DB2 for OS/390 Version 5

Preview of SQL Procedures

October 5, 1999

This document contains preliminary information that is subject to change.

We welcome your comments. Please send them to:

sqlproc@us.ibm.com

DB2 for OS/390 Version 5

Preview of SQL Procedures

October 5, 1999

Note

Before using this information and the product it supports, read the information in “Notices” on page 55.

First Edition (September 1999)

This edition applies to Version 5 of IBM DATABASE 2 Server for OS/390 (DB2 for OS/390), 5655-DB2, and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

Chapter 1. Introduction

This document describes DB2 for OS/390 support for the SQL procedure language. The SQL procedure language is an implementation of SQL/PSM, which is part 4 of the ANSI/ISO SQL full level standard of 1992.

The contents of this document will be incorporated into future editions of the following DB2 for OS/390 documentation:
- *Application Programming and SQL Guide*
- *Installation Guide*
- *SQL Reference*

Who should read this book

This book is for DB2 application developers who are familiar with Structured Query Language (SQL) and stored procedures and want to learn about writing stored procedures in the SQL procedure language.

How this book is organized

This book is organized as follows:
- "Chapter 1. Introduction" describes this book and gives general information that you need to know when you read it.
- "Chapter 2. Installing the SQL procedures code" on page 5 tells you how to install the SQL procedures code on your DB2 for OS/390 system.
- "Chapter 3. Writing an SQL procedure" on page 7 describes SQL procedures and shows examples of how to write them.
- "Chapter 4. Statements for SQL procedures" on page 15 describes the syntax of the CREATE PROCEDURE statement for an SQL procedure and the elements of the SQL procedure language.
- "Chapter 5. Preparing and running an SQL procedure" on page 39 describes how to prepare and run an SQL procedure.
- The appendixes contain supplemental information on:
  - DB2 tables that are used by SQL procedures
  - SQL statements that are valid in SQL procedures
  - SQL reserved words
  - Messages that DB2 can generate during SQL procedure program preparation

Located at the end of this book are:
- Legal notices
- An index

Other books you might need

DB2 for OS/390 is one of several relational database management systems developed by IBM®. Each of these systems understands its own variety of SQL. This book discusses only the variety used by DB2 for OS/390. Other IBM books describe the other varieties. For a list of these books, see the bibliography at the end of this book.

If DB2 is the only product you plan to use, you should have available *SQL Reference*, which is an encyclopedic reference to the syntax and semantics of every statement in DB2 SQL. For SQL fundamentals and concepts, see Chapter 2 of *SQL Reference*.

If you intend to develop applications that adhere to the definition of IBM SQL, see *IBM SQL Reference* for more information.

When preparing programs for execution, you will want to refer to the list of options for BIND and REBIND PLAN and PACKAGE, in *Command Reference*.

Product terminology and citations

In this book, DB2 Server for OS/390 is referred to as "DB2 for OS/390." In cases where the context makes the meaning clear, DB2 for OS/390 is referred to as "DB2." When this book refers to other books in this library, a short title is used. (For example, "See *SQL Reference*" is a citation to IBM DATABASE 2 Server for OS/390 *SQL Reference*.)

The following terms are used as indicated:

DB2	Represents either the DB2 licensed program or a particular DB2 subsystem.
MVS	Represents the MVS/Enterprise Systems Architecture (MVS/ESA) element of OS/390.

How to read the syntax diagrams

The following rules apply to the syntax diagrams that are used in this book:

- Read the syntax diagrams from left to right, from top to bottom, following the path of the line.

  The ►►── symbol indicates the beginning of a statement.

  The ──► symbol indicates that the statement syntax is continued on the next line.

  The ►── symbol indicates that a statement is continued from the previous line.

  The ──►◄ symbol indicates the end of a statement.

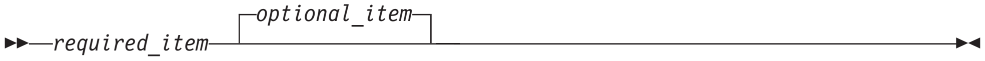
  Diagrams of syntactical units other than complete statements start with the ►── symbol and end with the ──► symbol.
- Required items appear on the horizontal line (the main path).

```
►►──required_item──────────────────────────────────────────────►◄
```

- Optional items appear below the main path.

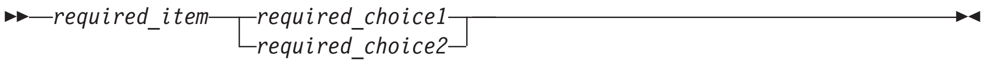
```
►►──required_item──┬───────────────┬───────────────────────────►◄
                   └─optional_item─┘
```

  If an optional item appears above the main path, that item has no effect on the execution of the statement and is used only for readability.

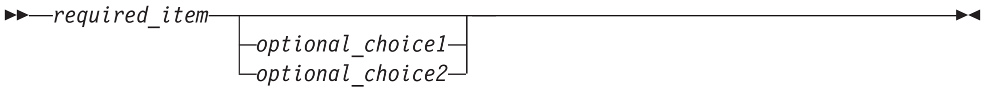


- If you can choose from two or more items, they appear vertically, in a stack.

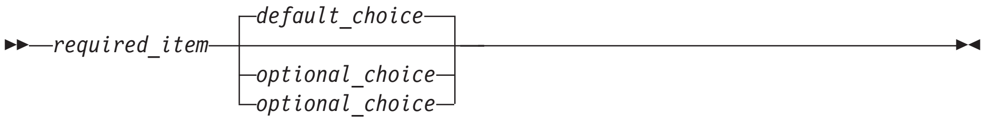
  If you *must* choose one of the items, one item of the stack appears on the main path.



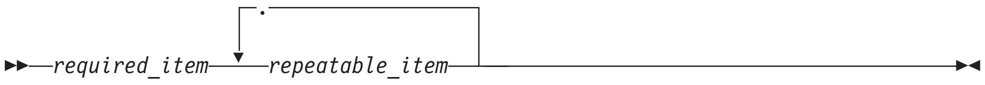
  If choosing one of the items is optional, the entire stack appears below the main path.



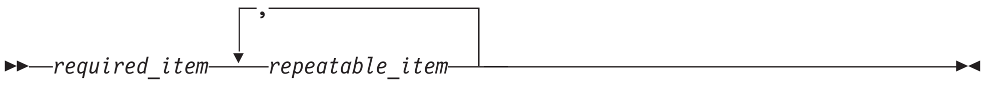
  If one of the items is the default, it appears above the main path and the remaining choices are shown below.



- An arrow returning to the left, above the main line, indicates an item that can be repeated.



  If the repeat arrow contains a comma, you must separate repeated items with a comma.



  A repeat arrow above a stack indicates that you can repeat the items in the stack.
- Keywords appear in uppercase (for example, `FROM`). They must be spelled exactly as shown. Variables appear in all lowercase letters (for example, *column-name*). They represent user-supplied names or values.
- If punctuation marks, parentheses, arithmetic operators, or other such symbols are shown, you must enter them as part of the syntax.

Chapter 2. Installing the SQL procedures code

The SQL procedures code for DB2 Version 5 is shipped as zip file sqlproc1.zip, which you can download from

`http://www.ibm.com/software/db2os390/sqlproc`

After you download the zip file to your PC and unzip it, follow the instructions in sqlproc1.readme to install SQL procedures support.

Chapter 3. Writing an SQL procedure

An SQL procedure is a stored procedure in which the source code for the procedure is in an SQL CREATE PROCEDURE statement. The part of the CREATE PROCEDURE statement that contains the code is called the *procedure body*.

Creating an SQL procedure involves writing the source statements for the SQL procedure, creating the executable form of the SQL procedure, and defining the SQL procedure to DB2. There are two ways to create an SQL procedure:

- Use the IBM DB2 Stored Procedure Builder product to specify the source statements for the SQL procedure, define the SQL procedure to DB2, and prepare the SQL procedure for execution.
- Write a CREATE PROCEDURE statement for the SQL procedure. Then use one of the methods in "Chapter 5. Preparing and running an SQL procedure" on page 39 to define the SQL procedure to DB2 and create an executable procedure.

This chapter discusses how to write a CREATE PROCEDURE statement for an SQL procedure. The following topics are included:

- "Comparison of an SQL procedure and an external procedure"
- "Statements that you can include in a procedure body" on page 8
- "Terminating statements in an SQL procedure" on page 10
- "Handling errors in an SQL procedure" on page 10
- "Examples of SQL procedures" on page 12

For information on the syntax of the CREATE PROCEDURE statement and the procedure body, see "Chapter 4. Statements for SQL procedures" on page 15.

Comparison of an SQL procedure and an external procedure

Like an external stored procedure, an SQL procedure consists of a stored procedure definition and the code for the stored procedure program.

An external stored procedure definition and an SQL procedure definition specify the following common information:

- The procedure name.
- Input and output parameter attributes.
- The language in which the procedure is written. For an SQL procedure, the language is SQL.
- Information that will be used when the procedure is called, such as run-time options, length of time that the procedure can run, and whether the procedure returns result sets.

An external stored procedure and an SQL procedure differ in the way that they specify the code for the stored procedure. An external stored procedure definition specifies the name of the stored procedure program. An SQL procedure definition contains the source code for the stored procedure.

For an external stored procedure, you define the stored procedure to DB2 by inserting a row into SYSIBM.SYSPROCEDURES. For an SQL procedure, you define the stored procedure to DB2 by preprocessing a CREATE PROCEDURE statement, then inserting a row into SYSIBM.SYSPROCEDURES. See “Chapter 5. Preparing and running an SQL procedure” on page 39 for more information on defining an SQL procedure to DB2.

Figure 1 shows a definition for an external stored procedure that is written in COBOL. The stored procedure program, which updates employee salaries, is called UPDSAL.

Figure 2 shows a definition for an equivalent SQL procedure.

```
INSERT INTO  SYSIBM.SYSPROCEDURES
 (PROCEDURE,
  PARMLIST,
  LANGUAGE,
  LOADMOD,
  COLLID, IBMREQD, RUNOPTS)
  VALUES('UPDATESALARY1',                               1
  'EMPNUMBR CHAR(10) IN, RATE DECIMAL(6,2) IN',         2
  'COBOL',                                              3
  'UPDSAL',                                             4
  ' ', 'N', ' ');
```

Figure 1. Example of an external stored procedure definition

Notes to Figure 1:

1 The stored procedure name is UPDATESALARY1.
2 The two parameters have data types of CHAR(10) and DECIMAL(6,2). Both are input parameters.
3 LANGUAGE COBOL indicates that this is an external procedure, so the code for the stored procedure is in a separate, COBOL program.
4 The name of the load module that contains the executable stored procedure program is UPDSAL.

```
CREATE PROCEDURE UPDATESALARY1         1
 (IN EMPNUMBR CHAR(10),                2
 IN RATE DECIMAL(6,2))
 LANGUAGE SQL                          3
  UPDATE EMP                           4
   SET SALARY = SALARY * RATE
   WHERE EMPNO = EMPNUMBR
```

Figure 2. Example of an SQL procedure definition

Notes to Figure 2:

1 The stored procedure name is UPDATESALARY1.
2 The two parameters have data types of CHAR(10) and DECIMAL(6,2). Both are input parameters.
3 LANGUAGE SQL indicates that this is an SQL procedure, so a procedure body follows the other parameters.
4 The procedure body consists of a single SQL UPDATE statement, which updates rows in the employee table.

Statements that you can include in a procedure body

A procedure body consists of a single simple or compound statement. The types of statements that you can include in a procedure body are:

Assignment statement
Assigns a value to an output parameter or to an SQL variable, which is a variable that is defined and used only within a procedure body.

CASE statement
Selects an execution path based on the evaluation of one or more conditions. This statement is similar to the CASE expression, which is described in Chapter 3 of *SQL Reference*.

IF statement
Selects an execution path based on the evaluation of a condition.

LEAVE statement
Transfers program control out of a loop or a block of code.

LOOP statement
Executes a statement or group of statements multiple times.

REPEAT statement
Executes a statement or group of statements until a search condition is true.

WHILE statement
Repeats the execution of a statement or group of statements while a specified condition is true.

Compound statement
Can contain one or more of any of the other types of statements in this list. In addition, a compound statement can contain SQL variable declarations, condition handlers, or cursor declarations.

The order of statements in a compound statement must be:

1. SQL variable and condition declarations
2. Cursor declarations
3. Handler declarations
4. Procedure body statements (CASE, IF, LOOP, REPEAT, WHILE, SQL)

SQL statement
A subset of the SQL statements that are described in Chapter 6 of *SQL Reference*. Certain SQL statements are valid in a compound statement, but not valid if the SQL statement is the only statement in the procedure body. Table 4 on page 49 lists the SQL statements that are valid in an SQL procedure.

See “The SQL procedure body” on page 23 for detailed descriptions and syntax of each of these statements.

Declaring variables in an SQL procedure

To store data that you use only within an SQL procedure, you can declare *SQL variables.* SQL variables can have the same data types and lengths as SQL procedure parameters. For a discussion of data types and lengths, see “CREATE PROCEDURE (SQL procedure)” on page 16.

The general form of an SQL variable declaration is:

```
DECLARE SQL-variable-name data-type;
```

SQL variables have these restrictions:

- Because DB2 folds all SQL variables to uppercase, you cannot declare two SQL variables that are the same except for case. For example, you cannot declare two SQL variables named varx and VARX.
- If you refer to an SQL procedure parameter in the procedure body, you cannot declare an SQL variable with a name that is the same as that parameter name.
- Do not use an SQL reserved word as an SQL variable name.
- When you use an SQL variable in an SQL statement, do not precede the variable with a colon.

You can perform any operations on SQL variables that you can perform on host variables in SQL statements.

If you specify a label for a compound statement, you can qualify the SQL variable name with that label name. Qualifying SQL variable names and other object names is a good way to avoid ambiguity. Use the following guidelines to determine when to qualify variable names:

- When you use an SQL procedure parameter in the procedure body, qualify the parameter name with the procedure name.
- Qualify SQL variable names with the label of the compound statement in which the SQL variables appear.
- Qualify column names with the associated table or view names.

Parameter style for an SQL procedure

DB2 supports only the SIMPLE WITH NULLS linkage convention for SQL procedures. This means that when you call an SQL procedure, you must include an indicator variable with each parameter in the CALL statement. See Section 6 of *Application Programming and SQL Guide* See for more information on stored procedure linkage conventions.

Terminating statements in an SQL procedure

The way that you terminate a statement in an SQL procedure depends on the use of the statement in that procedure:

- A procedure body has no terminating character. Therefore, if an SQL procedure statement is the outermost of a set of nested statements, or if the statement is the only statement in the procedure body, that statement does not have a terminating character.
- If a statement is nested within other statements in the procedure body, that statement ends with a semicolon.

Handling errors in an SQL procedure

If an SQL error occurs when an SQL procedure executes, the SQL procedure ends unless you include statements called *handlers* to tell the procedure to perform some other action. Handlers are similar to WHENEVER statements in external SQL application programs. Handlers tell the SQL procedure what to do when an SQL error or SQL warning occurs, or when no more rows are returned from a query. In addition, you can declare handlers for specific SQLSTATEs. You can refer to an SQLSTATE by its number in a handler, or you can declare a name for the SQLSTATE, then use that name in the handler.

The general form of a handler declaration is:

```
DECLARE handler-type HANDLER FOR condition SQL-procedure-statement;
```

In general, the way that a handler works is that when an error occurs that matches *condition*, *SQL-procedure-statement* executes. When *SQL-procedure-statement* completes, DB2 performs the action that is indicated by *handler-type*.

There are two types of handlers:

CONTINUE
Specifies that after *SQL-procedure-statement* completes, execution continues with the statement after the statement that caused the error.

EXIT
Specifies that after *SQL-procedure-statement* completes, execution continues at the end of the compound statement that contains the handler.

Example: CONTINUE handler: This handler sets flag at_end when no more rows satisfy a query. The handler then causes execution to continue after the statement that returned no rows.

```
DECLARE CONTINUE HANDLER FOR NOT FOUND SET at_end=1;
```

Example: EXIT handler: This handler places the string 'Table does not exist' into output parameter OUT_BUFFER when condition NO_TABLE occurs. NO_TABLE is previously declared as SQLSTATE 42704 (*name* is an undefined name). The handler then causes the SQL procedure to exit the compound statement in which the handler is declared.

```
DECLARE NO_TABLE CONDITION FOR '42704';
⋮

DECLARE EXIT HANDLER FOR NO_TABLE
 SET OUT_BUFFER='Table does not exist';
```

Referencing the SQLCODE and SQLSTATE values: When an SQL error or warning occurs in an SQL procedure, you might need to reference the SQLCODE or SQLSTATE values in your SQL procedure or pass those values to the procedure caller. Before you can reference SQLCODE or SQLSTATE values, you must declare the SQLCODE and SQLSTATE as SQL variables. The definitions are:

```
DECLARE SQLCODE INTEGER;
DECLARE SQLSTATE CHAR(5);
```

If you want to pass the SQLCODE or SQLSTATE values to the caller, your SQL procedure definition needs to include output parameters for those values. After an error occurs, and before control returns to the caller, you can assign the value of SQLCODE or SQLSTATE to the corresponding output parameter. For example, you might include assignment statements in an SQLEXCEPTION handler to assign the SQLCODE value to an output parameter:

```
CREATE PROCEDURE UPDATESALARY1
 (IN EMPNUMBR CHAR(6),
  OUT SQLCPARM INTEGER)
 LANGUAGE SQL
⋮

BEGIN:
 DECLARE SQLCODE INTEGER;
 DECLARE CONTINUE HANDLER FOR SQLEXCEPTION
  SET SQLCPARM = SQLCODE;
⋮
```

Handling truncation errors in an SQL procedure: Truncation during any of the following assignments in an SQL procedure causes the SQL procedure to end unless a CONTINUE handler is defined:

- Assignment of a value to an SQL variable or parameter
- Specification of a default value in a DECLARE statement

You can declare a general CONTINUE for SQLEXCEPTION, or you can declare the specific CONTINUE handlers for the following SQLSTATE values:

22001	For character truncation
22003	For numeric truncation

Examples of SQL procedures

This section contains examples of how to use each of the statements that can appear in an SQL procedure body.

Example: CASE statement: The following SQL procedure demonstates how to use a CASE statement. The procedure receives an employee's ID number and rating as input parameters. The CASE statement modifies the employee's salary and bonus, using a different UPDATE statement for each of the possible ratings.

```
CREATE PROCEDURE UPDATESALARY2
 (IN EMPNUMBR CHAR(6),
  IN RATING INT)
 LANGUAGE SQL

 CASE RATING
  WHEN 1 THEN
   UPDATE CORPDATA.EMPLOYEE
    SET SALARY = SALARY * 1.10, BONUS = 1000
    WHERE EMPNO = EMPNUMBR;
  WHEN 2 THEN
   UPDATE CORPDATA.EMPLOYEE
    SET SALARY = SALARY * 1.05, BONUS = 500
    WHERE EMPNO = EMPNUMBR;
  ELSE
   UPDATE CORPDATA.EMPLOYEE
    SET SALARY = SALARY * 1.03, BONUS = 0
    WHERE EMPNO = EMPNUMBR;
 END CASE
```

Example: Compound statement with nested IF and WHILE statements: The following example shows a compound statement that includes an IF statement, a WHILE statement, and assignment statements. The example also shows how to declare SQL variables, cursors, and handlers for classes of error codes.

The procedure receives a department number as an input parameter. A WHILE statement in the procedure body fetches the salary and bonus for each employee in the department, and uses an SQL variable to calculate a running total of employee salaries for the department. An IF statement within the WHILE statement tests for positive bonuses and increments an SQL variable that counts the number of bonuses in the department. When all employee records in the department have been processed, the FETCH statement that retrieves employee records receives SQLCODE 100. A NOT FOUND condition handler makes the search condition for the WHILE statement false, so execution of the WHILE statement ends. Assignment statements then assign the total employee salaries and the number of bonuses for the department to the output parameters for the stored procedure.

If any SQL statement in the procedure body receives a negative SQLCODE, the SQLEXCEPTION handler receives control. This handler sets output parameter DEPTSALARY to NULL and ends execution of the SQL procedure. When this handler is invoked, the SQLCODE and SQLSTATE are set to 0.

```
CREATE PROCEDURE RETURNDEPTSALARY
 (IN DEPTNUMBER CHAR(3),
  OUT DEPTSALARY DECIMAL(15,2),
  OUT DEPTBONUSCNT INT)
 LANGUAGE SQL

 P1: BEGIN
     DECLARE EMPLOYEE_SALARY DECIMAL(9,2);
     DECLARE EMPLOYEE_BONUS DECIMAL(9,2);
     DECLARE TOTAL_SALARY DECIMAL(15,2) DEFAULT 0;
     DECLARE BONUS_CNT INT DEFAULT 0;
     DECLARE END_TABLE INT DEFAULT 0;
     DECLARE C1 CURSOR FOR
      SELECT SALARY, BONUS FROM CORPDATA.EMPLOYEE
       WHERE WORKDEPT = DEPTNUMBER;
     DECLARE CONTINUE HANDLER FOR NOT FOUND
      SET END_TABLE = 1;
     DECLARE EXIT HANDLER FOR SQLEXCEPTION
      SET DEPTSALARY = NULL;
     OPEN C1;
     FETCH C1 INTO EMPLOYEE_SALARY, EMPLOYEE_BONUS;
     WHILE END_TABLE = 0 DO
      SET TOTAL_SALARY = TOTAL_SALARY + EMPLOYEE_SALARY + EMPLOYEE_BONUS;
      IF EMPLOYEE_BONUS > 0 THEN
       SET BONUS_CNT = BONUS_CNT + 1;
      END IF;
      FETCH C1 INTO EMPLOYEE_SALARY, EMPLOYEE_BONUS;
     END WHILE;
     CLOSE C1;
     SET DEPTSALARY = TOTAL_SALARY;
     SET DEPTBONUSCNT = BONUS_CNT;
 END P1
```

Example: Compound statement with dynamic SQL statements: The following example shows a compound statement that includes dynamic SQL statements.

The procedure receives a department number (P_DEPT) as an input parameter. In the compound statement, three statement strings are built, prepared, and executed. The first statement string executes a DROP statement to ensure that the table to be created does not already exist. This table is named DEPT_*deptno*_T, where *deptno* is the value of input parameter P_DEPT. The next statement string executes a CREATE statement to create DEPT_*deptno*_T. The third statement string inserts rows for employees in department *deptno* into DEPT_*deptno*_T. Just as statement strings that are prepared in host language programs cannot contain host variables, statement strings in SQL procedures cannot contain SQL variables or stored procedure parameters. Therefore, the third statement string contains a parameter marker that represents P_DEPT. When the prepared statement is executed, parameter P_DEPT is substituted for the parameter marker.

```
CREATE PROCEDURE CREATEDEPTTABLE (IN P_DEPT CHAR(3))
 LANGUAGE SQL
 BEGIN
  DECLARE STMT CHAR(1000);
  DECLARE MESSAGE CHAR(20);
  DECLARE TABLE_NAME CHAR(30);
  DECLARE CONTINUE HANDLER FOR SQLEXCEPTION
   SET MESSAGE = 'ok';
  SET TABLE_NAME = 'DEPT_'||P_DEPT||'_T';
  SET STMT = 'DROP TABLE '||TABLE_NAME;
 PREPARE S1 FROM STMT;
 EXECUTE S1;
 SET STMT = 'CREATE TABLE '||TABLE_NAME||
  '( EMPNO CHAR(6) NOT NULL, '||
  'FIRSTNME VARCHAR(6) NOT NULL, '||
  'MIDINIT CHAR(1) NOT NULL, '||
  'LASTNAME CHAR(15) NOT NULL, '||
  'SALARY DECIMAL(9,2))';
 PREPARE S2 FROM STMT;
 EXECUTE S2;
 SET STMT = 'INSERT INTO '||TABLE_NAME ||
  'SELECT EMPNO, FIRSTNME, MIDINIT, LASTNAME, SALARY '||
  'FROM EMPLOYEE '||
  'WHERE WORKDEPT = ?';
 PREPARE S3 FROM STMT;
 EXECUTE S3 USING P_DEPT;
END
```

Chapter 4. Statements for SQL procedures

An SQL procedure consists of a CREATE PROCEDURE statement with a procedure body. This chapter contains the syntax and parameter descriptions for the CREATE PROCEDURE statement in “CREATE PROCEDURE (SQL procedure)” on page 16 and the syntax and parameter descriptions for the procedure body in “The SQL procedure body” on page 23.

CREATE PROCEDURE (SQL procedure)

The CREATE PROCEDURE statement specifies the source statements for an SQL procedure.

Invocation

This statement cannot be embedded in an application program or dynamically prepared. This statement can appear in the following places:

- As the only statement in a partitioned data set member that is input to the DB2 precompiler or the SQL procedure processor
- As the only statement in a character string that is an input parameter for the SQL procedure processor

For more information on preparing SQL procedures for execution, see “Chapter 5. Preparing and running an SQL procedure” on page 39.

Authorization

None required.

Syntax

```
                                           ┌─,───────────────────────────┐
                                           ▼                             │
►►─CREATE PROCEDURE─procedure-name─(─────┬───────────────────────┬───┴─)─option-list─────────────►◄
                                           └─parameter-declaration─┘
```

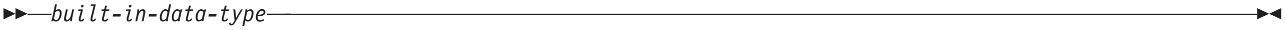
option-list:

```
   ┌─RESULT SET 0──────────────────┐                ┌─NO COLLID──────────────┐
►►─┼───────────────────────────────┼─LANGUAGE SQL─┼────────────────────────┼──────────────────►
   └─RESULT──┬─SET──┬───integer────┘                └─COLLID─collection-id─┘
             └─SETS─┘

                                                   ┌─ASUTIME NO LIMIT─────┐  ┌─STAY RESIDENT NO──┐
►──┬───────────────────────────────────────┬──┼──────────────────────┼──┼───────────────────┼──►
   ├─WLM ENVIRONMENT───┬─name──────────┬───┤  └─ASUTIME LIMIT─integer─┘  └─STAY RESIDENT YES─┘
   │                   └─(─name─,*─)───┘   │
   └─NO WLM ENVIRONMENT────────────────────┘
   ┌─PROGRAM TYPE MAIN─┐  ┌─SECURITY DB2──┐
►──┴───────────────────┴──┼───────────────┼──┬─────────────────────────────────┬──────────────►
                          └─SECURITY USER─┘  └─RUN OPTIONS─run-time-options─┘

   ┌─COMMIT ON RETURN NO──┐
►──┼──────────────────────┼─procedure-body─────────────────────────────────────────────────►◄
   └─COMMIT ON RETURN YES─┘
```

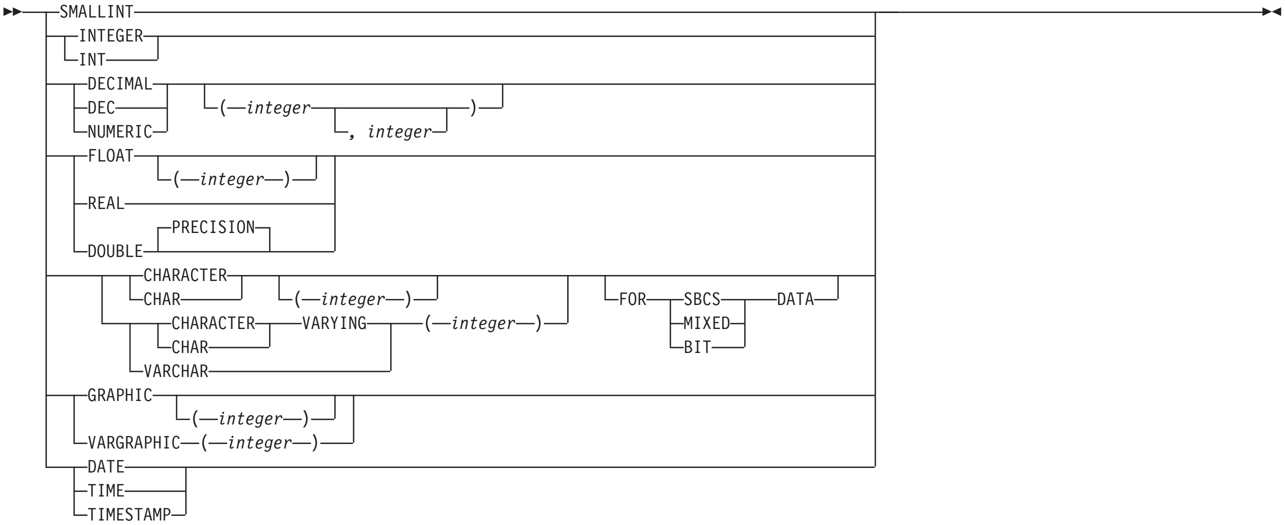
parameter-declaration:

```
   ┌─IN────┐
►►─┼───────┼─parameter-name──parameter-type─────────────────────────────────────────────►◄
   ├─OUT───┤
   └─INOUT─┘
```

parameter-type:



built-in-data-type:



Description

procedure-name
Names the stored procedure. The name is an unqualified long SQL identifer that must not identify an existing stored procedure at the current server.

The first eight bytes of the SQL procedure name form the default name for the SQL procedure load module. If you use the default load module name, the SQL procedure name must conform to MVS naming conventions for partitioned data set members.

(*parameter-declaration,...***)**
Specifies the number of parameters of the stored procedure and the data type of each parameter. A parameter for a stored procedure can be used only for input, only for output, or for both input and output. You must give each parameter a name.

IN Identifies the parameter as an input parameter to the stored procedure. The parameter does not contain a value when the stored procedure returns control to the calling SQL application.

IN is the default.

OUT
Identifies the parameter as an output parameter that is returned by the stored procedure.

INOUT
Identifies the parameter as both an input and output parameter for the stored procedure.

parameter-name
Names the parameter. *parameter-name* is a short SQL identifier, which can include only the characters A through Z, 0 through 9, or characters that correspond to the EBCDIC code points X'5B', X'7B', and X'7C', which correspond to $, #, and @ in code page 37 or 500. A parameter name cannot be an SQL reserved word. For a list of SQL reserved words, see "Appendix C. SQL reserved words" on page 51.

data-type
Specifies the data type of the parameter.

built-in-data-type
The data type of the parameter is a built-in data type. You can use the same built-in data types as for the CREATE TABLE statement except LONG VARCHAR or LONG VARGRAPHIC. Use VARCHAR or VARGRAPHIC with an explicit length instead.

The NUMERIC, DATE, TIME, and TIMESTAMP data types are valid in a CREATE PROCEDURE statement, but they are not valid data types for the PARMLIST column of SYSIBM.SYSPROCEDURES. When you define your SQL procedure to DB2 by inserting a row into SYSIBM.SYSPROCEDURES, you need to make the following substitutions:

Table 1. Substitutions for NUMERIC, DATE, TIME, and TIMESTAMP in SYSIBM.SYSPROCEDURES

For this data type in CREATE PROCEDURE	Substitute this data type in SYSPROCEDURES
NUMERIC	DECIMAL
DATE	VARCHAR(10)[1]
TIME	VARCHAR(8)[2]
TIMESTAMP	VARCHAR(26)

Notes:

1. If a date exit is installed on the DB2 subsystem, specify VARCHAR(*n*), where *n* is the length value from field LOCAL DATE LENGTH on installation panel DSNTIP4.
2. If a time exit is installed on the DB2 subsystem, specify VARCHAR(*n*), where *n* is the length value from field LOCAL TIME LENGTH on installation panel DSNTIP4.

For more information on the data types, including the subtype of character data types (the FOR *subtype* DATA clause), see "built-in-data-type" Chapter 6 of *SQL Reference*.

If you do not specify a specific value for the data types that have length, precision, or scale attributes (CHAR, GRAPHIC, DECIMAL, NUMERIC, FLOAT), the defaults are as follows:

CHAR	CHAR(1)
GRAPHIC	GRAPHIC(1)
DECIMAL	DECIMAL(5,0)
FLOAT	DOUBLE (length of 8)

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. 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.

RESULT SET *integer* or **RESULT SETS** *integer*
Specifies the maximum number of query result sets that the stored procedure can return. The default is RESULT SETS 0, which indicates that there are no result sets.

LANGUAGE
Specifies the application programming language in which the stored procedure is written.

SQL
The stored procedure is written in DB2 SQL procedure language.

NO COLLID or **COLLID** *collection-id*
Identifies the package collection that is used when the stored procedure is executed. This is the package collection into which the DBRM that is associated with the stored procedure is bound.

NO COLLID
The package collection for the stored procedure is the same as the package collection of the calling program. If the calling program does not use a package, the package collection is set to the value of special register CURRENT PACKAGESET.

NO COLLID is the default.

COLLID *collection-id*
The package collection for the stored procedure is the one specified.

WLM ENVIRONMENT
Identifies the MVS workload manager (WLM) environment in which the stored procedure is to run when the DB2 stored procedure address space is WLM-established. The *name* of the WLM environment is a long identifier that must not contain an underscore.

If you do not specify WLM ENVIRONMENT, the stored procedure runs in the default WLM-established stored procedure address space specified at installation time.

name
The WLM environment in which the stored procedure must run. If another stored procedure or a user-defined function calls the stored procedure and that calling routine is running in an address space that is not associated with the specified WLM environment, DB2 routes the stored procedure request to a different MVS address space.

(name,*)
When an SQL application program directly calls a stored procedure, the WLM environment in which the stored procedure runs.

If another stored procedure or a user-defined function calls the stored procedure, the stored procedure runs in the same WLM environment that the calling routine uses.

To define a stored procedure that is to run in a specified WLM environment, you must have appropriate authority for the WLM environment. For an example of a

RACF command that provides this authorization, see the discussion of CREATE PROCEDURE in Chapter 6 of *SQL Reference*.

NO WLM ENVIRONMENT
Indicates that the stored procedure is to run in the DB2-established stored procedure address space.

Do not specify NO WLM ENVIRONMENT if you implicitly or explicitly define the stored procedure with the SECURITY USER clause.

To define a stored procedure that is to run in the DB2-established stored procedure address space, you must have appropriate authority for the address space. For an example of a RACF command that provides this authorization, see the discussion of CREATE PROCEDURE in Chapter 6 of *SQL Reference*.

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

When you are debugging a stored procedure, setting a limit can be helpful in case the stored procedure gets caught in a loop. For information on service units, see *OS/390 MVS Initialization and Tuning Guide*.

NO LIMIT
There is no limit on the service units. NO LIMIT is the default.

LIMIT *integer*
The limit on the service units is a positive *integer* in the range of 1 to 2 GB. If the stored procedure uses more service units than the specified value, DB2 cancels the stored procedure.

STAY RESIDENT
Specifies whether the stored procedure load module remains resident in memory when the stored procedure ends.

NO
The load module is deleted from memory after the stored procedure ends. NO is the default.

YES
The load module remains resident in memory after the stored procedure ends.

PROGRAM TYPE
Specifies whether the stored procedure runs as a main routine or a subroutine.

MAIN
The stored procedure runs as a main routine. Only PROGRAM TYPE MAIN is allowed for an SQL procedure.

SECURITY
Specifies how the stored procedure interacts with an external security product, such as RACF, to control access to non-SQL resources.

DB2
The stored procedure does not require a special external security environment. If the stored procedure accesses resources that an external security product protects, the access is performed using the authorization ID associated with the stored procedure address space. DB2 is the default.

USER

An external security environment should be established for the stored procedure. If the stored procedure accesses resources that the external security product protects, the access is performed using the authorization ID of the user who invoked the stored procedure.

RUN OPTIONS *run-time-options*

Specifies the Language Environment run-time options to be used for the stored procedure. You must specify *run-time-options* as a character string that is no longer than 254 bytes. If you do not specify RUN OPTIONS or pass an empty string, DB2 does not pass any run-time options to Language Environment, and Language Environment uses its installation defaults.

For a description of the Language Environment run-time options, see *OS/390 Language Environment for OS/390 & VM Programming Reference*.

COMMIT ON RETURN

Indicates whether DB2 commits the transaction immediately on return from the stored procedure.

NO

DB2 does not issue a commit when the stored procedure returns. NO is the default.

YES

DB2 issues a commit when the stored procedure returns if the following statements are true:
- The SQLCODE that is returned by the CALL statement is not negative.
- The stored procedure is not in a must abort state.

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

If the stored 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.

procedure-body

Specifies the source code for an SQL procedure. See “The SQL procedure body” on page 23 for information on how to write a procedure body.

Notes

The following restrictions apply to the use of parameters in SQL procedures:

- If IN is specified for a parameter in an SQL procedure, the parameter cannot be modified within the SQL procedure body.
- If OUT is specified for a parameter in an SQL procedure, the parameter can be used only as the target of an assignment in the SQL procedure body. The parameter cannot be checked or used to set other variables. If the parameter is not set, DB2 returns the null value to the caller.

See the description of the CREATE PROCEDURE statement in Chapter 6 of *SQL Reference* for information about:
- Choosing data types for parameters
- Specifying the encoding scheme for parameters
- Environments for running stored procedures

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
  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 procedure runs in a WLM environment called PARTSA.
- The same input always produces the same output.
- SQL work is committed on return to the caller.
- The Language Environment run-time options to be used when the SQL procedure executes are 'MSGFILE(OUTFILE),RPTSTG(ON),RPTOPTS(ON)'.

```
CREATE PROCEDURE UPDATE_SALARY_1
 (IN EMPLOYEE_NUMBER CHAR(10),
 IN RATE DECIMAL(6,2))
 LANGUAGE SQL
 WLM ENVIRONMENT PARTSA
 COMMIT ON RETURN YES
 RUN OPTIONS 'MSGFILE(OUTFILE),RPTSTG(ON),RPTOPTS(ON)'
   UPDATE EMP
   SET SALARY = SALARY * RATE
   WHERE EMPNO = EMPLOYEE_NUMBER
```

For more examples of SQL procedures, see “The SQL procedure body” on page 23.

The SQL procedure body

The procedure body in an SQL stored procedure definition contains the source statements for the stored procedure.

This chapter contains syntax diagrams, semantic descriptions, rules, and examples of the use of the statements that constitute the procedure body.

Procedure body

The procedure body contains the source code for an SQL stored procedure.

Syntax

```
►►──┬─SQL-statement────────┬───────────────────────────────────────────────────────►◄
    ├─assignment-statement─┤
    ├─case-statement───────┤
    ├─compound-statement───┤
    ├─if-statement─────────┤
    ├─leave-statement──────┤
    ├─loop-statement───────┤
    ├─repeat-statement─────┤
    └─while-statement──────┘
```

Notes

See Table 4 on page 49 for a list of valid values for *SQL-statement*.

Assignment statement

The assignment statement assigns a value to an output parameter or to an SQL variable.

Syntax

```
►►──SET──┬─SQL-parameter-name─┬──=──┬─expression─┬──────────────────────────────────────────────►◄
         └─SQL-variable-name──┘     └─NULL───────┘
```

Description

SQL-parameter-name
Identifies the parameter that is the assignment target. The parameter must be specified in *parameter-declaration* in the CREATE PROCEDURE statement and must be defined as OUT or INOUT.

SQL-variable-name
Identifies the SQL variable that is the assignment target. An SQL variable must be declared before it is used. For information on declaring SQL variables, see “Compound statement” on page 28.

expression or NULL
Specifies the expression or value that is the assignment source. See Chapter 3 of *SQL Reference* for information on expressions.

Notes

Assignments statements in SQL procedures must conform to the SQL assignment rules. See Chapter 3 of *SQL Reference* for assignment rules.

The data type of the target and source must be compatible.

When a string is assigned to a fixed-length variable and the length of the string is less than the length attribute of the target, the string is padded on the right with the necessary number of single-byte or double-byte blanks.

When a string is assigned to a variable and the string is longer than the length attribute of the variable, a negative SQLCODE is set.

If truncation of the whole part of the number occurs on assignment to a numeric variable, a negative SQLCODE is set.

If an assignment statement is the only statement in the procedure body, the statement cannot end with a semicolon. Otherwise, the statement must end with a semicolon.

Examples

Increase the SQL variable p_salary by 10 percent.

```
SET p_salary = p_salary + (p_salary * .10)
```

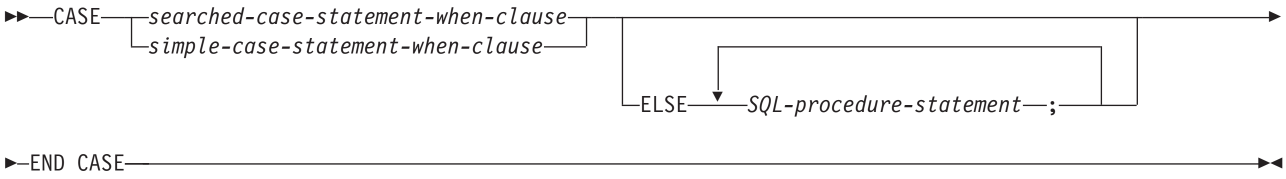
Set SQL variable p_salary to the null value.

```
SET p_salary = NULL
```

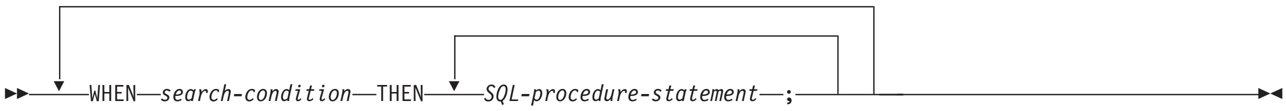
CASE statement

The CASE statement selects an execution path based on the evaluation of one or more conditions. A CASE statement operates in the same way as a CASE expression, which is discussed in Chapter 3 of *SQL Reference*.

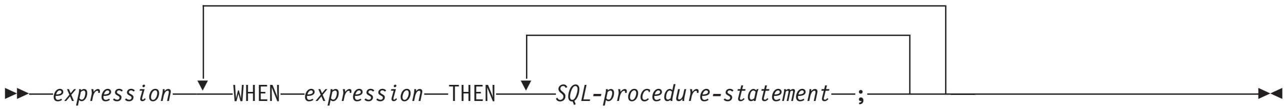
Syntax


CASE
searched-case-statement-when-clause
simple-case-statement-when-clause
ELSE
SQL-procedure-statement
;
END CASE

searched-case-statement-when-clause:



simple-case-statement-when-clause:



Description

CASE
Begins a *case-expression*.

searched-case-statement-when-clause
Specifies a search-condition that is applied to each row or group of table data presented for evaluation, and the result when that condition is true.

simple-case-statement-when-clause
Specifies that the value of the *expression* prior to the first WHEN keyword is tested for equality with the value of each *expression* that follows the WHEN keyword. Specifies the result for each WHEN keyword when the expressions are equal.

The *expression* prior to the first WHEN keyword is tested for equality with the value of the *expression* that follows the WHEN keyword. The data type of the *expression* prior to the first WHEN keyword must be comparable to the data types of each *expression* that follows the WHEN keywords.

SQL-procedure-statement
Specifies a statement that follows the THEN and ELSE keyword. The statement must be one of the statements listed under “SQL procedure statement” on page 37. It specifies the result of a *searched-case-statement-when-clause* or a *simple-case-statement-when-clause* that is true, or the result if no case is true.

search-condition
Specifies a condition that is true, false, or unknown about a row or group of table data. The search condition cannot contain a subselect.

END CASE
Ends a *case-statement*.

Notes

If none of the conditions specified in the WHEN are true, and an ELSE is not specified, an error is issued when the statement executes and the execution of the CASE statement is terminated.

CASE statements that use a simple case statement when clause can be nested up to three levels. CASE statements that use a searched statement when clause have no limit to the number of nesting levels.

If a CASE statement is the only statement in the procedure body, the statement cannot end with a semicolon. Otherwise, the statement must end with a semicolon.

Examples

Use a simple case statement when clause to update column DEPTNAME in table DEPT, depending on the value of SQL variable v_workdept.

```
CASE v_workdept
 WHEN 'A00'
  THEN UPDATE DEPT SET
   DEPTNAME = 'DATA ACCESS 1';
 WHEN 'B01'
  THEN UPDATE DEPT SET
   DEPTNAME = 'DATA ACCESS 2';
 ELSE UPDATE DEPT SET
   DEPTNAME = 'DATA ACCESS 3';
END CASE
```

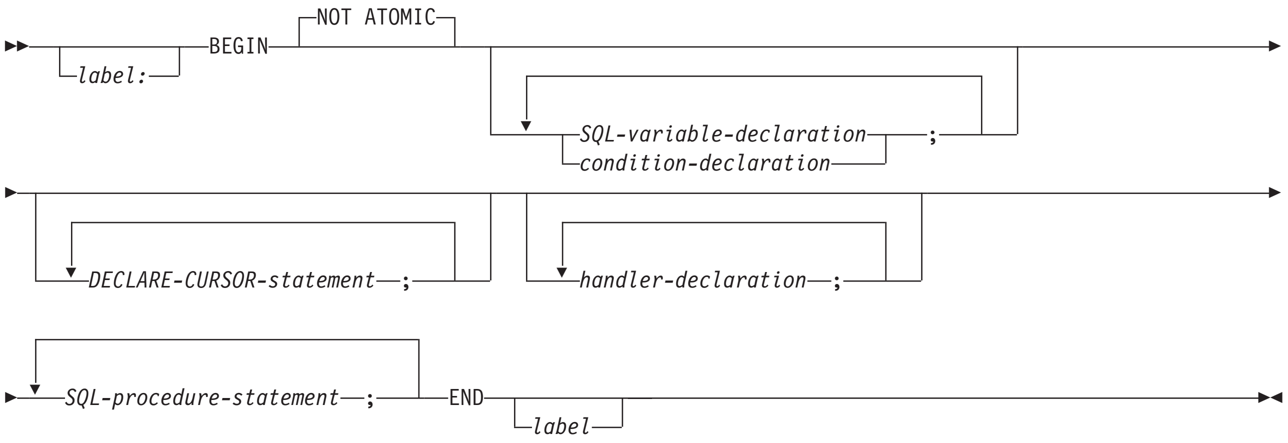
Use a searched case statement when clause to update column DEPTNAME in table DEPT, depending on the value of SQL variable v_workdept.

```
CASE
WHEN v_workdept = 'A00'
THEN UPDATE department SET
deptname = 'DATA ACCESS 1';
WHEN v_workdept = 'B01'
THEN UPDATE department SET
deptname = 'DATA ACCESS 2';
ELSE UPDATE department SET
deptname = 'DATA ACCESS 3';
END CASE
```

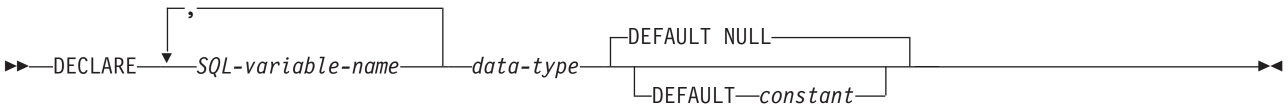
Compound statement

A compound statement contains a group of statements and declarations for SQL variables, cursors, and condition handlers.

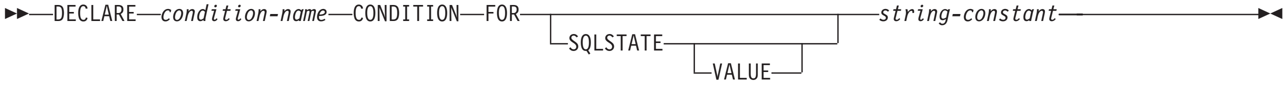
Syntax



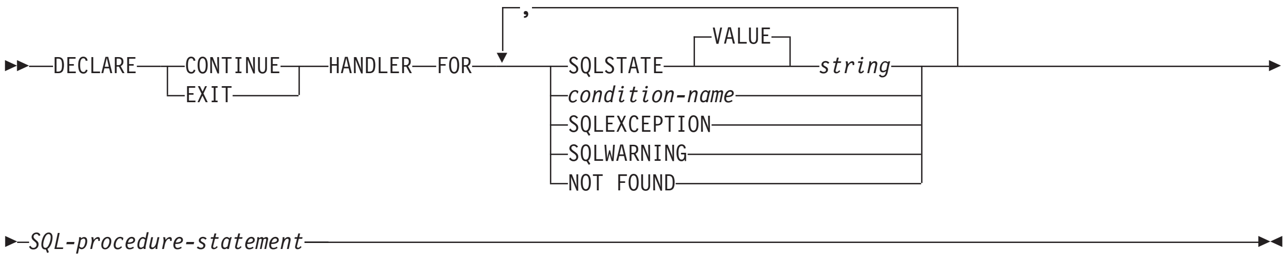
SQL-variable-declaration:



condition-declaration:



handler-declaration:



Description

label

Defines the label for the code block. If the beginning label is specified, it can be used to qualify SQL variables declared in the compound statement and can also be specified on a LEAVE statement. If the ending label is specified, it must be the same as the beginning label.

A label name cannot be the same as the name of the SQL procedure in which the label is used.

NOT ATOMIC
NOT ATOMIC indicates that an error within the compound statement does not cause the compound statement to be rolled back.

SQL-variable-declaration
Declares variable that is local to the compound statement.

SQL-variable-name
Specifies the name of a local variable. The name cannot be the same as another SQL variable within the same compound statement and cannot be the same as a parameter name or an SQL reserved word. DB2 folds all SQL variable names to uppercase. SQL variable names should not be the same as column names. If an SQL statement contains an SQL variable or parameter and a column reference with the same name, DB2 interprets the name as an SQL variable or parameter name. To refer to the column, qualify the name with the table name.

data-type
Specifies the data type and length of the variable. SQL variables follow the same rules for default lengths and maximum lengths as SQL procedure parameters. See “CREATE PROCEDURE (SQL procedure)” on page 16 for a description of SQL data types and lengths.

DEFAULT constant or NULL
Defines the default for the SQL variable. The variable is initialized when the SQL procedure is called. If a default value is not specified, the variable is initialized to NULL.

If the SQL variable name is SQLCODE and the data type is INT, the variable is used as a stand-alone SQLCODE in the procedure and can be checked to determine whether SQL statements are successful. Similarly, if the SQL variable name is SQLSTATE and the data type is CHAR(5), the variable is used as a stand-alone SQLSTATE in the procedure. After the SQLCODE and SQLSTATE variables are declared, they can be referenced anywhere in the procedure. The SQLCODE and SQLSTATE variables cannot be set to NULL. An SQLCODE or SQLSTATE variable should not be used on the left side of an assignment statement.

condition-declaration
Declares a condition name and corresponding SQLSTATE value.

condition-name
Specifies the name of the condition. The condition name is a long SQL identifier that must be unique within the procedure body and can be referenced only within the compound statement in which it is declared.

string-constant
Specifies the SQLSTATE that is associated with the condition. The string must be specified as five characters enclosed in single quotes, and cannot be '00000'.

declare-cursor-statement
Declares a cursor. Each cursor in the procedure body must have a unique name. The cursor can be referenced only from within the compound statement. For more information on declaring a cursor, see Chapter 6 of *SQL Reference*.

handler-declaration
Specifies a set of statements to execute when an exception or completion condition occurs in the compound statement. *SQL-procedure-statement* is the

set of statements that execute when the handler receives control. See “SQL procedure statement” on page 37 for information on *SQL-procedure-statement*.

A handler is active only within the compound statement in which it is declared.

The actions that a handler can perform are:

CONTINUE
After the handler is invoked successfully, control is returned to the SQL statement that follows the statement that raised the exception. If the error that raised the exception is an IF, CASE, or WHILE statement, control returns to the statement that follows END IF, END CASE, END WHILE, or END REPEAT.

EXIT
After the handler is invoked successfully, control is returned to the end of the compound statement.

The conditions that can cause the handler to gain control are:

SQLSTATE string
Specifies an SQLSTATE for which the handler is invoked. The SQLSTATE cannot be '00000'.

condition-name
Specifies a condition name for which the handler is invoked. The condition name must be previously defined in a condition declaration.

The conditions under which the handler is invoked are:

SQLEXCEPTION
Specifies that the handler is invoked when an SQLEXCEPTION occurs. An SQLEXCEPTION is an SQLSTATE in which the class code is a value other than "00", "01", or "02". For more information on SQLSTATE values, see Appendix C of *Messages and Codes*.

SQLWARNING
Specifies that the handler is invoked when an SQLWARNING occurs. An SQLWARNING is an SQLSTATE value with a class code of "01".

NOT FOUND
Specifies that the handler is invoked when a NOT FOUND condition occurs. NOT FOUND corresponds to an SQLSTATE value with a class code of "02".

Notes

The order of statements in a compound statement must be:

1. SQL variable and condition declarations
2. Cursor declarations
3. Handler declarations
4. Procedure body statements (CASE, IF, LOOP, REPEAT, WHILE, SQL)

Compound statements cannot be nested.

Unlike host variables, SQL variables are not preceded by colons when they are used in SQL statements.

Datetime arithmetic operations cannot be performed on SQL variables.

The following rules apply to handler declarations:

- A handler declaration that contains SQLEXCEPTION, SQLWARNING, or NOT FOUND cannot contain additional SQLSTATE or condition names.
- Handler declarations within the same compound statement cannot contain duplicate conditions.
- A handler declaration cannot contain the same condition code or SQLSTATE value more than once, and cannot contain an SQLSTATE value and a condition name that represent the same SQLSTATE value.
- If an error occurs for which there is no handler, execution of the compound statement is terminated.

If a compound statement is the only statement in the procedure body, the statement cannot end with a semicolon. Otherwise, the statement must end with a semicolon.

Examples

Create a procedure body with a compound statement that performs the following actions:

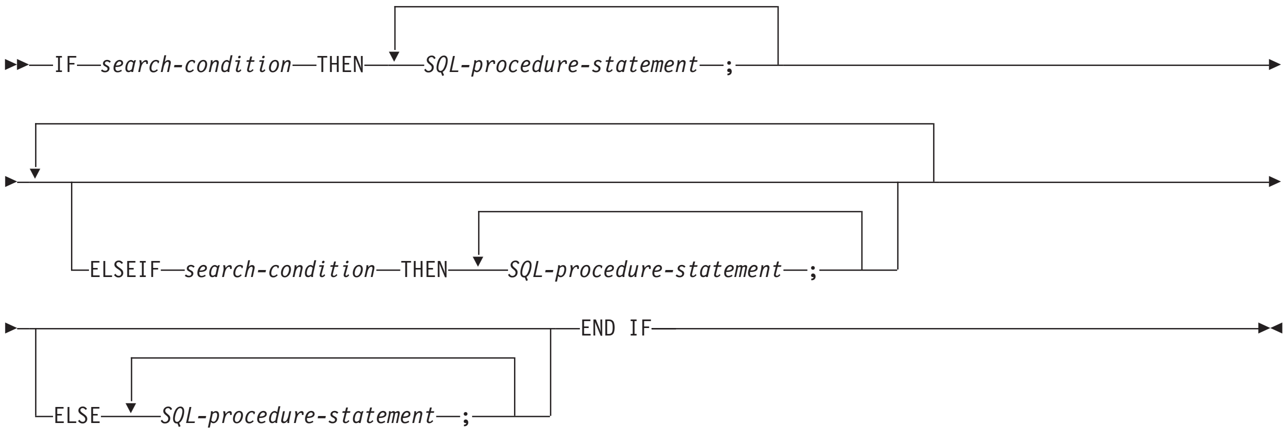
- Declares SQL variables, a condition for SQLSTATE '02000', a handler for the condition, and a cursor
- Opens the cursor, fetches a row, and closes the cursor

```
CREATE PROCEDURE PROC1(OUT NOROWS INT) LANGUAGE SQL
BEGIN
 DECLARE v_firstnme VARCHAR(12);
 DECLARE v_midinit CHAR(1);
 DECLARE v_lastname VARCHAR(15);
 DECLARE v_edlevel SMALLINT;
 DECLARE v_salary DECIMAL(9,2);
 DECLARE at_end INT DEFAULT 0;
 DECLARE not_found
  CONDITION FOR '02000';
 DECLARE c1 CURSOR FOR
 SELECT FIRSTNME, MIDINIT, LASTNAME,
  EDLEVEL, SALARY
  FROM EMP;
 DECLARE CONTINUE HANDLER FOR not_found SET NOROWS=1;
 OPEN c1;
 FETCH c1 INTO v_firstnme, v_midinit,
  v_lastname, v_edlevel, v_salary;
 CLOSE c1;
END
```

IF statement

The IF statement selects an execution path based on the evaluation of a condition.

Syntax



Description

search-condition
Specifies a search-condition that is applied to each row or group of table data presented for evaluation, and the result when that condition is true.

SQL-procedure-statement
Specifies a statement that follows the THEN and ELSE keyword. The statement must be one of the statements listed under “SQL procedure statement” on page 37.

Examples

Assign a value to the SQL variable new_salary based on the value of SQL variable rating.

```
IF rating = 1
 THEN SET new_salary =
  new_salary + (new_salary * .10);
 ELSEIF rating = 2
  THEN SET new_salary =
   new_salary + (new_salary * .05);
 ELSE SET new_salary =
  new_salary + (new_salary * .02);
END IF
```

LEAVE statement

The LEAVE statement transfers program control out of a loop or a block of code.

Syntax

```
►►──LEAVE──label──────────────────────────────────────────────────────────────►◄
```

Description

label

Specifies the label of the block or loop to exit.

A label name cannot be the same as the name of the SQL procedure in which the label is used.

Notes

When a LEAVE statement transfers control out of a compound statement, all open cursors in the compound statement, except cursors that are used to return result sets, are closed.

If a LEAVE statement is the only statement in the procedure body, the statement cannot end with a semicolon. Otherwise, the statement must end with a semicolon.

Examples

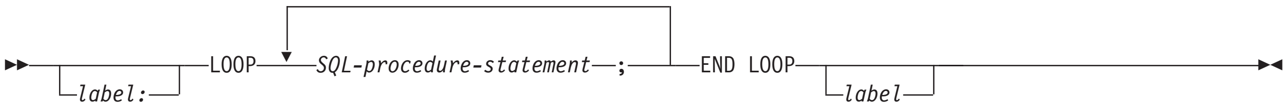
Use a LEAVE statement to transfer control out of a LOOP statement when a negative SQLCODE occurs.

```
ftch_loop: LOOP
 FETCH c1 INTO
  v_firstnme, v_midinit,
  v_lastname, v_edlevel, v_salary;
 IF SQLCODE=100 THEN LEAVE ftch_loop;
 END IF;
END LOOP
```

LOOP statement

The LOOP statement executes a statement or group of statements multiple times.

Syntax



Description

label

Specifies the label for the LOOP statement. If the ending label is specified, the beginning label must be specified, and the two must match.

A label name cannot be the same as the name of the SQL procedure in which the label is used.

SQL-procedure-statement

Specifies the statements to be executed in the loop.

Notes

If a LOOP statement is the only statement in the procedure body, the statement cannot end with a semicolon. Otherwise, the statement must end with a semicolon.

Examples

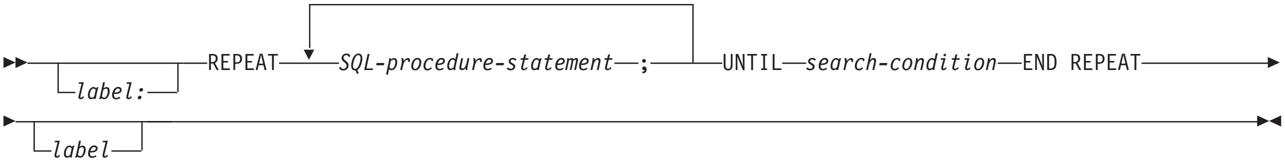
Use a LOOP statement to fetch rows from a table.

```
ftch_loop: LOOP
 FETCH c1 INTO
  v_firstnme, v_midinit,
  v_lastname, v_edlevel, v_salary;
 IF SQLCODE<>0 THEN SET badsql=1;
 END IF;
END LOOP
```

REPEAT statement

The REPEAT statement executes a statement or group of statements until a search condition is true.

Syntax



Description

label

Specifies the label for the REPEAT statement. If the ending label is specified, the beginning label must be specified, and the two must match.

A label name cannot be the same as the name of the SQL procedure in which the label is used.

SQL-procedure-statement

Specifies the statements to be executed.

search-condition

Specifies a condition that is evaluated after each execution of the SQL procedure statement. If the condition is true, the SQL procedure statement is not executed again.

Notes

If a REPEAT statement is the only statement in the procedure body, the statement cannot end with a semicolon. Otherwise, the statement must end with a semicolon.

Examples

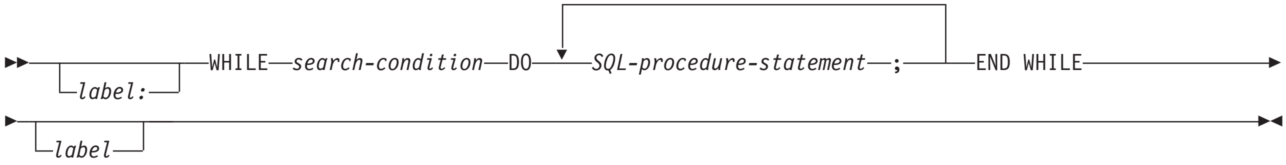
Use a REPEAT statement to fetch rows from a table.

```
fetch_loop:
REPEAT
 FETCH c1 INTO
  v_firstnme, v_midinit, v_lastname;
UNTIL
    SQLCODE <> 0
END REPEAT fetch_loop
```

WHILE statement

The WHILE statement repeats the execution of a statement or group of statements while a specified condition is true.

Syntax



Description

label

Specifies the label for the WHILE statement. If the ending label is specified, it must be the same as the beginning label.

A label name cannot be the same as the name of the SQL procedure in which the label is used.

search-condition

Specifies a condition that is evaluated before each execution of the loop. If the condition is true, the SQL procedure statement in the loop is executed.

SQL-procedure-statement

Specifies the statements to be executed in the loop.

Notes

If a WHILE statement is the only statement in the procedure body, the statement cannot end with a semicolon. Otherwise, the statement must end with a semicolon.

Examples

Use a WHILE statement to fetch rows from a table while SQL variable at_end, which indicates whether the end of the table has been reached, is 0.

```
WHILE at_end = 0 DO
 FETCH c1 INTO
  v_firstnme, v_midinit,
  v_lastname, v_edlevel, v_salary;
 IF SQLCODE=100 THEN SET at_end=1;
 END IF;
END WHILE
```

SQL procedure statement

Syntax

```
►►──┬─assignment-statement──┬──────────────────────────────────────────────►◄
    ├─case-statement────────┤
    ├─if-statement──────────┤
    ├─leave-statement───────┤
    ├─loop-statement────────┤
    ├─repeat-statement──────┤
    ├─while-statement───────┤
    └─nested-SQL-statement──┘
```

Notes

See Table 4 on page 49 for a list of valid values for *nested-SQL-statement*.

If an SQL procedure statement is the only statement in the procedure body, the statement cannot end with a semicolon. Otherwise, the statement must end with a semicolon.

Chapter 5. Preparing and running an SQL procedure

After you create the source statements for an SQL procedure, you need to prepare the procedure to run. This process involves two basic tasks:

- Creating an executable load module and a DB2 package from the SQL procedure source statements

  This task includes the following steps:

  - Preprocessing the CREATE PROCEDURE statement to generate a C language source program
  - Precompiling the C language source program to generate a DBRM and a modified C source program
  - Binding the DBRM to generate a DB2 package
- Defining the stored procedure to DB2

  This is done by inserting a row into the SYSIBM.SYSPROCEDURES catalog table that describes the SQL procedure. If you prepare an SQL procedure through the SQL procedure processor or the IBM DB2 Stored Procedure Builder, this task is performed for you.

There are three methods available for preparing an SQL procedure to run:

- Using IBM DB2 Stored Procedure Builder, which runs on Windows NT, Windows 95, or Windows 98.
- Using JCL. See “Using JCL to prepare an SQL procedure”.
- Using the DB2 for OS/390 SQL procedure processor. See “Using the DB2 for OS/390 SQL procedure processor to prepare an SQL procedure” on page 40

To run an SQL procedure, you must call it from a client program, using the SQL CALL statement. See the description of the CALL statement in Chapter 6 of *SQL Reference* for more information.

Using JCL to prepare an SQL procedure

Use the following steps to prepare an SQL procedure using JCL.

1. Preprocess the CREATE PROCEDURE statement.

   To do this, execute program DSNHPSM, with the file that contains the CREATE PROCEDURE statement as input. The output from this step is:
   - A C language source program
   - An INSERT statement for defining the stored procedure in SYSIBM.SYSPROCEDURES
2. Precompile the C language source program that was generated in step 1.

   This process produces a DBRM and modified C language source statements.

   You need to ensure that the DBRM name is the same as the name of the load module for the SQL procedure.
3. Compile and link-edit the modified C source statements that were produced in step 2.

   This process produces an executable C language program.

   The default name for the C language program is the first eight bytes of the SQL procedure name. You can override the default name, but you must ensure that the new name matches the name of the DBRM that is produced in step 2.
4. Bind the DBRM that was produced in step 2 into a package.
5. Define the stored procedure to DB2.

To do this, first modify the INSERT statement that was was produced in step 1 to match the characteristics of the SQL procedure. For example, if you change the load module name for the SQL procedure, you must change the LOADMOD value in the INSERT statement. Then execute the INSERT statement to add a row for the SQL procedure to SYSIBM.SYSPROCEDURES.

Using the DB2 for OS/390 SQL procedure processor to prepare an SQL procedure

The SQL procedure processor, DSNTPSMP, is a REXX stored procedure that you can use to prepare an SQL procedure for execution. You can also use DSNTPSMP to perform selected steps in the preparation process or delete an existing SQL procedure. The following sections contain information on invoking DSNTPSMP.

Environment for calling and running DSNTPSMP

You can invoke DSNTPSMP only through an SQL CALL statement in an application program or through IBM DB2 Stored Procedure Builder.

Before you can run DSNTPSMP, you need to perform the following steps to set up the DSNTPSMP environment:

1. Install the PTFs for DB2 APARs PQ24199 and PQ29706.
2. Install DB2 for OS/390 REXX Language Support feature.

   Contact your IBM service representative for more information.
3. If you plan to call DSNTPSMP directly, write and prepare an application program that executes an SQL CALL statement for DSNTPSMP.

   See “Writing and preparing an application that calls DSNTPSMP” on page 42 for more information.

   If you plan to invoke DSNTPSMP through the IBM DB2 Stored Procedure Builder, see the following URL for information on installing and using the IBM DB2 Stored Procedure Builder.

   `http://www.software.ibm.com/data/db2/os390/spb`
4. Define DSNTPSMP to DB2.

   Customize and run job DSNTIJSQ to perform this task.
5. Create DB2 tables and indexes that are used by DSNTPSMP. Job DSNTIJSQ performs this task. See “Creating tables that are used by DSNTPSMP” on page 42.
6. Set up a WLM environment in which to run DSNTPSMP. See Section 5 (Volume 2) of *Administration Guide* for general information on setting up WLM application environments for stored procedures and “Setting up a WLM application environment for DSNTPSMP” for specific information for DSNTPSMP.

Setting up a WLM application environment for DSNTPSMP

You must run DSNTPSMP in a WLM-established stored procedures address space. You should run only DSNTPSMP in that address space, and you should not run multiple copies of DSNTPSMP concurrently.

Figure 3 on page 41 shows sample JCL for a startup procedure for the address space in which DSNTPSMP runs.

```
//DSNWLM   PROC RGN=0K,APPLENV=WLMTEST,DB2SSN=DSN,NUMTCB=1     1
//IEFPROC EXEC PGM=DSNX9WLM,REGION=&RGN,TIME=NOLIMIT,
//        PARM='&DB2SSN,&NUMTCB,&APPLENV'
//STEPLIB  DD  DISP=SHR,DSN=DSN510.RUNLIB.LOAD                 2
//         DD  DISP=SHR,DSN=CBC.SCBCCMP
//         DD  DISP=SHR,DSN=CEE.SCEERUN
//         DD  DISP=SHR,DSN=DSN510.SDSNLOAD
//SYSEXEC  DD  DISP=SHR,DSN=DSN510.SDSNCLST                    3
//SYSTSPRT DD  SYSOUT=A
//CEEDUMP  DD  SYSOUT=A
//SYSPRINT DD  SYSOUT=A
//SYSABEND DD  DUMMY
//SQLSRC   DD  DSN=USER.PSMLIB.DATA,DISP=SHR                   4
//SQLDBRM  DD  DISP=SHR,DSN=DSN510.DBRMLIB.DATA                5
//SQLCIN   DD  DISP=SHR,USER.SRCLIB.C                          6
//SQLLMOD  DD  DISP=SHR,DSN=DSN510.RUNLIB.LOAD                 7
//SQLLIBC  DD  DISP=SHR,DSN=CEE.SCEEH.H                        8
//SQLLIBL  DD  DISP=SHR,DSN=CEE.SCEELKED                       9
//         DD  DISP=SHR,DSN=DSN510.RUNLIB.LOAD
//         DD  DISP=SHR,DSN=DSN510.SDSNEXIT
//         DD  DISP=SHR,DSN=DSN510.SDSNLOAD
//SYSMSGS  DD  DSN=EDC.SEDCDMSG(EDCMSGE),DISP=SHR              10
```

Figure 3. Startup procedure for a WLM address space in which DSNTPSMP runs

Notes to Figure 3:

1 APPLENV specifies the application environment in which DSNTPSMP runs. To ensure that DSNTPSMP always uses the correct data sets and parameters for preparing each SQL procedure, you can set up different application environments for preparing different types of SQL procedures. For example, if all payroll applications use the same set of data sets during program preparation, you could set up an application environment called PAYROLL for preparing only payroll applications. The startup procedure for PAYROLL would point to the data sets that are used for payroll applications.

DB2SSN specifies the DB2 subsystem name.

NUMTCB specifies the number of programs that can run concurrently in the address space. You should always set NUMTCB to 1 to ensure that executions of DSNTPSMP occur serially.

2 STEPLIB specifies the Language Environment run-time library that DSNTPSMP uses when it runs.

3 SYSEXEC specifies the library that contains DSNTPSMP.

4 SQLSRC specifies the library into which DSNTPSMP puts the SQL procedure source code if the source code is passed to DSNTPSMP in an input string.

5 DBRMLIB specifies the library into which DSNTPSMP puts the DBRM that it generates when it precompiles your SQL procedure.

6 SQLCIN specifies the library into which DSNTPSMP puts the C source code that DB2 generates for the SQL procedure.

7 SQLLMOD specifies the library into which DSNTPSMP puts the load module that it generates when it compiles and link-edits your SQL procedure.

8 SQLLIBC specifies the library that contains standard C header files. This library is used during compilation of the generated C program.

9 SQLLIBL specifies the following libraries, which DSNTPSMP uses when it link-edits the SQL procedure:

- Language Environment run-time library
- DB2 application load library
- DB2 exit library
- DB2 load library

10 SQLMSGS specifies the library that contains messages that are used by the C prelink-edit utility.

Creating tables that are used by DSNTPSMP

DSNTPSMP uses two DB2 tables and and three indexes:

- Table SYSIBM.SYSPSM holds the source code for SQL procedures that DSNTPSMP prepares.
- Table SYSIBM.SYSPSMOPTS holds information about the program preparation options that you specify when you invoke DSNTPSMP.
- Index SYSIBM.DSNPSMX1 is an index on SYSIBM.SYSPSM.
- Index SYSIBM.DSNPSMX2 is a unique index on SYSIBM.SYSPSM.
- Index SYSIBM.DSNPSMOX1 is a unique index on SYSIBM.SYSPSMOPTS.

Before you can run DSNTPSMP, SYSIBM.SYSPSM, SYSIBM.SYSPSMOPTS, and SYSIBM.DSNPSMOX1 must exist on your DB2 subsystem. “Appendix A. DB2 objects required by the SQL procedure processor” on page 47 shows the format of these objects. To create the objects, customize job DSNTIJSQ according to the instructions in its prolog, then execute DSNTIJSQ.

Authorization to execute DSNTPSMP

The program that invokes DSNTPSMP must have the following authorizations:

- Authorization to execute the CALL statement. See the description of the CALL statement in Chapter 6 of *SQL Reference* for more information.
- The BIND privilege for any stored procedure packages that DSNTPSMP binds.

Writing and preparing an application that calls DSNTPSMP

DSNTPSMP must be invoked through an SQL CALL statement in an application program. This section contains information that you need to write and prepare the calling application.

DSNTPSMP Syntax

```
►►──CALL──DSNTPSMP──(──function──,──SQL-procedure-name──,──┬─SQL-procedure-source─┬──,──────────────►
                                                           └─empty-string─────────┘

►──┬─bind-options─┬──,──┬─compiler-options─┬──,──┬─precompiler-options─┬──,────────────────────────►
   └─empty-string─┘     └─empty-string─────┘     └─empty-string────────┘

►──┬─prelink-edit-options─┬──,──┬─link-edit-options─┬──,──┬─run-time-options─┬──,──────────────────►
   └─empty-string─────────┘     └─empty-string──────┘     └─empty-string─────┘

►──┬─source-data-set-name─┬──,──)────────────────────────────────────────────────────────────────►◄
   └─empty-string─────────┘
```

bind-options, compiler-options, precompiler-options, prelink-edit-options, link-edit options,* or *run-time-options:

```
          ┌─,───────┐
          ▼         │
►►──'─────option────┴──'─────────────────────────────────────────────────────────────────────────►◄
```

DSNTPSMP parameters

function

A VARCHAR(20) input parameter that identifies the task that you want DSNTPSMP to perform. The tasks are:

BUILD

Creates a load module, DBRM, package, and definition for an SQL procedure.

If you choose the create function, and an SQL procedure with name *SQL-procedure-name* already exists, DSNTPSMP issues a warning message and terminates.

DESTROY

Deletes the load module, DBRM, package, stored source code, and definition for an SQL procedure.

For the DESTROY function to execute successfully, the current SQL ID of the application that calls DSNTPSMP must match the value of BUILDOWNER in the row for the SQL procedure in the SYSIBM.SYSPSMOPTS table.

REBUILD

Replaces the load module, package, and definition for an SQL procedure.

REBIND

Rebinds an SQL procedure package.

ALTER_RUNOPTS

Changes the run-time options for an SQL procedure.

SQL-procedure-name

A VARCHAR(8) input parameter performs the following meanings:

- Specifies the SQL procedure name for the DESTROY, REBIND, or ALTER_RUNOPTS function
- Specifies the name of the SQL procedure load module for the BUILD or REBUILD function

SQL-procedure-source

A VARCHAR(37675) input parameter that contains the source code for the SQL procedure. If you specify an empty string for this parameter, you need to specify the name of a data set that contains the SQL procedure source code, in *source-data-set-name*.

bind-options

A VARCHAR(255) input parameter that contains the options that you want to specify for binding the SQL procedure package. For a list of valid bind options, see Chapter 2 of *Command Reference*.

You must specify the PACKAGE bind option for the BUILD, REBUILD, and REBIND functions.

compiler-options

A VARCHAR(255) input parameter that contains the options that you want to specify for compiling the C language program that DB2 generates for the SQL procedure. For a list of valid compiler options, see *IBM C/C++ for OS/390 User's Guide*.

precompiler-options

A VARCHAR(255) input parameter that contains the options that you want to specify for precompiling the C language program that DB2 generates for the SQL procedure. For a list of valid precompiler options, see Section 5 of *Application Programming and SQL Guide*.

prelink-edit-options
A VARCHAR(255) input parameter that contains the options that you want to specify for prelink-editing the C language program that DB2 generates for the SQL procedure. For a list of valid prelink-edit options, see *IBM C/C++ for OS/390 User's Guide*.

link-edit-options
A VARCHAR(255) input parameter that contains the options that you want to specify for link-editing the C language program that DB2 generates for the SQL procedure. For a list of valid link-edit options, see *DFSMS/MVS: Program Management*.

run-time-options
A VARCHAR(254) input parameter that contains the Language Environment run-time options that you want to specify for the SQL procedure. For a list of valid Language Environment run-time options, see *OS/390 Language Environment for OS/390 & VM Programming Reference*.

source-data-set-name
A VARCHAR(80) input parameter that contains the name of an MVS sequential data set or partitioned data set member that contains the source code for the SQL procedure. If you specify an empty string for this parameter, you need to provide the SQL procedure source code in *source-procedure-source*.

return-codes
A VARCHAR(255) output parameter in which DB2 puts the return codes from all steps of the DSNTPSMP invocation.

Result sets that DSNTPSMP returns

When you invoke DSNTPSMP, DB2 returns a result set that contains messages and listings from each step that DSNTPSMP performs. To obtain the information from the result set, you can write your client program to retrieve information from one result set with known contents. However, for greater flexibility, you might want to write your client program to retrieve data from an unknown number of result sets with unknown contents. Both techniques are shown in Section 6 of *Application Programming and SQL Guide*.

Each row of the result set contains the following information:

Processing step
The step in the *function* process to which the message applies.

ddname
The ddname of the data set that contains the message.

Sequence number
The sequence number of a line of message text within a message.

Message
A line of message text.

Rows in the message result set are ordered by processing step, ddname, and sequence number.

Examples of DSNTPSMP invocation

DSNTPSMP BUILD function: Call DSNTPSMP to build an SQL procedure. The information that DSNTPSMP needs is:

Function	BUILD
Source location	String in variable procsrc
Bind options	SQLERROR(NOPACKAGE), VALIDATE(RUN), ISOLATION(RR), RELEASE(COMMIT)
Compiler options	SOURCE, LIST, MAR(1,80), LONGNAME, RENT
Precompiler options	HOST(SQL), SOURCE, XREF, MAR(1,72), STDSQL(NO)
Prelink-edit options	None specified
Link-edit options	AMODE=31, RMODE=ANY, MAP, RENT
Run-time options	MSGFILE(OUTFILE), RPTSTG(ON), RPTOPTS(ON)

The CALL statement is:

```
EXEC SQL CALL DSNTPSMP('BUILD','',procsrc,
 'SQLERROR(NOPACKAGE),VALIDATE(RUN),ISOLATION(RR),RELEASE(COMMIT)',
 'SOURCE,LIST,MAR(1,80),LONGNAME,RENT',
 'HOST(SQL),SOURCE,XREF,MAR(1,72),STDSQL(NO)',
 '',
 'AMODE=31,RMODE=ANY,MAP,RENT',
 'MSGFILE(OUTFILE),RPTSTG(ON),RPTOPTS(ON)',
 '');
```

DSNTPSMP DESTROY function: Call DSNTPSMP to delete an SQL procedure definition and the associated load module. The information that DSNTPMSP needs is:

Function	DESTROY
SQL procedure name	OLDPROC

The CALL statement is:

```
EXEC SQL CALL DSNTPSMP('DESTROY','OLDPROC','',
 '','','','','','','');
```

DSNTPSMP REBUILD function: Call DSNTPSMP to recreate an existing SQL procedure. The information that DSNTPMSP needs is:

Function	REBUILD
Source location	Member PROCSRC of partitioned data set DSN510.SDSNSAMP
Bind options	SQLERROR(NOPACKAGE), VALIDATE(RUN), ISOLATION(RR), RELEASE(COMMIT)
Compiler options	SOURCE, LIST, MAR(1,80), LONGNAME, RENT
Precompiler options	HOST(SQL), SOURCE, XREF, MAR(1,72), STDSQL(NO)
Prelink-edit options	MAP
Link-edit options	AMODE=31, RMODE=ANY, MAP, RENT
Run-time options	MSGFILE(OUTFILE), RPTSTG(ON), RPTOPTS(ON)

The CALL statement is:

```
EXEC SQL CALL DSNTPSMP('REBUILD','','',
 'SQLERROR(NOPACKAGE),VALIDATE(RUN),ISOLATION(RR),RELEASE(COMMIT)',
 'SOURCE,LIST,MAR(1,80),LONGNAME,RENT',
 'HOST(SQL),SOURCE,XREF,MAR(1,72),STDSQL(NO)',
 'MAP',
 'AMODE=31,RMODE=ANY,MAP,RENT',
 'MSGFILE(OUTFILE),RPTSTG(ON),RPTOPTS(ON)',
 'DSN510.SDSNSAMP(PROCSRC)');
```

DSNTPSMP REBIND function: Call DSNTPSMP to rebind the package for an existing SQL procedure. The information that DSNTPMSP needs is:

Function	REBIND
SQL procedure name	SQLPROC

Run-time options	VALIDATE(BIND), ISOLATION(RR), RELEASE(DEALLOCATE)

The CALL statement is:

```
EXEC SQL CALL DSNTPSMP('REBIND','SQLPROC','',
 'VALIDATE(BIND),ISOLATION(RR),RELEASE(DEALLOCATE)',
 '','','','','','');
```

DSNTPSMP ALTER_RUNOPTS function: Call DSNTPSMP to change Language Environment run-time options for an existing SQL procedure. The information that DSNTPMSP needs is:

Function	ALTER_RUNOPTS
SQL procedure name	SQLPROC
Run-time options	POSIX(ON), TEST(,,,VADTCPIP&9.63.51.17:*)

The CALL statement is:

```
EXEC SQL CALL DSNTPSMP('ALTER_RUNOPTS','SQLPROC','',
 '','','','','',
 'POSIX(ON),TEST(,,,VADTCPIP&9.63.51.17:*)','');
```

Preparing a program that invokes DSNTPSMP

To prepare the program that calls DSNTPSMP for execution, you need to perform the following steps:

1. Precompile, compile, and link-edit the application program.
2. Bind a package for the application program.
3. Bind the package for DB2 REXX support, DSNTRXCS.DSNTREXX, and the package for the application program into a plan.

Sample programs to help you prepare and run SQL procedures

Table 2 lists the sample jobs that DB2 provides to help you prepare and run SQL procedures. All samples are in data set DSN510.SDSNSAMP. Before you can run the samples, you must customize them for your installation. See the prolog of each sample for specific instructions.

Table 2. SQL procedure samples shipped with DB2

Member that contains source code	Contents	Purpose
DSNHSQL	JCL procedure	Preprocesses, precompiles, compiles, prelink-edits, and link-edits an SQL procedure
DSNTEJ63	JCL job	Invokes JCL procedure DSNHSQL to prepare SQL procedure DSN8ES1 for execution
DSN8ES1	SQL procedure	A stored procedure that accepts a department number as input and returns a result set that contains salary information for each employee in that department
DSNTEJ64	JCL job	Prepares client program DSN8ED3 for execution
DSN8ED3	C program	Calls SQL procedure DSN8ES1

Appendix A. DB2 objects required by the SQL procedure processor

The SQL procedure processor (DSNTPSMP) uses the tables and indexes that are described in the following sections. It is recommended that you create the tables and indexes in their own database and table space. You can these objects by customizing and running job DSNTIJSQ.

Table spaces and indexes

47 shows the table spaces to which the SQL procedure tables are assigned, and which indexes are defined on the tables.

Table 3. Table spaces and indexes for SQL procedure tables

TABLE SPACE DSNDPSM. ...	**TABLE SYSIBM. ...**	**Page**	**INDEX SYSIBM. ...**	**INDEX FIELDS**
DSNSPSM	SYSPSM	47	DSNPSMX1	PROCEDURENAME
			DSNPSMX2	SCHEMA PROCEDURENAME SEQNO
	SYSPSMOPTS	47	DSNPSMOX1	SCHEMA PROCEDURENAME

The SQL procedure source table (SYSIBM.SYSPSM)

SYSIBM.SYSPSM is used by the SQL procedure processor and IBM DB2 Stored Procedure Builder to hold the source code for a stored procedure. SYSIBM.SYSPSM contains at least one row for each SQL procedure that is prepared by the SQL procedure processor or SQL Procedure Builder. The number of rows that represent an SQL procedure is

`CEILING(n/3800)`

n is the number of bytes in the SQL procedure source statement.

Column Name	**Data Type**	**Description**	**Use**
SCHEMA	CHAR(8)	Schema of the SQL procedure. Blank for SQL procedures created before DB2 Version 6.	G
PROCEDURENAME	CHAR(18) NOT NULL	Name of the SQL procedure.	G
SEQNO	SMALLINT NOT NULL	Number of the SQL statement piece in PROCCREATESTMT. SEQNO is between 1 and CEILING(*n*/3800), where *n* is the number of bytes in the SQL procedure source statement.	G
PSMDATE	DATE NOT NULL	The date on which the SQL procedure was created.	G
PSMTIME	TIME NOT NULL	The time at which the SQL procedure was created.	G
PSMTIME	TIME NOT NULL	The time at which the SQL procedure was created.	G

Column Name	Data Type	Description	Use
PROCCREATESTMT	VARCHAR(3800) NOT NULL	All or part of an SQL procedure source statement. If the SQL procedure statement is more than 3800 bytes, this field contains the portion of the source statement indicated by SEQNO.	G

The SQL procedure options table (SYSIBM.SYSPSMOPTS)

SYSIBM.SYSPSMOPTS is used by the SQL procedure processor and IBM DB2 Stored Procedure Builder to hold the program preparation options for an SQL procedure. SYSIBM.SYSPSMOPTS contains one row for each SQL procedure that is prepared by the SQL procedure processor or SQL Procedure Builder.

Column Name	Data Type	Description	Use
SCHEMA	CHAR(8)	Schema of the SQL procedure. Blank for SQL procedures created before DB2 Version 6.	G
PROCEDURENAME	CHAR(18) NOT NULL	Name of the SQL procedure.	G
BUILDSCHEMA	CHAR(8)	The schema name that is the qualifier for the procedure name that is specified in the BUILDNAME column. The schema name is SYSPROC.	G
BUILDNAME	CHAR(18)	A procedure name that is associated with stored procedure DSNTPSMP. Users of DSNTPSMP might create several stored procedure definitions for DSNTPSMP so that they can run DSNTPSMP in different WLM environments. The caller specifies the environment in which DSNTPSMP runs by specifying the procedure name that is associated with that environment in the SQL CALL statement.	G
BUILDOWNER	CHAR(8)	The authorization ID that was used to create the SQL procedure.	G
PRECOMPILE_OPTS	VARCHAR(256)	The options that were specified in the *precompiler-options* parameter in the most recent invocation of DSNTPSMP for the SQL procedure specified in this row.	G
COMPILE_OPTS	VARCHAR(256)	The options that were specified in the *compiler-options* parameter in the most recent invocation of DSNTPSMP for the SQL procedure specified in this row.	G
PRELINK_OPTS	VARCHAR(256)	The options that were specified in the *prelink-edit-options* parameter in the most recent invocation of DSNTPSMP for the SQL procedure specified in this row.	G
LINK_OPTS	VARCHAR(256)	The options that were specified in the *link-edit-options* parameter in the most recent invocation of DSNTPSMP for the SQL procedure specified in this row.	G
BIND_OPTS	VARCHAR(1024)	The options that were specified in the *bind-options* parameter in the most recent invocation of DSNTPSMP for the SQL procedure specified in this row.	G
SOURCEDSN	VARCHAR(255)	If the SQL procedure source code that is input to DSNTPSMP is stored in a data set, the name of that data set.	G

Appendix B. SQL statements allowed in SQL procedures

Table 4 lists the SQL statements that are valid in an SQL procedure body. The table lists the SQL statements that can be used as the only statement in the SQL procedure and the statements that can be nested in a compound statement.

Table 4. Valid SQL statements in an SQL procedure body

	SQL statement is...	
SQL statement	**The only statement in the procedure**	**Nested in a compound statement**
ALLOCATE CURSOR		
ALTER	**Y**	**Y**
ASSOCIATE LOCATORS		
BEGIN DECLARE SECTION		
CALL		
CLOSE		**Y**
COMMENT ON	**Y**	**Y**
COMMIT		
CONNECT (Type 1 and Type 2)		
CREATE	**Y**	**Y**
DECLARE CURSOR		
DECLARE STATEMENT		
DECLARE TABLE		
DELETE	**Y**	**Y**
DESCRIBE		
DESCRIBE CURSOR		
DESCRIBE INPUT		
DESCRIBE PROCEDURE		
DROP	**Y**	**Y**
END DECLARE SECTION		
EXECUTE		**Y**
EXECUTE IMMEDIATE	**Y**	**Y**
EXPLAIN		
FETCH		**Y**
FREE LOCATOR		
GRANT	**Y**	**Y**
HOLD LOCATOR		
INCLUDE		
INSERT	**Y**	**Y**
LABEL ON	**Y**	**Y**
LOCK TABLE	**Y**	**Y**
OPEN		**Y**

Table 4. Valid SQL statements in an SQL procedure body (continued)

	SQL statement is...	
SQL statement	**The only statement in the procedure**	**Nested in a compound statement**
PREPARE FROM		**Y**
RELEASE		
RENAME	**Y**	**Y**
REVOKE	**Y**	**Y**
ROLLBACK		
SELECT		
SELECT INTO	**Y**	**Y**
SET Assignment		
SET CONNECTION		
SET special register		
UPDATE	**Y**	**Y**
WHENEVER		

Appendix C. SQL reserved words

Table 5 lists the words that cannot be used as ordinary identifiers in some contexts because they might be interpreted as SQL keywords. For example, ALL cannot be a column name in a SELECT statement. Each word, however, can be used as a delimited identifier in contexts where it otherwise cannot be used as an ordinary identifier. For example, if the quotation mark (") is the escape character that begins and ends delimited identifiers, “ALL” can appear as a column name in a SELECT statement. In addition, some sections of this book might indicate words that cannot be used in the specific context that is being described.

Table 5. SQL reserved words

ADD
AFTER
ALL
ALLOW
ALTER
AND
ANY
AS
ASUTIME
AUDIT
AUX
AUXILIARY
BEFORE
BEGIN
BETWEEN
BUFFERPOOL
BY
CALL
CAPTURE
CASCADED
CASE
CAST
CCSID
CHAR
CHARACTER
CHECK
CLOSE
CLUSTER
COLLECTION
COLLID
COLUMN
COMMENT
COMMIT
CONCAT
CONDITION
CONNECT
CONNECTION
CONSTRAINT
CONTAINS
CONTINUE
CREATE
CURRENT
CURRENT_DATE
CURRENT_LC_CTYPE
CURRENT_PATH
CURRENT_TIME
CURRENT_TIMESTAMP
CURSOR
DATA
DATABASE
DAY
DAYS
DBINFO
DB2SQL
DECLARE
DEFAULT
DELETE
DESCRIPTOR
DETERMINISTIC
DISALLOW
DISTINCT
DO
DOUBLE
DROP
DSSIZE
EDITPROC
ELSE
ELSEIF
END
END-EXEC[1]
ERASE
ESCAPE
EXCEPT
EXECUTE
EXISTS
EXIT
EXTERNAL
FENCED
FETCH
FIELDPROC
FINAL
FOR
FROM
FULL
FUNCTION
GENERAL
GENERATED
GO
GOTO
GRANT
GROUP
HANDLER
HAVING
HOUR
HOURS
IF
IMMEDIATE
IN
INDEX
INNER
INOUT
INSERT
INTO
IS
ISOBID
JOIN
KEY
LABEL
LANGUAGE
LC_CTYPE
LEAVE
LEFT
LIKE
LOCAL
LOCALE
LOCATOR
LOCATORS
LOCK
LOCKMAX
LOCKSIZE
LONG
LOOP
MICROSECOND
MICROSECONDS
MINUTE
MINUTES
MODIFIES
MONTH
MONTHS
NAME
NO
NOT
NULL
NULLS
NUMPARTS
OBID
OF
ON
OPEN
OPTIMIZATION
OPTIMIZE
OR
ORDER
OUT
OUTER
OVERRIDING
PACKAGE
PARAMETER
PART
PATH
PIECESIZE
PLAN
PRECISION
PREPARE
PRIQTY
PRIVILEGES
PROCEDURE
PROGRAM
PSID
QUERYNO
READS
REFERENCES
RELEASE
RENAME
REPEAT
RESTRICT
RESULT
RETURN
RETURNS
REVOKE
RIGHT
ROLLBACK
RUN
SAVEPOINT
SCHEMA
SCRATCHPAD
SECOND
SECONDS
SECQTY
SECURITY
SELECT
SET
SIMPLE
SOME
SOURCE
SPECIFIC
STANDARD
STAY
STOGROUP
STORES
STYLE
SUBPAGES
SYNONYM
SYSFUN
SYSIBM
SYSPROC
SYSTEM
TABLE
TABLESPACE
THEN
TO
TRIGGER
TYPE
UNDO
UNION
UNIQUE
UNTIL
UPDATE
USER
USING
VALIDPROC
VALUE
VALUES
VARIANT
VCAT
VIEW
VOLUMES
WHEN
WHERE
WHILE
WITH
WLM
YEAR
YEARS

Note: [1] COBOL only

IBM SQL has additional reserved words that DB2 for OS/390 does not enforce. Therefore, we suggest that you do not use these additional reserved words as ordinary identifiers in names that have a continuing use. See *IBM SQL Reference* for a list of the words.

Appendix D. Messages for SQL procedures

The following messages are generated when errors occur during program preparation of SQL procedures.

DSNH20060I E *csectname* **LINE** *nnnn* **COL** *cc* **UNSUPPORTED DATA TYPE** *data-type* **ENCOUNTERED IN SQL** *object-type object-name*

Explanation: *data-type* was specified in the definition of *object-name*. *object-type* is an SQL procedure parameter or variable. *data-type* is not supported for SQL procedure parameters or variables.

You can use the same built-in data types for SQL procedure parameters or variables that you can use for the CREATE TABLE statement, except these:
- LONG VARCHAR
- LONG VARGRAPHIC
- CLOB
- DBCLOB
- BLOB
- ROWID
- distinct type

Severity: 8 (error)

System Action: The statement cannot be executed.

User Response: Change the syntax to specify one of the supported data types. Instead of a LONG VARCHAR or CLOB data type, use a VARCHAR data type with an explicit length. Instead of a LONG VARGRAPHIC or DBCLOB data type, use a VARGRAPHIC data type with an explicit length.

DSNH20061I E *csectname* **LINE** *nnnn* **COL** *cc* **UNEXPECTED ERROR RETURNED FROM LANGUAGE ENVIRONMENT: REASON CODE** *reason-code***, RETURN CODE** *return-code module-name*

Explanation: An Language Environment error occurred while the DB2 precompiler was processing an SQL procedure. The reason codes and associated return codes are:

Reason code	Meaning and associated return code
1	No PIPI token. *return-code* is the return code from the CEEPIPI(init_sub) call.
2	CEE could not be loaded. *return-code* is the return code from the CEEPIPI(add_entry) call.
3	PIPI would not terminate. *return-code* is the return code from the CEEPIPI(term) call.
4	Call to *module-name* failed. *return-code* is the return code from the CEEPIPI(call_sub) call.
5	Bad response from PIPI. *return-code* is the return code from *module-name*.

See the explanation of return codes for the appropriate CEEPIPI call in *OS/390 Language Environment for OS/390 & VM Programming Guide* for explanations of the Language Environment return codes.

Severity: 8 (error)

System Action: The statement cannot be executed.

User Response: Correct the condition that is described by *reason-code*.

DSNH4775I E *csectname* **LINE** *nnnn* **COL** *cc* **STATEMENT NOT ALLOWED IN A COMPOUND SQL STATEMENT**

Explanation: In an SQL procedure, a compound statement contains an SQL procedure statement that is not allowed.

Severity: 8 (error)

System Action: The statement cannot be executed.

User Response: Remove the incorrect statement from the compound statement. See “The SQL procedure body” on page 23 for the correct syntax for a compound statement in an SQL procedure.

DSNH4776I E *csectname* **LINE** *nnnn* **COL** *cc* **CURSOR** *cursor-name* **SPECIFIED IN FOR STATEMENT NOT ALLOWED**

Explanation: In an SQL procedure, a FOR statement contains an OPEN, FETCH, or CLOSE statement for cursor *cursor-name*. Cursor operations are not allowed in the FOR statement.

Severity: 8 (error)

System Action: The statement cannot be executed.

User Response: Remove the incorrect statement from the compound statement.

DSNH4777I E *csectname* **LINE** *nnnn* **COL** *cc* **NESTED COMPOUND STATEMENTS NOT ALLOWED**

Explanation: An SQL procedure contains nested compound statements, which are not allowed.

Severity: 8 (error)

System Action: The statement cannot be executed.

User Response: Rewrite the SQL procedure body so that it does not contain nested compound statements.

DSNH4778I E *csectname* LINE *nnnn* COL *cc* END LABEL *label-name* NOT SAME AS BEGIN LABEL

Explanation: An SQL procedure statement cantains an ending label and a beginning label that do not match.

Severity: 8 (error)

System Action: The statement cannot be executed.

User Response: Change the ending label in the statement to match the beginning label.

DSNH4779I E *csectname* LINE *nnnn* COL *cc* LABEL *label-name* SPECIFIED ON LEAVE STATEMENT IS NOT VALID

Explanation: In an SQL procedure, the label on a LEAVE statement does not match the label for a block of code or loop that contains the LEAVE statement.

Severity: 8 (error)

System Action: The statement cannot be executed.

User Response: Change the label in the LEAVE statement to match a label in the loop or block of code that contains the LEAVE statement.

DSNH4780I E *csectname* LINE *nnnn* COL *cc* UNDO SPECIFIED FOR A HANDLER AND ATOMIC NOT SPECIFIED

Explanation: In an SQL procedure, a compound statement is defined as NOT ATOMIC, but the compound statement contains an UNDO handler. An UNDO handler can be used only for a compound statement that is defined as ATOMIC.

Severity: 8 (error)

System Action: The statement cannot be executed.

User Response: Define the compound statement as ATOMIC, or change the UNDO handler to a CONTINUE or EXIT handler.

DSNH4781I E *csectname* LINE *nnnn* COL *cc* CONDITION *condition-name* SPECIFIED IN HANDLER NOT DEFINED

Explanation: In an SQL procedure, a handler is declared for condition *condition-name*, but the SQL procedure does not contain a condition declaration statement that defines *condition-name*.

Severity: 8 (error)

System Action: The statement cannot be executed.

User Response: Include a condition declaration statement in the SQL procedure that relates *condition-name* to an SQLSTATE value.

DSNH4782I E *csectname* LINE *nnnn* COL *cc* CONDITION VALUE *value* SPECIFIED IN HANDLER NOT VALID

Explanation: In an SQL procedure, a condition handler is not valid for one of the following reasons:

- The handler specifies an SQLSTATE value that is not valid.
- The handler specifies duplicate conditions.
- The handler specifies SQLWARNING, SQLEXCEPTION, or NOT FOUND with other conditions.

Severity: 8 (error)

System Action: The statement cannot be executed.

User Response: Specify a valid condition in the handler. Ensure that a handler specifies a condition only once.

DSNH4785I E *csectname* LINE *nnnn* COL *cc* USE OF SQLCODE OR SQLSTATE NOT VALID

Explanation: In an SQL procedure, the name SQLCODE or SQLSTATE is used in one of the following invalid ways:

- An SQLCODE is declared as an SQL variable with a data type other than INTEGER.
- An SQLSTATE is declared as an SQL variable with a data type other than CHAR(5).
- An SQLCODE or SQLSTATE is declared as an SQL variable with DEFAULT NULL.
- An SQLCODE or SQLSTATE is assigned the value NULL in an assignment statement.
- An SQLCODE or SQLSTATE is the name of an SQL procedure parameter.

Severity: 8 (error)

System Action: The statement cannot be executed.

User Response: Correct the declaration or assignment of the SQLCODE or SQLSTATE.

Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
J74/G4
555 Bailey Avenue
P.O. Box 49023
San Jose, CA 95161-9023
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Programming interface information

This book is intended to help the customer write applications that use REXX to access IBM DB2 for OS/390 servers. This book documents General-use Programming Interface and Associated Guidance Information provided by DATABASE 2 for OS/390 (DB2 for OS/390).

General-use programming interfaces allow the customer to write programs that obtain the services of DB2 for OS/390.

Trademarks

The following terms are trademarks of the International Business Machines Corporation in the United States, or other countries, or both:

AD/Cycle
AIX
APL2
AS/400
BookManager
CICS
CICS/ESA
CICS/MVS
COBOL/370
C/370
DATABASE 2
DataPropagator
DB2
DB2 Extenders
DB2 Universal Database
DFSMSdfp
DFSMShsm
DFSMS/MVS
Distributed Relational Database Architecture
DRDA
IBM
IMS
IMS/ESA
Language Environment
MVS/ESA
MVS/XA
Net.Data
OS/2
OS/390
OS/400
Parallel Sysplex
QMF
RACF
SQL/DS
System/370
System/390
VTAM

Lotus and Notes are trademarks of Lotus Development Corporation in the United States, or other countries, or both

Microsoft™, Windows™, Windows NT™, and the Windows logo are trademarks or registered trademarks of Microsoft Corporation in the United States and/or other countries.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and/or other countries.

NetView™ is a trademark of Tivoli Systems Inc. in the United States, or other countries, or both.

Other company, product, and service names may be trademarks or service marks of others.

Index

Program Number: 5655-DB2

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.