behaviors, restrictions, and requirements to all queries that will be accelerated from the application package. These behaviors, restrictions, and requirements can cause queries that were formerly accelerated successfully to no longer be accelerated or to fail. See "SET CURRENT QUERY ACCELERATION WAITFORDATA" on page 2136 for more information about WAITFORDATA behaviors, restrictions, and requirements.

ACCELERATOR
Specifies an accelerator server that, if enabled and available, Db2 will consider as the preferred accelerator for eligible SQL queries before sending the queries to other accelerator servers. If the specified accelerator server is not enabled or available, Db2 will send the queries to other available accelerator servers.

VALIDATE RUN or VALIDATE BIND
Specifies whether to recheck, at run time, errors of the type "OBJECT NOT FOUND" and "NOT AUTHORIZED" that are found during bind or rebind. The option has no effect if all objects and needed privileges exist.

VALIDATE RUN
Specifies that if needed objects or privileges do not exist when the CREATE statement is processed, warning messages are returned, but the CREATE statement succeeds. The Db2 subsystem rechecks for the objects and privileges at run time for those SQL statements that failed the checks during processing of the CREATE statement. The authorization checks the use of the authorization ID of the owner of the routine.

VALIDATE RUN is the default.

VALIDATE BIND
Specifies that if needed objects or privileges do not exist at the time the CREATE statement is processed, an error is issued and the CREATE statement fails.

ROUNDING
Specifies the rounding mode for manipulation of DECFLOAT data. The default value is taken from the DEFAULT DECIMAL FLOATING POINT ROUNDING MODE in DECP.

DEC_ROUND_CEILING
Specifies numbers are rounded towards positive infinity.

DEC_ROUND_DOWN
Specifies numbers are rounded towards 0 (truncation).

DEC_ROUND_FLOOR
Specifies numbers are rounded towards negative infinity.

DEC_ROUND_HALF_DOWN
Specifies numbers are rounded to nearest; if equidistant, round down.

DEC_ROUND_HALF_EVEN
Specifies numbers are rounded to nearest; if equidistant, round so that the final digit is even.

DEC_ROUND_HALF_UP
Specifies numbers are rounded to nearest; if equidistant, round up.

DEC_ROUND_UP
Specifies numbers are rounded away from 0.

DATE FORMAT ISO, EUR, USA, JIS, or LOCAL
Specifies the date format for result values that are string representations of date or time values. For more information, see "String representations of datetime values" on page 119.

The default format is specified in the DATE FORMAT field of installation panel DSNTIP4 of the system where the routine is defined. You cannot use the LOCAL option unless you have a date exit routine.

DECIMAL(15), DECIMAL(31), DECIMAL(15,*s*), or DECIMAL(31,*s*)
Specifies the maximum precision that is to be used for decimal arithmetic operations. For more information see "Arithmetic with two decimal operands" on page 245. The default format is specified in the DECIMAL ARITHMETIC field of installation panel DSNTIPF of the system where the routine is defined. If the form *pp*.*s* is specified, *s* must be a number between 1 and 9. *s* represents the minimum scale that is to be used for division.

FOR UPDATE CLAUSE OPTIONAL or FOR UPDATE CLAUSE REQUIRED
Specifies whether the FOR UPDATE clause is required for a DECLARE CURSOR statement if the cursor is to be used to perform positioned updates.

FOR UPDATE CLAUSE REQUIRED
Specifies that a FOR UPDATE clause must be specified as part of the cursor definition if the cursor will be used to make positioned updates.

FOR UPDATE CLAUSE REQUIRED is the default.

FOR UPDATE CLAUSE OPTIONAL
Specifies that the FOR UPDATE clause does not need to be specified in order for a cursor to be used for positioned updates. The routine body can include positioned UPDATE statements that update columns that the user is authorized to update.

The FOR UPDATE clause with no column list applies to static or dynamic SQL statements. Even if you do not use this clause, you can specify FOR UPDATE OF with a column list to restrict updates to only the columns that are identified in the FOR UPDATE clause and to specify the acquisition of update locks.

TIME FORMAT ISO, EUR, USA, JIS, or LOCAL
Specifies the time format for result values that are string representations of date or time values. For more information, see "String representations of datetime values" on page 119.

The default format is specified in the TIME FORMAT field of installation panel DSNTIP4 of the system where the routine is defined. You cannot use the LOCAL option unless you have a date exit routine.

BUSINESS_TIME SENSITIVE
Determines whether references to application-period temporal tables in both static and dynamic SQL statements are affected by the value of the CURRENT TEMPORAL BUSINESS_TIME special register.

YES
References to application-period temporal tables are affected by the value of the CURRENT TEMPORAL BUSINESS_TIME special register. YES is the default value.

NO
References to application-period temporal tables are not affected by the value of the CURRENT TEMPORAL BUSINESS_TIME special register.

For more information, see ."CURRENT TEMPORAL BUSINESS_TIME" on page 205

SYSTEM_TIME SENSITIVE
Determines whether references to system-period temporal tables in both static and dynamic SQL statements are affected by the value of the CURRENT TEMPORAL SYSTEM_TIME special register.

YES
References to system-period temporal tables are affected by the value of the CURRENT TEMPORAL SYSTEM_TIME special register. YES is the default value.

NO
References to system-period temporal tables are not affected by the value of the CURRENT TEMPORAL SYSTEM_TIME special register.

For more information, see "CURRENT TEMPORAL SYSTEM_TIME" on page 206.

ARCHIVE SENSITIVE
Determines whether references to archive-enabled tables in SQL statements are affected by the value of the SYSIBMADM.GET_ARCHIVE built-in global variable.

YES
References to archive-enabled tables are affected by the value of the SYSIBMADM.GET_ARCHIVE built-in global variable. YES is the default value.

NO
References to archive-enabled tables are not affected by the value of the SYSIBMADM.GET_ARCHIVE built-in global variable.

For related information, see "GET_ARCHIVE" on page 327

APPLCOMPAT *applcompat-level*
Specifies the application compatibility level behavior for static SQL statements in the procedure body. If this option is not specified, the behavior is determined by the APPLCOMPAT subsystem parameter.

The following *applcompat-level* values can be specified:

V*vv*R*r*M*mmm*
Compatibility with the behavior of the identified Db2 function level. For example, V13R1M503 specifies compatibility with the highest available Db2 13 function level. The equivalent function level or higher must be activated. For a list of supported Db2 13 function levels, see Db2 13 function levels (Db2 for z/OS What's New?). Db2 13 also supports values from Db2 12. See Db2 12 function levels.

V12R1
Compatibility with the behavior of Db2 12 function level 500. This value has the same result as specifying V12R1M500.

V11R1
Compatibility with the behavior of Db2 11 new-function mode.

V10R1
Compatibility with the behavior of DB2 10 new-function mode.

CONCENTRATE STATEMENTS OFF or CONCENTRATE STATEMENTS WITH LITERALS
Specifies whether each dynamic SQL statement in the routine that specifies literal constants will be cached as a separate unique statement entry in the dynamic statement cache, instead of sharing an existing statement in the cache. Dynamic SQL statements are eligible to share an existing statement in the cache if the new statement meets all of the conditions for sharing a cached version of the same dynamic statement, except that the new statement specifies one or more literal constants that are different than the cached statement.

CONCENTRATE STATEMENTS OFF

Specifies that each dynamic SQL statement that specifies literal constants will be cached as a unique statement entry if it specifies one or more constants that are different than the cached version of the same dynamic statement. CONCENTRATE STATEMENTS OFF is the default dynamic statement caching behavior.

CONCENTRATE STATEMENTS WITH LITERALS

Specifies that each dynamic SQL statement that specifies literal constants will share a cached version of the same dynamic statement that is also prepared using the CONCENTRATE STATEMENTS WITH LITERALS option, if the new dynamic statement meets all of the conditions for sharing the cached statement, and the constants that are specified can be reused in place of the constants in the cached statement.

SQL-routine-body
Specifies the statements that define the body of the SQL procedure. For information on the SQL control statements that are supported in native SQL procedures, see Chapter 8, "SQL procedural language (SQL PL)," on page 2189. If an *SQL-procedure-statement* is the only statement in the procedure body, the statement must not end with a semicolon.

WRAPPED *obfuscated-statement-text*
Specifies the encoded definition of the function. A CREATE PROCEDURE statement can be encoded using the WRAP scalar function.

WRAPPED must not be specified on a static CREATE statement, or a CREATE statement that adds or replaces a version of an existing procedure.

Notes

Considerations for all types of procedures
For considerations that apply to all types of procedures, see “CREATE PROCEDURE” on page 1570.

Error handling in SQL procedures:
You should consider the possible exceptions that can occur for each SQL statement in the body of a procedure. Any exception SQLSTATE that is not handled within the procedure using a handler within a compound statement results in the exception SQLSTATE being returned to the caller of the procedure.

Versions of a procedure:
The CREATE PROCEDURE statement for an SQL procedure defines the initial version of the procedure. You can define an additional version using the ADD VERSION clause of the ALTER PROCEDURE statement, or the CREATE PROCEDURE statement with the OR REPLACE clause and the VERSION clause when the procedure already exists. You can replace a version using the REPLACE VERSION clause of the ALTER PROCEDURE, or the CREATE PROCEDURE statement with the OR REPLACE clause and the VERSION clause when the procedure version already exists.

The data types, CCSID specifications and character data attributes (FOR BIT/SBCS/MIXED DATA) of the parameters must be the same as the attributes of the corresponding parameters for the currently active version of the procedure, unless the OR REPLACE is specified and the VERSION keyword is not specified.

Important: Do not create additional versions of procedures that are supplied with Db2 by specifying the VERSION keyword. Only versions that are supplied with Db2 are supported. Additional versions of such routines cause the installation and configuration of the supplied routines to fail.

Considerations for an existing procedure that is defined using a TABLE LIKE name AS LOCATOR clause:
If an existing native SQL procedure is defined with a table parameter (the TABLE LIKE *name* AS LOCATOR clause was specified in the original CREATE PROCEDURE statement to indicate that one of the parameters is a transition table), the procedure cannot be changed with a CREATE PROCEDURE statement to add or replace a version of the procedure. In this case, the procedure must be dropped and re-created.

Characteristics of the package that is generated for a procedure:
The package that is associated with the first version of a procedure is named as follows:

- *location* is set to the value of the CURRENT SERVER special register
- *collection-id* (schema) for the package is the same as the schema qualifier of the procedure.
- *package-id* is the same as the specific name of the procedure
- *version-id* is the same as the version identifier for the initial version of the procedure.

If you want to change the *collection-id* for the name of the package, you need to make a copy of the package.

The package is generated using the bind options that correspond to the implicitly or explicitly specified procedure options. See "Implicit rebinding and regeneration that occurs because of an ALTER PROCEDURE statement" in Chapter 8, “SQL procedural language (SQL PL),” on page 2189 for more information. In addition to the corresponding bind options, the package is generated using the following bind options:

- DBPROTOCOL(DRDA)
- FLAG(1)
- SQLERROR(NOPACKAGE)

- ENABLE(*)

See Table 175 on page 1211 for additional information about the correspondence of procedure options to bind options.

Application compatibility level considerations for procedure objects

The application compatibility level controls the adoption and use of new capabilities and enhancements. When an object is created or altered, two separate application compatibility levels are used: one to process the definition of the object, and the other for processing the SQL statements in the object body:

Object definition	The CURRENT APPLICATION COMPATIBILITY special register value is used to process the object definition, except for statements in the object body This application compatibility level is stored in the SYSENVIRONMENT.APPLCOMPAT column. You can use the environment ID value in the catalog definition of the object to locate the SYSENVIRONMENT row with the matching ENVID value. This application compatibility level can be changed when the object is regenerated.
Statements in the object body	The application compatibility level that is implicitly or explicitly specified with the APPLCOMPAT option of the CREATE or ALTER statement is used to process statements in the object body. This application compatibility level is stored in the SYSPACKAGE.APPLCOMPAT column for the package associated with the object definition.

Considerations for SQL processor programs:

SQL processor programs, such as SPUFI, the command line processor, and DSNTEP2, might not correctly parse SQL statements that end with semicolons in the routine body of a CREATE PROCEDURE statement. These processor programs accept multiple SQL statements as input, with each statement separated with a terminator character. Processor programs that use a semicolon as the SQL statement terminator can truncate a CREATE PROCEDURE statement with embedded semicolons and pass only a portion of it to Db2. Therefore, you might need to change the SQL terminator character for these processor programs. For more information, see Setting the SQL terminator character in a SPUFI input data set (Db2 Application programming and SQL), and SQLTERM in DSNTEP2 and DSNTEP4 sample programs (Db2 Application programming and SQL).

Identifier resolution:

See Chapter 8, “SQL procedural language (SQL PL),” on page 2189 for information on how names are resolved to columns, variables, or SQL parameters within an SQL routine.

If duplicate names are used for columns, variables, and parameters, qualify the duplicate names by using the table designator for columns, the routine name for parameters, the label name for SQL variables, and the schema name for global variables.

Lines within the SQL procedure definition:

When an SQL procedure is created, information is retained on lines in the CREATE statement. Lines are determined by the presence of the new line control character.

In an SQL procedure, a *new line control character* is a special character that is used for a new line. The new line control characters for an SQL procedure include:

- Line feed
- New line
- Carriage return
- Carriage return, followed by a line feed

- Carriage return, followed by a new line

For more information about control characters, see Tokens.

Stored procedures with a parameter that is defined as an array type:
A procedure that is defined with a parameter that is an array type, other than an array global variable, can be invoked only from within an SQL PL context, or from a Java application program that uses IBM Data Server Driver for JDBC and SQLJ type 4 connectivity, unless the corresponding argument in the CALL statement is an array global variable. If the corresponding argument in the CALL statement is an array global variable, the procedure can be invoked outside an SQL PL context.

Obfuscated statements:
A CREATE PROCEDURE statement for a native SQL procedure can be executed in obfuscated form. In an obfuscated statement, only the procedure name, parameters, and the WRAPPED keyword are readable. The rest of the statement is encoded in such a way that it is not readable but can be decoded by a database server that supports obfuscated statements. The WRAP scalar function produces obfuscated statements. Any debug options that are specified when the function is created from an obfuscated statement are ignored.

Compatibilities:
For compatibility with previous versions of Db2, the following clauses can be specified, but they will be ignored and a warning will be issued:

- STAY RESIDENT
- PROGRAM TYPE
- RUN OPTIONS
- NO DBINFO
- COLLID or NOCOLLID
- SECURITY
- PARAMETER STYLE GENERAL WITH NULLS
- STOP AFTER SYSTEM DEFAULT FAILURES
- STOP AFTER *nn* FAILURES
- CONTINUE AFTER FAILURES
- PARAMETER VARCHAR

If the FENCED or EXTERNAL clause is specified, an external SQL procedure will be generated. See "CREATE PROCEDURE (SQL - external) (deprecated)" on page 1589 for more information.

If WLM ENVIRONMENT is specified without the FOR DEBUG MODE keywords, and error is returned. If WLM ENVIRONMENT is specified for a native SQL procedure, WLM ENVIRONMENT FOR DEBUG MODE must be specified.

Alternative syntax and synonyms:
To provide compatibility with previous releases of Db2 or other products in the Db2 family, Db2 supports the following alternative syntax:

- RESULT SET and RESULT SETS as synonyms for DYNAMIC RESULT SETS
- VARIANT as a synonym for NOT DETERMINISTIC
- NOT VARIANT as a synonym for DETERMINISTIC
- NULL CALL as a synonym for CALLED ON NULL

Examples

Example 1

Create the definition for an SQL procedure. The procedure accepts an employee number and a multiplier for a pay raise as input. The following tasks are performed in the procedure body:

- Calculate the employee's new salary.
- Update the employee table with the new salary value.

```
CREATE PROCEDURE UPDATE_SALARY_1
 (IN EMPLOYEE_NUMBER CHAR(10),
 IN RATE DECIMAL(6,2))
 LANGUAGE SQL
 MODIFIES SQL DATA
  UPDATE EMP
  SET SALARY = SALARY * RATE
  WHERE EMPNO = EMPLOYEE_NUMBER
```

Example 2

Create the definition for the SQL procedure described in example 1, but specify that the procedure has these characteristics:

- The same input always produces the same output.
- SQL work is committed on return to the caller.

```
CREATE PROCEDURE UPDATE_SALARY_1
 (IN EMPLOYEE_NUMBER CHAR(10),
 IN RATE DECIMAL(6,2))
 LANGUAGE SQL
 MODIFIES SQL DATA
 DETERMINISTIC
 COMMIT ON RETURN YES
   UPDATE EMP
   SET SALARY = SALARY * RATE
   WHERE EMPNO = EMPLOYEE_NUMBER
```

Example 3:

Create the definition for an SQL procedure that uses arrays as IN and OUT parameters. The procedure is named GETWEEKENDS. It accepts an array of DATE values as input, and returns an array that contains only the dates that fall on a Saturday or Sunday. For example, if the input dates are a Saturday, a Friday, and a Sunday, the procedure returns only the dates that fall on Saturday and Sunday.

Suppose that the following user-defined array type has been defined:

```
CREATE TYPE DATEARRAY AS DATE ARRAY[100];
```

After the array type is created, any references to it need to specify the fully qualified user-defined array type name. Otherwise, the schema for the type needs to be in the CURRENT PATH.

Suppose that the SQL procedure is defined like this:

```
CREATE PROCEDURE GETWEEKENDS(IN MYDATES DATEARRAY, OUT WEEKENDS DATEARRAY)
 BEGIN
  -- ARRAY INDEX VARIABLES
  DECLARE DATEINDEX, WEEKENDINDEX INT DEFAULT 1;
  -- VARIABLE TO STORE THE ARRAY LENGTH OF MYDATES,
  -- INITIALIZED USING THE CARDINALITY FUNCTION.
  DECLARE DATESCOUNT INT;
  SET DATESCOUNT = CARDINALITY(MYDATES);
  -- FOR EACH DATE IN MYDATES, IF THE DATE IS A SUNDAY OR SATURDAY,
  -- ADD IT TO THE OUTPUT ARRAY NAMED "WEEKENDS"
  WHILE DATEINDEX <= DATESCOUNT DO
   IF DAYOFWEEK(MYDATES[DATEINDEX]) IN (1, 7) THEN
    SET WEEKENDS[WEEKENDINDEX] = MYDATES[DATEINDEX];
    SET WEEKENDINDEX = WEEKENDINDEX + 1;
   END IF;
   SET DATEINDEX = DATEINDEX + 1;
  END WHILE;
 END
```

Also suppose that input array MYDATES contains the following content:

```
['2012-04-28', '2012-02-10','2012-03-18']
```

After the procedure returns, output array WEEKENDS contains the following content:

```
['2012-04-28', '2012-03-18']
```

Example 4

Create the definition for an SQL procedure that uses arrays as OUT parameters. The procedure is named GET_PHONES. It returns an array that contains phone numbers for employee 1775. If more than five phone numbers exist for the employee, an error is returned because the array is defined for only five elements.

Suppose that the following user-defined array type and table have been defined:

```
CREATE TYPE PHONELIST AS DECIMAL(10, 0) ARRAY[5];
CREATE TABLE EMP_PHONES(ID INTEGER, PHONENUMBER DECIMAL(10,0));
```

The SQL procedure is defined like this:

```
CREATE PROCEDURE GET_PHONES(OUT EPHONES PHONELIST)
 BEGIN
  SELECT ARRAY_AGG(PHONENUMBER)
   INTO EPHONES
   FROM EMP_PHONES
   WHERE ID = 1775;
 END
```

For more examples of SQL procedures, see Chapter 8, “SQL procedural language (SQL PL),” on page 2189.

Related concepts

Procedures (Introduction to Db2 for z/OS)

Naming conventions

The rules for forming a name depend on the type of the object designated by the name.

Related tasks

Creating native SQL procedures (Db2 Application programming and SQL)

Migrating an external SQL procedure to a native SQL procedure (Db2 Application programming and SQL)

CREATE ROLE

The CREATE ROLE statement creates a role at the current server.

Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared only if DYNAMICRULES RUN behavior is in effect. For more information, see “Authorization IDs and dynamic SQL” on page 94.

Authorization

The privilege set that is defined below must include at least one of the following authorities:

- SYSADM authority
- SYSCTRL authority
- SECADM

Privilege set: If the statement is embedded in an application program, the privilege set is the set of privileges that are held by the owner of the plan or package.

If the statement is dynamically prepared, the privilege set is the set of privileges that are held by the SQL authorization ID of the process or by the role that is associated with the primary authorization ID, if the statement is run in a trusted context and the ROLE AS OBJECT OWNER clause is specified.