DB2 for OS/390

REXX Language Support

Version 5

Note

Before using this information and the product it supports, read the information in “Notices” on page 41.

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 IBM DATABASE 2 Server for OS/390 REXX Language Support, which is a separately-orderable feature of DB2. DB2 REXX Language Support provides the ability to write SQL application programs in the REXX programming language. 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 who know the REXX programming language.

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 MVS/Enterprise Systems Architecture (MVS/ESA) or the MVS 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.

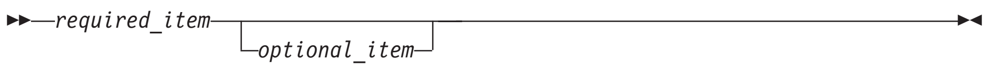
  The ▶── symbol indicates that a statement is continued from the previous line.

  The ──▶◀ symbol indicates the end of a statement.

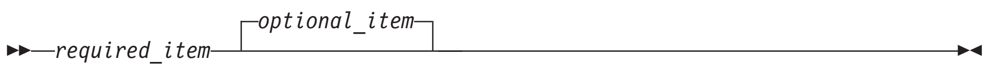
  Diagrams of syntactical units other than complete statements start with the ▶── symbol and end with the ──▶ symbol.
- Required items appear on the horizontal line (the main path).

```
▶▶──required_item──────────────────────────────────────────────▶◀
```

- Optional items appear below the main path.

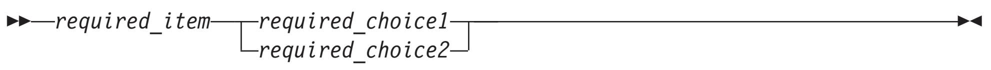


If an optional item appears above the main path, that item has no effect on the execution of the statement and is used only for readability.

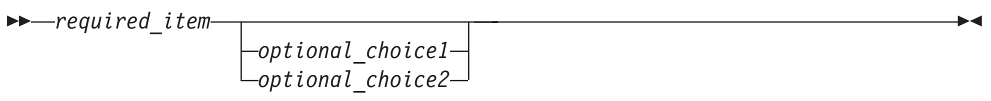


- If you can choose from two or more items, they appear vertically, in a stack.

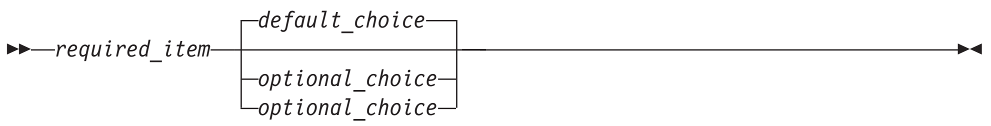
  If you *must* choose one of the items, one item of the stack appears on the main path.



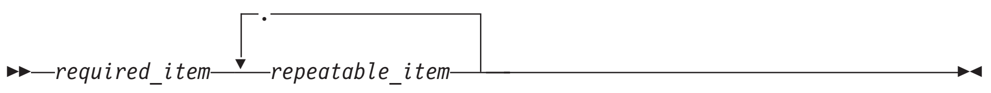
  If choosing one of the items is optional, the entire stack appears below the main path.



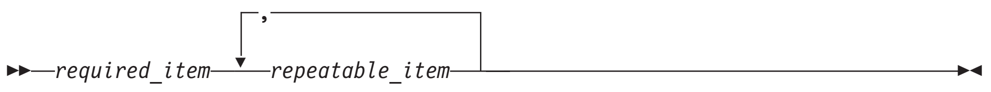
  If one of the items is the default, it appears above the main path and the remaining choices are shown below.



- 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. Coding SQL statements in a REXX application

This section helps you with the programming techniques that are unique to coding SQL statements in a REXX procedure. For an example of a complete DB2 REXX procedure, see “Appendix. Sample DB2 REXX application” on page 29.

Defining the SQL communication area

When DB2 prepares a REXX procedure that contains SQL statements, DB2 automatically includes an SQL communication area (SQLCA) in the procedure. The REXX SQLCA differs from the SQLCA for other languages in the following ways:

- The REXX SQLCA consists of a set of separate variables, rather than a structure.

  If you use the `ADDRESS DSNREXX 'CONNECT'` *`ssid`* syntax to connect to DB2, the SQLCA variables are a set of simple variables.

  If you connect to DB2 using the `CALL SQLDBS 'ATTACH TO'` syntax, the SQLCA variables are compound variables that begin with the stem SQLCA.

  See “Accessing the DB2 REXX Language Support application programming interfaces” on page 8 for a discussion of the methods for connecting a REXX application to DB2.
- You cannot use the INCLUDE SQLCA statement to include an SQLCA in a REXX program.

Table 1 lists the variables in a REXX SQLCA.

Table 1. Variables in a REXX SQLCA

Variable	Contents
SQLCODE	The SQL return code.
SQLERRMC	One or more tokens, separated by X'FF', that are substituted for variables in the descriptions of error conditions.
SQLERRP	A product signature and, in the case of an error, diagnostic information such as the name of the module that detected the error. For DB2 for OS/390, the product signature is 'DSN'.
SQLERRD.1	An internal error code.
SQLERRD.2	An internal error code.
SQLERRD.3	The number of rows affected after INSERT, UPDATE, and DELETE (but not rows deleted as a result of CASCADE delete). Set to 0 if the SQL statement fails, indicating that all changes made in executing the statement were canceled. Set to -1 for a mass delete from a table in a segmented table space. For SQLCODE -911 or -913, SQLERRD.3 can also contain the reason code for a timeout or deadlock.
SQLERRD.4	Generally contains timerons, a short floating-point value that indicates a rough relative estimate of resources required. This value does not reflect an estimate of the time required to execute the SQL statement. After you prepare an SQL statement, you can use this field as an indicator of the relative cost of the prepared SQL statement. For a particular statement, this number can vary with changes to the statistics in the catalog. This value is subject to change between releases of DB2 for OS/390.
SQLERRD.5	The position or column of a syntax error for a PREPARE or EXECUTE IMMEDIATE statement.
SQLERRD.6	An internal error code.

Table 1. Variables in a REXX SQLCA (continued)

Variable	Contents
SQLWARN.0	Blank if all other indicators are blank; W if at least one other indicator also contains a W.
SQLWARN.1	W if the value of a string column was truncated when assigned to a host variable.
SQLWARN.2	W if null values were eliminated from the argument of a column function; not necessarily set to W for the MIN function because its results are not dependent on the elimination of null values.
SQLWARN.3	W if the number of result columns is larger than the number of host variables. Z if the ASSOCIATE LOCATORS statement contains fewer locators than the stored procedure returned.
SQLWARN.4	W if a prepared UPDATE or DELETE statement does not include a WHERE clause.
SQLWARN.5	W if the SQL statement was not executed because it is not a valid SQL statement in DB2 for OS/390.
SQLWARN.6	W if the addition of a month or year duration to a DATE or TIMESTAMP value results in an invalid day (for example, June 31). Indicates that the value of the day was changed to the last day of the month to make the result valid.
SQLWARN.7	W if one or more nonzero digits were eliminated from the fractional part of a number that was used as the operand of a decimal multiply or divide operation.
SQLWARN.8	W if a character that could not be converted was replaced with a substitute character.
SQLWARN.9	W if arithmetic exceptions were ignored during COUNT DISTINCT processing. Z if the stored procedure returned multiple result sets.
SQLWARN.10	W if at least one character field of the SQLCA is invalid due to a character conversion error.
SQLSTATE	A return code for the outcome of the most recent execution of an SQL statement.

DB2 sets the SQLCODE and SQLSTATE values after each SQL statement executes. An application can check these variable values to determine whether the last SQL statement was successful.

Defining SQL descriptor areas

The following statements require an SQL descriptor area (SQLDA):
- CALL...USING DESCRIPTOR *descriptor-name*
- DESCRIBE *statement-name* INTO *descriptor-name*
- DESCRIBE CURSOR *host-variable* INTO *descriptor-name*
- DESCRIBE INPUT *statement-name* INTO *descriptor-name*
- DESCRIBE PROCEDURE *host-variable* INTO *descriptor-name*
- DESCRIBE TABLE *host-variable* INTO *descriptor-name*
- EXECUTE...USING DESCRIPTOR *descriptor-name*
- FETCH...USING DESCRIPTOR *descriptor-name*
- OPEN...USING DESCRIPTOR *descriptor-name*
- PREPARE...INTO *descriptor-name*

A REXX procedure can contain more than one SQLDA. Each SQLDA consists of a set of REXX variables with a common stem. The stem must be a REXX variable name that contains no periods and is the same as the value of *descriptor-name* that you specify when you use the SQLDA in an SQL statement. DB2 does not support the INCLUDE SQLDA statement in REXX.

Table 2 shows the variable names in a REXX SQLDA. The values in the second column of the table are values that DB2 inserts into the SQLDA when the statement executes. Except where noted otherwise, the values in the third column of the table are values that the application must put in the SQLDA before the statement executes.

Table 2. Fields of a REXX SQLDA

Variable name	Usage in DESCRIBE and PREPARE INTO	Usage in FETCH, OPEN, EXECUTE, and CALL
stem.SQLD	The number of columns that are described in the SQLDA. Double that number if USING BOTH appears in the DESCRIBE or PREPARE INTO statement. Contains a 0 if the statement string is not a query. For DESCRIBE PROCEDURE, the number of result sets returned by the stored procedure. Contains a 0 if no result sets are returned.	The number of host variables that are used by the SQL statement.
Each SQLDA contains *stem*.SQLD of the following variables. Therefore, 1<=*n*<=*stem*.SQLD. There is one occurrence of each variable for each column of the result table or host variable that is described by the SQLDA.		
stem.n.SQLTYPE	Indicates the data type of the column or parameter and whether it can contain null values. For a description of the type codes, see Appendix C of *SQL Reference*.	Indicates the data type of the host variable and whether an indicator variable is provided. Host variables for datetime values must be character string variables. For FETCH, a datetime type code means a fixed-length character string. For a description of the type codes, see Appendix C of *SQL Reference*.
stem.n.SQLLEN	For a column other than a DECIMAL or NUMERIC column, the length attribute of the column or parameter. For datetime data, the length of the string representation of the value. See Appendix C of *SQL Reference* for a description of allowable values.	For a host variable that does not have a decimal data type, the length attribute of the host variable. See Appendix C of *SQL Reference* for a description of allowable values.
stem.n.SQLPRECISION	For a DECIMAL or NUMERIC column, the precision of the column or parameter.	For a host variable with a decimal data type, the precision of the host variable.
stem.n.SQLSCALE	For a DECIMAL or NUMERIC column, the scale of the column or parameter.	For a host variable with a decimal data type, the scale of the host variable.
stem.n.SQLCCSID	For a string column or parameter, the CCSID of the column or parameter.	For a string host variable, the CCSID of the host variable.
stem.n.SQLLOCATOR	For DESCRIBE PROCEDURE, the value of a result set locator.	Not used.
stem.n.SQLDATA	Not used.	Before EXECUTE or OPEN, contains the value of an input host variable. The application must supply this value. After FETCH, contains the values of an output host variable.

Table 2. Fields of a REXX SQLDA (continued)

Variable name	**Usage in DESCRIBE and PREPARE INTO**	**Usage in FETCH, OPEN, EXECUTE, and CALL**
stem.n.SQLIND	Not used.	Before EXECUTE or OPEN, contains a negative number to indicate that the input host variable in *stem.n*.SQLDATA is null. The application must supply this value. After FETCH, contains a negative number if the value of the output host variable in *stem.n*.SQLDATA is null.
stem.n.SQLNAME	The name of the *n*th column in the result table. For DESCRIBE PROCEDURE, contains the cursor name that is used by the stored procedure to return the result set. The values for SQLNAME appear in the order that the cursors were opened by the stored procedure.	Not used.

Accessing the DB2 REXX Language Support application programming interfaces

DB2 REXX Language Support includes the following application programming interfaces:

CONNECT
Connects the REXX procedure to a DB2 subsystem. You must execute CONNECT before you can execute SQL statements. The syntax of CONNECT is:

```
                                          (1)
►►──┬─────────────────┬──'CONNECT'──────┬─'subsystem-ID'─┬───────────────────────────────────►◄
    └─Address DSNREXX─┘                 └─REXX-variable──┘
```

Notes:

1 CALL SQLDBS 'ATTACH TO' *-ssid* is equivalent to ADDRESS DSNREXX 'CONNECT' *-ssid*.

EXECSQL
Executes SQL statements in REXX procedures. The syntax of EXECSQL is:

```
                                          (1)
►►──┬─────────────────┬──"EXECSQL"──────┬─"SQL-statement"─┬──────────────────────────────────►◄
    └─Address DSNREXX─┘                 └─REXX-variable───┘
```

Notes:

1 CALL SQLEXEC is equivalent to EXECSQL.

See "Embedding SQL statements in a REXX procedure" on page 9 for more information.

DISCONNECT
Disconnects the REXX procedure from a DB2 subsystem. You should execute

DISCONNECT to release resources that are held by DB2. The syntax of DISCONNECT is:

```
                                              (1)
►►──┬──────────────────┬──'DISCONNECT'──────────────────────────────────────────────►◄
    └─Address DSNREXX──┘
```

Notes:

1 CALL SQLDBS 'DETACH' is equivalent to DISCONNECT.

These application programming interfaces are available through the DSNREXX host command environment. To make DSNREXX available to the application, invoke the RXSUBCOM function. The syntax is:

```
►►──RXSUBCOM──(───┬─'ADD'────┬──,──'DSNREXX'──,──'DSNREXX'───)──────────────────────►◄
                  └─'DELETE'─┘
```

The ADD function adds DSNREXX to the REXX host command environment table. The DELETE function deletes DSNREXX from the REXX host command environment table.

Figure 1 shows an example of REXX code that makes DSNREXX available to an application.

```
'SUBCOM DSNREXX'                       /* HOST CMD ENV AVAILABLE?     */

IF RC THEN                             /* IF NOT, MAKE IT AVAILABLE   */
  S_RC = RXSUBCOM('ADD','DSNREXX','DSNREXX')
                                       /* ADD HOST CMD ENVIRONMENT    */

ADDRESS DSNREXX                        /* SEND ALL COMMANDS OTHER     */
                                       /* THAN REXX INSTRUCTIONS TO   */
                                       /* DSNREXX                     */
                                       /* CALL CONNECT, EXECSQL, AND  */
                                       /* DISCONNECT INTERFACES       */
⋮

S_RC = RXSUBCOM('DELETE','DSNREXX','DSNREXX')
                                       /* WHEN DONE WITH              */
                                       /* DSNREXX, REMOVE IT.         */
```

Figure 1. Making DSNREXX available to an application

Embedding SQL statements in a REXX procedure

You can code SQL statements in a REXX procedure wherever you can use REXX commands. DB2 REXX Language Support allows all SQL statements that DB2 for OS/390 supports, *except* the following statements:

- BEGIN DECLARE SECTION
- DECLARE STATEMENT
- END DECLARE SECTION
- INCLUDE
- SELECT INTO
- WHENEVER

Each SQL statement in a REXX procedure must begin with EXECSQL, in either upper-, lower-, or mixed-case. One of the following items must follow EXECSQL:

- An SQL statement enclosed in single or double quotation marks.
- A REXX variable that contains an SQL statement. The REXX variable must not be preceded by a colon.

For example, you can use either of the following methods to execute the COMMIT statement in a REXX procedure:

```
EXECSQL "COMMIT"
```

```
rexxvar="COMMIT"
EXECSQL rexxvar
```

You cannot execute a SELECT, INSERT, UPDATE, or DELETE statement that contains host variables. Instead, you must execute PREPARE on the statement, with parameter markers substituted for the host variables, and then use the host variables in an EXECUTE, OPEN, or FETCH statement. See “Using REXX host variables and data types” on page 11 for more information.

An SQL statement follows rules that apply to REXX commands. The SQL statement can optionally end with a semicolon and can be enclosed in single or double quotation marks, as in the following example:

```
'EXECSQL COMMIT';
```

Comments: You cannot include REXX comments (/* ... */) or SQL comments (--) within SQL statements. However, you can include REXX comments anywhere else in the procedure.

Continuation for SQL statements: SQL statements that span lines follow REXX rules for statement continuation. You can break the statement into several strings, each of which fits on a line, and separate the strings with commas or with concatenation operators followed by commas. For example, either of the following statements is valid:

```
EXECSQL ,
   "UPDATE DSN8510.DEPT" ,
   "SET MGRNO = '000010'" ,
   "WHERE DEPTNO = 'D11'"
```

```
"EXECSQL " || ,
"  UPDATE DSN8510.DEPT " || ,
"  SET MGRNO = '000010'" || ,
"  WHERE DEPTNO = 'D11'"
```

Including code: The EXECSQL INCLUDE statement is not valid for REXX. You therefore cannot include externally defined SQL statements in a procedure.

Margins: Like REXX commands, SQL statements can begin and end anywhere on a line.

Names: You can use any valid REXX name that does not end with a period as a host variable. However, host variable names should not begin with 'SQL', 'RDI', 'DSN', 'RXSQL', or 'QRW'. Variable names can be at most 64 bytes.

Nulls: A REXX null value and an SQL null value are different. The REXX language has a null string (a string of length 0) and a null clause (a clause that contains only blanks and comments). The SQL null value is a special value that is distinct from all nonnull values and denotes the absence of a value. Assigning a REXX null value to a DB2 column does not make the column value null.

Statement labels: You can precede an SQL statement with a label, in the same way that you label REXX commands.

Handling errors and warnings: DB2 does not support the SQL WHENEVER statement in a REXX procedure. To handle SQL errors and warnings, use the following methods:

- To test for SQL errors or warnings, test the SQLCODE or SQLSTATE value and the SQLWARN. values after each EXECSQL call. This method does not detect errors in the REXX interface to DB2.
- To test for SQL errors or warnings or errors or warnings from the REXX interface to DB2, test the REXX RC variable after each EXECSQL call. Table 3 lists the values of the RC variable.

  You can also use the REXX SIGNAL ON ERROR and SIGNAL ON FAILURE keyword instructions to detect negative values of the RC variable and transfer control to an error routine.

Table 3. REXX return codes after SQL statements

Return code	Meaning
0	No SQL warning or error occurred.
+1	An SQL warning occurred.
-1	An SQL error occurred.

Using cursors and statement names

In REXX SQL applications, you must use a predefined set of names for cursors or prepared statements. The following names are valid for cursors and prepared statements in REXX SQL applications:

c1 to c100
Cursor names for DECLARE CURSOR, OPEN, CLOSE, and FETCH statements. Use c1 to c50 for cursors that are defined without the WITH HOLD option. Use c51 to c100 for cursors that are defined with the WITH HOLD option. All cursors are defined with the WITH RETURN option, so any cursor name can be used to return result sets from a REXX stored procedure.

c101 to c200
Cursor names for ALLOCATE, DESCRIBE, FETCH, and CLOSE statements that are used to retrieve result sets in a program that calls a stored procedure.

s1 to s100
Prepared statement names for DECLARE STATEMENT, PREPARE, DESCRIBE, and EXECUTE statements.

Use only the predefined names for cursors and statements. Do not use any of the predefined names for host variables.

Using REXX host variables and data types

You do not declare host variables in REXX. When you need a new variable, you use it in a REXX command. When you use a REXX variable as a host variable in an SQL statement, you must precede the variable with a colon.

A REXX host variable can be a simple or compound variable. DB2 REXX Language Support evaluates compound variables before DB2 processes SQL statements that contain the variables. In the following example, the host variable that is passed to DB2 is :x.1.2:

```
a=1
b=2
EXECSQL 'OPEN C1 USING :x.a.b'
```

Determining equivalent SQL and REXX data types

All REXX data is string data. Therefore, when a REXX procedure assigns input data to a table column, DB2 converts the data from a string type to the table column type. When a REXX procedure assigns column data to an output variable, DB2 converts the data from the column type to a string type.

When you assign input data to a DB2 table column, you can either let DB2 determine the type that your input data represents, or you can use an SQLDA to tell DB2 the intended type of the input data.

Letting DB2 determine the input data type

You can let DB2 assign a data type to input data based on the format of the input string. Table 4 shows the SQL data types that DB2 assigns to input data and the corresponding formats for that data. The two SQLTYPE values that are listed for each data type are the value for a column that does not accept null values and the value for a column that accepts null values.

If you do not assign a value to a host variable before you assign the host variable to a column, DB2 returns an error code.

Table 4. SQL input data types and REXX data formats

SQL data type assigned by DB2	SQLTYPE for data type	REXX input data format
INTEGER	496/497	A string of numerics that does not contain a decimal point or exponent identifier. The first character can be a plus (+) or minus (−) sign. The number that is represented must be between -2147483647 and 2147483647, inclusive.
DECIMAL(*p*,*s*)	484/485	One of the following formats: • A string of numerics that contains a decimal point but no exponent identifier. *p* represents the precision and *s* represents the scale of the decimal number that the string represents. The first character can be a plus (+) or minus (−) sign. • A string of numerics that does not contain a decimal point or an exponent identifier. The first character can be a plus (+) or minus (−) sign. The number that is represented is less than -2147483647 or greater than 2147483647.
FLOAT	480/481	A string that represents a number in scientific notation. The string consists of a series of numerics followed by an exponent identifier (an E or e followed by an optional plus (+) or minus (−) sign and a series of numerics). The string can begin with a plus (+) or minus (−) sign.

Table 4. SQL input data types and REXX data formats (continued)

SQL data type assigned by DB2	SQLTYPE for data type	REXX input data format
VARCHAR(*n*)	448/449	One of the following formats: • A string of length *n*, enclosed in single or double quotation marks. • The character X or x, followed by a string enclosed in single or double quotation marks. The string within the quotation marks has a length of 2**n* bytes and is the hexadecimal representation of a string of *n* characters. • A string of length *n* that does not have a numeric or graphic format, and does not satisfy either of the previous conditions.
VARGRAPHIC(*n*)	464/465	One of the following formats: • The character G, g, N, or n, followed by a string enclosed in single or double quotation marks. The string within the quotation marks begins with a shift-out character (X'0E') and ends with a shift-in character (X'0F'). Between the shift-out character and shift-in character are *n* double-byte characters. • The characters GX, Gx, gX, or gx, followed by a string enclosed in single or double quotation marks. The string within the quotation marks has a length of 4**n* bytes and is the hexadecimal representation of a string of *n* double-byte characters.

For example, when DB2 executes the following statements to update the MIDINIT column of the EMP table, DB2 must determine a data type for HVMIDINIT:

```
SQLSTMT="UPDATE EMP" ,
  "SET MIDINIT = ?"  ,
  "WHERE EMPNO = '000200'"
"EXECSQL PREPARE S100 FROM :SQLSTMT"
HVMIDINIT='H'
"EXECSQL EXECUTE S100 USING" ,
   ":HVMIDINIT"
```

Because the data that is assigned to HVMIDINIT has a format that fits a character data type, DB2 REXX Language Support assigns a VARCHAR type to the input data.

Ensuring that DB2 correctly interprets character input data

To ensure that DB2 REXX Language Support does not interpret character literals as graphic or numeric literals, precede and follow character literals with a double quotation mark, followed by a single quotation mark, followed by another double quotation mark ("'").

Enclosing the string in apostrophes is not adequate because REXX removes the apostrophes when it assigns a literal to a variable. For example, suppose that you want to pass the value in host variable stringvar to DB2. The value that you want to pass is the string '100'. The first thing that you need to do is to assign the string to the host variable. You might write a REXX command like this:

```
stringvar = '100'
```

After the command executes, stringvar contains the characters 100 (without the apostrophes). DB2 REXX Language Support then passes the numeric value 100 to DB2, which is not what you intended.

However, suppose that you write the command like this:

```
stringvar = "'"100"'"
```

In this case, REXX assigns the string '100' to stringvar, including the single quotation marks. DB2 REXX Language Support then passes the string '100' to DB2, which is the desired result.

Passing the data type of an input variable to DB2

In some cases, you might want to determine the data type of input data for DB2. For example, DB2 does not assign data types of SMALLINT, CHAR, or GRAPHIC to input data. If you assign or compare this data to columns of type SMALLINT, CHAR, or GRAPHIC, DB2 must do more work than if the data types of the input data and columns match.

To indicate the data type of input data to DB2, use an SQLDA. For example, suppose you want to tell DB2 that the data with which you update the MIDINIT column of the EMP table is of type CHAR, rather than VARCHAR. You need to set up an SQLDA that contains a description of a CHAR column, and then prepare and execute the UPDATE statement using that SQLDA:

```
INSQLDA.SQLD = 1                /* SQLDA contains one variable   */
INSQLDA.1.SQLTYPE = 453         /* Type of the variable is CHAR, */
                                /* and the value can be null     */
INSQLDA.1.SQLLEN  = 1           /* Length of the variable is 1   */
INSQLDA.1.SQLDATA = 'H'         /* Value in variable is H        */
INSQLDA.1.SQLIND  = 0           /* Input variable is not null    */
SQLSTMT="UPDATE EMP" ,
  "SET MIDINIT = ?"  ,
  "WHERE EMPNO = '000200'"
"EXECSQL PREPARE S100 FROM :SQLSTMT"
"EXECSQL EXECUTE S100 USING" ,
   "DESCRIPTOR :INSQLDA"
```

Retrieving data from DB2 tables

Although all output data is string data, you can determine the data type that the data represents from its format and from the data type of the column from which the data was retrieved. Table 5 gives the format for each type of output data.

Table 5. SQL output data types and REXX data formats

SQL data type	REXX output data format
SMALLINT INTEGER	A string of numerics that does not contain leading zeroes, a decimal point, or an exponent identifier. If the string represents a negative number, it begins with a minus (−) sign. The numeric value is between -2147483647 and 2147483647, inclusive.
DECIMAL(*p,s*)	A string of numerics with one of the following formats: • Contains a decimal point but not an exponent identifier. The string is padded with zeroes to match the scale of the corresponding table column. If the value represents a negative number, it begins with a minus (−) sign. • Does not contain a decimal point or an exponent identifier. The numeric value is less than -2147483647 or greater than 2147483647. If the value is negative, it begins with a minus (−) sign.
FLOAT(*n*) REAL DOUBLE	A string that represents a number in scientific notation. The string consists of a numeric, a decimal point, a series of numerics, and an exponent identifier. The exponent identifier is an E followed by a minus (−) sign and a series of numerics if the number is between -1 and 1. Otherwise, the exponent identifier is an E followed by a series of numerics. If the string represents a negative number, it begins with a minus (−) sign.

Table 5. SQL output data types and REXX data formats (continued)

SQL data type	REXX output data format

CHAR(*n*)
VARCHAR(*n*)
	A character string of length *n* bytes. The string is not enclosed in single or double quotation marks.

GRAPHIC(*n*)
VARGRAPHIC(*n*)
	A string of length 2**n* bytes. Each pair of bytes represents a double-byte character. This string does not contain a leading G, is not enclosed in quotation marks, and does not contain shift-out or shift-in characters.

Because you cannot use the SELECT INTO statement in a REXX procedure, to retrieve data from a DB2 table you must prepare a SELECT statement, open a cursor for the prepared statement, and then fetch rows into host variables or an SQLDA using the cursor. The following example demonstrates how you can retrieve data from a DB2 table using an SQLDA:

```
SQLSTMT= ,
'SELECT EMPNO, FIRSTNME, MIDINIT, LASTNAME,' ,
'  WORKDEPT, PHONENO, HIREDATE, JOB,' ,
'  EDLEVEL, SEX, BIRTHDATE, SALARY,' ,
'  BONUS, COMM' ,
'  FROM EMP'
EXECSQL DECLARE C1 CURSOR FOR S1
EXECSQL PREPARE S1 INTO :OUTSQLDA FROM :SQLSTMT
EXECSQL OPEN C1
Do Until(SQLCODE ¬= 0)
  EXECSQL FETCH C1 USING DESCRIPTOR :OUTSQLDA
  If SQLCODE = 0 Then Do
    Line = ''
    Do I = 1 To OUTSQLDA.SQLD
      Line = Line OUTSQLDA.I.SQLDATA
    End I
    Say Line
   End
End
```

Using indicator variables

When you retrieve a null value from a column, DB2 puts a negative value in an indicator variable to indicate that the data in the corresponding host variable is null. When you pass a null value to DB2, you assign a negative value to an indicator variable to indicate that the corresponding host variable has a null value.

The way that you use indicator variables for input host variables in REXX procedures is slightly different from the way that you use indicator variables in other languages. When you want to pass a null value to a DB2 column, in addition to putting a negative value in an indicator variable, you also need to put a valid value in the corresponding host variable. For example, to set a value of WORKDEPT in table EMP to null, use statements like these:

```
SQLSTMT="UPDATE EMP" ,
  "SET WORKDEPT = ? ?"
HVWORKDEPT='000'
INDWORKDEPT=-1
"EXECSQL PREPARE S100 FROM :SQLSTMT"
"EXECSQL EXECUTE S100 USING :HVWORKDEPT :INDWORKDEPT"
```

After you retrieve data from a column that can contain null values, you should always check the indicator variable that corresponds to the output host variable for that column. If the indicator variable value is negative, the retrieved value is null, so you can disregard the value in the host variable.

In the following example, the phone number for employee Haas is selected into variable HVPhone. After the SELECT statement executes, if no phone number for employee Haas is found, indicator variable INDPhone contains -1.

```
SQLSTMT = ,
"SELECT PHONENO WHERE LASTNAME='HAAS'"
"EXECSQL PREPARE S1 FROM :SQLSTMT"
"EXECSQL DECLARE C1 CURSOR FOR S1"
"EXECSQL OPEN C1"
"EXECSQL FETCH C1 INTO :HVPhone :INDPhone"
If INDPhone < 0 Then ,
  Say 'Phone number for Haas is null.'
```

Setting the isolation level of SQL statements in a REXX procedure

When you install DB2 REXX Language Support, you bind four packages for accessing DB2, each with a different isolation level:

Package name	**Isolation level**
DSNREXRR	Repeatable read (RR)
DSNREXRS	Read stability (RS)
DSNREXCS	Cursor stability (CS)
DSNREXUR	Uncommitted read (UR)

To change the isolation level for SQL statements in a REXX procedure, execute the SET CURRENT PACKAGESET statement to select the package with the isolation level you need. For example, to change the isolation level to cursor stability, execute this SQL statement:

```
"EXECSQL SET CURRENT PACKAGESET='DSNREXCS'"
```

Chapter 3. Using REXX stored procedures

This chapter contains information about defining, writing, and calling REXX stored procedures. For information that is common to all stored procedures, see Section 6 of *Application Programming and SQL Guide*.

Defining a REXX stored procedure

To define a stored procedure to DB2, you insert a row in catalog table SYSIBM.SYSPROCEDURES. The stored procedure code does not need to exist when you insert the row. Defining a REXX stored procedure is the same as defining any other stored procedure, with the following exceptions:

- To indicate that the stored procedure is written in REXX, specify REXX for the LANGUAGE column value.
- The PARMLIST string can include only the CHAR, VARCHAR, GRAPHIC, or VARGRAPHIC data types.
- A REXX stored procedure can return only one parameter. Therefore, you can specify only one entry in the PARMLIST column value as OUT or INOUT. All other entries must be of type IN. The output variable must be specified last.
- A REXX stored procedure can run only in a WLM-established address space. You therefore need to define a WLM application environment for the stored procedure and specify the name of that environment in the WLM_ENV column.
- Although DB2 REXX Language Support runs under Language Environment, the REXX stored procedure itself does not. Therefore, if you specify a value for PGM_TYPE, it is ignored. You can, however, specify Language Environment run-time options for the DB2 REXX Language Support environment in the RUNOPTS column.

Example of a definition for a REXX stored procedure: Suppose that you have written a stored procedure that has these characteristics:

- The stored procedure name is RXCHKTBL.
- User ID GROUPADM at any location can run the stored procedure.
- The REXX procedure that runs when the CALL statement executes is named RXCHKTBL.
- The parameters can have null values.
- The stored procedure is written in the REXX language.
- The stored procedure should run for no more than 900 CPU service units.
- The stored procedure should be deleted from memory when it completes.
- The Language Environment run-time option TRAP(ON) should be passed to the stored procedure.
- The stored procedure takes two parameters:
  - An input parameter that has a CHAR(8) format
  - An output parameter that has a VARCHAR(18) format
- The stored procedure returns no result sets.
- The stored procedure is part of the WLM application environment named TESTPRGS.
- The stored procedure does not access non-DB2 resources, so it does not need a special RACF environment.
- When control returns to the client program, DB2 should not commit updates automatically.

The following statement places a row that describes RXCHKTBL in the catalog table SYSPROCEDURES:

```
INSERT INTO  SYSIBM.SYSPROCEDURES
        (PROCEDURE, AUTHID, LUNAME, LOADMOD, LINKAGE, COLLID,
         LANGUAGE, ASUTIME, STAYRESIDENT, IBMREQD, RUNOPTS,
         PARMLIST,RESULT_SETS,WLM_ENV,
         EXTERNAL_SECURITY,COMMIT_ON_RETURN)
  VALUES('RXCHKTBL', 'GROUPADM', ' ', 'RXCHKTBL', 'N', ' ',
         'REXX', 900, ' ', 'N', 'TRAP(ON)',
         'IN1 CHAR(8) IN,OUT1 VARCHAR(18) OUT',0,'TESTPRGS',
         'N',NULL);
```

Writing a REXX stored procedure

A REXX stored procedure is much like any other REXX procedure and follows the same rules as stored procedures in other languages. It receives input parameters, executes REXX commands, optionally executes SQL statements, and returns at most one output parameter. A REXX stored procedure is different from other REXX procedures in the following ways:

- A REXX stored procedure cannot execute the ADDRESS DSNREXX CONNECT and ADDRESS DSNREXX DISCONNECT commands. When you execute SQL statements in your stored procedure, DB2 establishes the connection for you.
- As in other stored procedures, you cannot include the following statements in a REXX stored procedure:
  - CALL
  - COMMIT
  - CONNECT
  - RELEASE
  - SET CONNECTION
  - SET CURRENT SQLID

Figure 3 on page 20 shows an example of a REXX stored procedure that executes DB2 commands. The stored procedure performs the following actions:

- Receives one input parameter, which contains a DB2 command.
- Calls the IFI COMMAND function to execute the command.
- Extracts the command result messages from the IFI return area and places the messages in a temporary table. Each row of the temporary table contains a sequence number and the text of one message.
- Opens a cursor to return a result set that contains the command result messages.
- Returns the unformatted contents of the IFI return area in an output parameter.

Figure 2 on page 19 shows the definition of the stored procedure.

```
INSERT INTO SYSIBM.SYSPROCEDURES
( "PROCEDURE", "AUTHID", "LUNAME", "LOADMOD", "LINKAGE", "COLLID",
  "LANGUAGE", "ASUTIME", "STAYRESIDENT", "IBMREQD", "RUNOPTS",
  "PARMLIST",
  "RESULT_SETS", "WLM_ENV", "PGM_TYPE",
  "EXTERNAL_SECURITY", "COMMIT_ON_RETURN")
   VALUES ('COMMAND', 'SYSADM', '        ', 'COMMAND', ' ', ' ',
           'REXX', 0, ' ', 'N', 'TRAP(ON)',
           'VARCHAR(254) IN, VARCHAR(32704) OUT',
           1, 'WLMENV1', 'M',
           'N', 'N');
```

Figure 2. Definition for REXX stored procedure COMMAND

```
/* REXX */
PARSE UPPER ARG CMD                    /* Get the DB2 command text */
                                       /* Remove enclosing quotes  */
IF LEFT(CMD,2) = ""'" & RIGHT(CMD,2) = "'"" THEN
CMD = SUBSTR(CMD,2,LENGTH(CMD)-2)
ELSE
IF LEFT(CMD,2) = """'" & RIGHT(CMD,2) = "'""" THEN
CMD = SUBSTR(CMD,3,LENGTH(CMD)-4)
COMMAND    = SUBSTR("COMMAND",1,18," ")
  /****************************************************************/
  /*  Set up the IFCA, return area, and output area for the       */
  /*  IFI COMMAND call.                                           */
  /****************************************************************/
IFCA = SUBSTR('00'X,1,180,'00'X)
IFCA = OVERLAY(D2C(LENGTH(IFCA),2),IFCA,1+0)
IFCA = OVERLAY("IFCA",IFCA,4+1)
RTRNAREASIZE = 262144 /*1048572*/
RTRNAREA = D2C(RTRNAREASIZE+4,4)LEFT(' ',RTRNAREASIZE,' ')
OUTPUT = D2C(LENGTH(CMD)+4,2)||'0000'X||CMD
BUFFER = SUBSTR(" ",1,16," ")
  /****************************************************************/
  /*  Make the IFI COMMAND call.                                  */
  /****************************************************************/
ADDRESS LINKPGM "DSNWLIR COMMAND IFCA RTRNAREA OUTPUT"
WRC = RC
RTRN=    SUBSTR(IFCA,12+1,4)
REAS=    SUBSTR(IFCA,16+1,4)
TOTLEN = C2D(SUBSTR(IFCA,20+1,4))
  /****************************************************************/
  /*  Set up the host command environment for SQL calls.          */
  /****************************************************************/
"SUBCOM DSNREXX"                       /* Host cmd env available?  */
IF RC THEN                             /* No--add host cmd env     */
   S_RC = RXSUBCOM('ADD','DSNREXX','DSNREXX')
  /****************************************************************/
  /*  Set up SQL statements to insert command output messages     */
  /*  into a temporary table.                                     */
  /****************************************************************/
SQLSTMT='INSERT INTO SYSIBM.SYSPRINT(SEQNO,TEXT) VALUES(?,?)'
ADDRESS DSNREXX "EXECSQL DECLARE C1 CURSOR FOR S1"
IF SQLCODE ¬= 0 THEN CALL SQLCA

ADDRESS DSNREXX "EXECSQL PREPARE S1 FROM :SQLSTMT"
IF SQLCODE ¬= 0 THEN CALL SQLCA
  /****************************************************************/
  /*  Extract messages from the return area and insert them into  */
  /*  the temporary table.                                        */
  /****************************************************************/
SEQNO = 0
OFFSET = 4+1
DO WHILE ( OFFSET < TOTLEN )
  LEN = C2D(SUBSTR(RTRNAREA,OFFSET,2))
  SEQNO = SEQNO + 1
  TEXT = SUBSTR(RTRNAREA,OFFSET+4,LEN-4-1)
  ADDRESS DSNREXX "EXECSQL EXECUTE S1 USING :SEQNO,:TEXT"
  IF SQLCODE ¬= 0 THEN CALL SQLCA
  OFFSET = OFFSET + LEN
END
```

Figure 3. Example of a REXX stored procedure: COMMAND (Part 1 of 3)

```
  /****************************************************************/
  /*  Set up a cursor for a result set that contains the command  */
  /*  output messages from the temporary table.                   */
  /****************************************************************/
SQLSTMT='SELECT SEQNO,TEXT FROM SYSIBM.SYSPRINT ORDER BY SEQNO'
ADDRESS DSNREXX "EXECSQL DECLARE C2 CURSOR FOR S2"
IF SQLCODE ¬= 0 THEN CALL SQLCA

ADDRESS DSNREXX "EXECSQL PREPARE S2 FROM :SQLSTMT"
IF SQLCODE ¬= 0 THEN CALL SQLCA
  /****************************************************************/
  /*  Open the cursor to return the message output result set to  */
  /*  the caller.                                                 */
  /****************************************************************/
ADDRESS DSNREXX "EXECSQL OPEN C2"
IF SQLCODE ¬= 0 THEN CALL SQLCA

S_RC = RXSUBCOM('DELETE','DSNREXX','DSNREXX') /* REMOVE CMD ENV  */

EXIT SUBSTR(RTRNAREA,1,TOTLEN+4)
  /****************************************************************/
  /*  Routine to display the SQLCA                                */
  /****************************************************************/
SQLCA:
TRACE O
SAY 'SQLCODE ='SQLCODE
SAY 'SQLERRM ='SQLERRM
SAY 'SQLERRP ='SQLERRP
SAY 'SQLERRD ='SQLERRD.1',',
            || SQLERRD.2',',
            || SQLERRD.3',',
            || SQLERRD.4',',
            || SQLERRD.5',',
            || SQLERRD.6

SAY 'SQLWARN ='SQLWARN.0',',
            || SQLWARN.1',',
            || SQLWARN.2',',
            || SQLWARN.3',',
            || SQLWARN.4',',
            || SQLWARN.5',',
            || SQLWARN.6',',
            || SQLWARN.7',',
            || SQLWARN.8',',
            || SQLWARN.9',',
            || SQLWARN.10
SAY 'SQLSTATE='SQLSTATE
SAY 'SQLCODE ='SQLCODE
```

Figure 3. Example of a REXX stored procedure: COMMAND (Part 2 of 3)

```
EXIT 'SQLERRM ='SQLERRM';' ,
||  'SQLERRP ='SQLERRP';' ,
||  'SQLERRD ='SQLERRD.1',',
             || SQLERRD.2',',
             || SQLERRD.3',',
             || SQLERRD.4',',
             || SQLERRD.5',',
             || SQLERRD.6';' ,

||  'SQLWARN ='SQLWARN.0',',
             || SQLWARN.1',',
             || SQLWARN.2',',
             || SQLWARN.3',',
             || SQLWARN.4',',
             || SQLWARN.5',',
             || SQLWARN.6',',
             || SQLWARN.7',',
             || SQLWARN.8',',
             || SQLWARN.9',',
             || SQLWARN.10';' ,
||  'SQLSTATE='SQLSTATE';'
```

Figure 3. Example of a REXX stored procedure: COMMAND (Part 3 of 3)

Calling a stored procedure from a REXX Procedure

Use the SQL CALL statement in a REXX program to call a stored procedure in any supported language. The format of the parameters that you pass in the CALL statement must be compatible with the data types in the PARMLIST column value of the SYSPROCEDURES row that defines the stored procedure. Table 6 lists each SQL data type that you can specify in the PARMLIST column and the corresponding format for a REXX parameter that represents that data type.

Table 6. Parameter formats for a CALL statement in a REXX procedure

SQL data type	REXX format

CHARACTER(*n*)
VARCHAR(*n*)
VARCHAR(*n*) FOR BIT DATA
	A string of length *n*, enclosed in single quotation marks.

GRAPHIC(*n*)
VARGRAPHIC(*n*)
	The character G followed by a string enclosed in single quotation marks. The string within the quotation marks begins with a shift-out character (X'0E') and ends with a shift-in character (X'0F'). Between the shift-out character and shift-in character are *n* double-byte characters.

Figure 4 on page 23 demonstrates how a REXX procedure calls the stored procedure in Figure 3 on page 20. The REXX procedure performs the following actions:

- Connects to the DB2 subsystem that was specified by the REXX procedure invoker.
- Calls the stored procedure to execute a DB2 command that was specified by the REXX procedure invoker.
- Retrieves rows from a result set that contains the command output messages.

```
/* REXX */
PARSE ARG SSID COMMAND                  /* Get the SSID to connect to */
                                        /* and the DB2 command to be  */
                                        /* executed                   */
  /****************************************************************/
  /*  Set up the host command environment for SQL calls.          */
  /****************************************************************/
"SUBCOM DSNREXX"                        /* Host cmd env available? */
IF RC THEN                              /* No--make one               */
   S_RC = RXSUBCOM('ADD','DSNREXX','DSNREXX')
  /****************************************************************/
  /*  Connect to the DB2 subsystem.                               */
  /****************************************************************/
ADDRESS DSNREXX "CONNECT" SSID

IF SQLCODE ¬= 0 THEN CALL SQLCA

PROC = 'COMMAND'

RESULTSIZE = 32703
RESULT = LEFT(' ',RESULTSIZE,' ')
  /****************************************************************/
  /*  Call the stored procedure that executes the DB2 command.    */
  /*  The input variable (COMMAND) contains the DB2 command.      */
  /*  The output variable (RESULT) will contain the return area   */
  /*  from the IFI COMMAND call after the stored procedure        */
  /*  executes.                                                   */
  /****************************************************************/
ADDRESS DSNREXX "EXECSQL" ,
"CALL" PROC "(:COMMAND, :RESULT)"
IF SQLCODE < 0 THEN CALL SQLCA

SAY 'RETCODE ='RETCODE
SAY 'SQLCODE ='SQLCODE
SAY 'SQLERRM ='SQLERRM
SAY 'SQLERRP ='SQLERRP
SAY 'SQLERRD ='SQLERRD.1',',
            || SQLERRD.2',',
            || SQLERRD.3',',
            || SQLERRD.4',',
            || SQLERRD.5',',
            || SQLERRD.6

SAY 'SQLWARN ='SQLWARN.0',',
            || SQLWARN.1',',
            || SQLWARN.2',',
            || SQLWARN.3',',
            || SQLWARN.4',',
            || SQLWARN.5',',
            || SQLWARN.6',',
            || SQLWARN.7',',
            || SQLWARN.8',',
            || SQLWARN.9',',
            || SQLWARN.10
SAY 'SQLSTATE='SQLSTATE
SAY C2X(RESULT) "'"||RESULT||"'"
```

Figure 4. Example of a REXX procedure that calls a stored procedure (Part 1 of 3)

```
  /*****************************************************************/
  /*  Display the IFI return area in hexadecimal.                   */
  /*****************************************************************/
OFFSET = 4+1
TOTLEN = LENGTH(RESULT)
DO WHILE ( OFFSET <  TOTLEN )
  LEN = C2D(SUBSTR(RESULT,OFFSET,2))
  SAY SUBSTR(RESULT,OFFSET+4,LEN-4-1)
  OFFSET = OFFSET + LEN
END
  /*****************************************************************/
  /*  Get information about result sets returned by the            */
  /*  stored procedure.                                             */
  /*****************************************************************/
ADDRESS DSNREXX "EXECSQL DESCRIBE PROCEDURE :PROC INTO :SQLDA"
IF SQLCODE ¬= 0 THEN CALL SQLCA

DO I = 1 TO SQLDA.SQLD
  SAY "SQLDA."I".SQLNAME    ="SQLDA.I.SQLNAME";"
  SAY "SQLDA."I".SQLTYPE    ="SQLDA.I.SQLTYPE";"
  SAY "SQLDA."I".SQLLOCATOR ="SQLDA.I.SQLLOCATOR";"
  SAY "SQLDA."I".SQLESTIMATE="SQLDA.I.SQLESTIMATE";"
END I
  /*****************************************************************/
  /*  Set up a cursor to retrieve the rows from the result          */
  /*  set.                                                          */
  /*****************************************************************/
ADDRESS DSNREXX "EXECSQL ASSOCIATE LOCATOR (:RESULT) WITH PROCEDURE :PROC"
IF SQLCODE ¬= 0 THEN CALL SQLCA
SAY RESULT

ADDRESS DSNREXX "EXECSQL ALLOCATE C101 CURSOR FOR RESULT SET :RESULT"
IF SQLCODE ¬= 0 THEN CALL SQLCA

CURSOR = 'C101'
ADDRESS DSNREXX "EXECSQL DESCRIBE CURSOR :CURSOR INTO :SQLDA"
IF SQLCODE ¬= 0 THEN CALL SQLCA
  /*****************************************************************/
  /*  Retrieve and display the rows from the result set, which     */
  /*  contain the command output message text.                     */
  /*****************************************************************/
DO UNTIL(SQLCODE ¬= 0)
  ADDRESS DSNREXX "EXECSQL FETCH C101 INTO :SEQNO, :TEXT"
  IF SQLCODE = 0 THEN
    DO
      SAY TEXT
    END
END
IF SQLCODE ¬= 0 THEN CALL SQLCA

ADDRESS DSNREXX "EXECSQL CLOSE C101"
IF SQLCODE ¬= 0 THEN CALL SQLCA

ADDRESS DSNREXX "EXECSQL COMMIT"
IF SQLCODE ¬= 0 THEN CALL SQLCA
```

Figure 4. Example of a REXX procedure that calls a stored procedure (Part 2 of 3)

```
  /****************************************************************/
  /*  Disconnect from the DB2 subsystem.                          */
  /****************************************************************/
ADDRESS DSNREXX "DISCONNECT"
IF SQLCODE ¬= 0 THEN CALL SQLCA
  /****************************************************************/
  /*  Delete the host command environment for SQL.                */
  /****************************************************************/
S_RC = RXSUBCOM('DELETE','DSNREXX','DSNREXX') /* REMOVE CMD ENV  */

RETURN
  /****************************************************************/
  /*  Routine to display the SQLCA                                */
  /****************************************************************/
SQLCA:
TRACE O
SAY 'SQLCODE ='SQLCODE
SAY 'SQLERRM ='SQLERRM
SAY 'SQLERRP ='SQLERRP
SAY 'SQLERRD ='SQLERRD.1',',
            || SQLERRD.2',',
            || SQLERRD.3',',
            || SQLERRD.4',',
            || SQLERRD.5',',
            || SQLERRD.6

SAY 'SQLWARN ='SQLWARN.0',',
            || SQLWARN.1',',
            || SQLWARN.2',',
            || SQLWARN.3',',
            || SQLWARN.4',',
            || SQLWARN.5',',
            || SQLWARN.6',',
            || SQLWARN.7',',
            || SQLWARN.8',',
            || SQLWARN.9',',
            || SQLWARN.10
SAY 'SQLSTATE='SQLSTATE
EXIT
```

Figure 4. Example of a REXX procedure that calls a stored procedure (Part 3 of 3)

Chapter 4. Installing DB2 REXX Language Support

This section describes the steps that you must perform to install DB2 REXX Language Support. For additional information, see *IBM DATABASE 2 Program Directory*.

Step 1: Copy and edit the SMP/E jobs

Use the sample JCL shown in Figure 5 on page 26 to invoke the MVS utility IEBCOPY to copy the SMP/E jobs to DASD.

```
//* COMPID:  DB2,5740XYR00
//* DOC:     LOAD REXX SMP INSTALLATION JCL FROM TAPE FOR DB2
//LOAD    EXEC PGM=IEBCOPY
//SYSPRINT DD SYSOUT=*
//JCLTAPE  DD DSN=IBM.JDB551H.F1,VOL=(PRIVATE,,SER=DB551H),
//           UNIT=TAPE,LABEL=(2,SL),DISP=(OLD,PASS)
//*
//JCLDISK DD DSN=SYSADM.JCL.CNTL,VOL=SER=USER01,UNIT=SYSDA,
//           DISP=OLD
//SYSUT3   DD UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT4   DD UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSIN    DD *
  COPY I=JCLTAPE,O=JCLDISK
  SELECT MEMBER=(DSNTTJAC,DSNTTJAP,DSNTTJRC)
//*
```

Figure 5. Sample JCL to Copy SMP/E jobs to DASD

After you have copied the SMP/E jobs to DASD, add a job statement to each job and customize the jobs to specify the unit names and volume serial numbers that your site uses.

The SMP/E jobs move all files for DB2 REXX Language Support to the target and distribution libraries for the DB2 base product. Therefore, you do not need to set up target and distribution libraries for DB2 REXX Language Support.

Step 2: Run the receive job: DSNTTJRC

DSNTTJRC invokes SMP/E to receive the FMIDs for DB2 REXX Language Support into the SMP/E control data sets.

Step 3: Run the apply job: DSNTTJAP

DSNTTJAP invokes SMP/E to apply the FMIDs for DB2 REXX Language Support to the DB2 target libraries.

Step 4: Run the Accept Job: DSNTTJAC

DSNTTJAC invokes SMP/E to accept the FMIDs for DB2 REXX Language Support into the DB2 distribution libraries.

Step 4: Bind the DB2 REXX Language Support packages: DSNTIJRX

DSNTIJRX, which is in data set DSN510.SDSNSAMP, binds DB2 packages that are used by DB2 REXX Language Support. Before you run DSNTIJRX, make the following changes:

- Add a job statement.
- Change `DSN SYSTEM(DSN)` to `DSN SYSTEM(`*`ssid`*`)`, where *ssid* is the name of the DB2 subsystem on which you will use DB2 REXX Language Support.
- Change all instances of DSN!!0 to your DB2 data set name prefix.
- Change all instances of DSNTIA!! to the plan name for the DSNTIAD program.

Chapter 5. Running a DB2 REXX application

You run DB2 REXX procedures under TSO. You do not precompile, compile, link-edit or bind DB2 REXX procedures before you run them.

In a batch environment, you might use statements like these to invoke procedure REXXPROG:

```
//RUNREXX EXEC PGM=IKJEFT01,DYNAMNBR=20
//SYSEXEC  DD DISP=SHR,DSN=SYSADM.REXX.EXEC
//SYSTSPRT DD SYSOUT=*
//SYSTSIN  DD *
%REXXPROG parameters
```

The SYSEXEC data set contains your REXX application, and the SYSTSIN data set contains the command that you use to invoke the application.

Appendix. Sample DB2 REXX application

This appendix contains an example of a complete DB2 REXX application named DRAW. DRAW must be invoked from the command line of an ISPF edit session. DRAW takes a table or view name as input and produces a SELECT, INSERT, or UPDATE SQL statement or a LOAD utility control statement that includes the columns of the table as output.

DRAW syntax:

```
►►──%DRAW──object-name──(──┬───────────┬──┬──────────────────────┬──────────────────────────────►◄
                           └─SSID=ssid─┘  └─TYPE=──┬─SELECT─┬─┘
                                                   ├─INSERT─┤
                                                   ├─UPDATE─┤
                                                   └─LOAD───┘
```

DRAW parameters:

object-name
: The name of the table or view for which DRAW builds an SQL statement or utility control statement. The name can be a one-, two-, or three-part name. The table or view to which *object-name* refers must exist before DRAW can run.

  object-name is a required parameter.

SSID=*ssid*
: Specifies the name of the local DB2 subsystem.

  S can be used as an abbreviation for SSID.

  If you invoke DRAW from the command line of the edit session in SPUFI, SSID=*ssid* is an optional parameter. DRAW uses the subsystem ID from the DB2I Defaults panel.

TYPE=*operation-type*
: The type of statement that DRAW builds.

  T can be used as an abbreviation for TYPE.

  operation-type has one of the following values:

SELECT	Builds a SELECT statement in which the result table contains all columns of *object-name*. S can be used as an abbreviation for SELECT.
INSERT	Builds a template for an INSERT statement that inserts values into all columns of *object-name*. The template contains comments that indicate where the user can place column values. I can be used as an abbreviation for INSERT.
UPDATE	Builds a template for an UPDATE statement that updates columns of *object-name*. The template contains comments that indicate where the user can place column values and qualify the update operation for selected rows. U can be used as an abbreviation for UPDATE.

LOAD	Builds a template for a LOAD utility control statement for *object-name*. L can be used as an abbreviation for LOAD.

TYPE=operation-type is an optional parameter. The default is TYPE=SELECT.

DRAW data sets:

Edit data set
The data set from which you issue the DRAW command when you are in an ISPF edit session. If you issue the DRAW command from a SPUFI session, this data set is the data set that you specify in field 1 of the main SPUFI panel (DSNESP01). The output from the DRAW command goes into this data set.

DRAW return codes:

Return code	Meaning
0	Successful completion.
12	An error occurred when DRAW edited the input file.
20	One of the following errors occurred: • No input parameters were specified. • One of the input parameters was not valid. • An SQL error occurred when the output statement was generated.

Examples of DRAW invocation:

Generate a SELECT statement for table DSN8510.EMP at the local subsystem. Use the default DB2I subsystem ID.

The DRAW invocation is:

```
DRAW DSN8510.EMP (TYPE=SELECT
```

The output is:

```
SELECT "EMPNO" , "FIRSTNME" , "MIDINIT" , "LASTNAME" , "WORKDEPT" ,
  "PHONENO" , "HIREDATE" , "JOB" , "EDLEVEL" , "SEX" , "BIRTHDATE" ,
  "SALARY" , "BONUS" , "COMM"
FROM DSN8510.EMP
```

Generate a template for an INSERT statement that inserts values into table DSN8510.EMP at location SAN_JOSE. The local subsystem ID is DSN.

The DRAW invocation is:

```
DRAW SAN_JOSE.DSN8510.EMP (TYPE=INSERT SSID=DSN
```

The output is:

```
INSERT INTO SAN_JOSE.DSN8510.EMP  ( "EMPNO" , "FIRSTNME" , "MIDINIT" ,
  "LASTNAME" ,  "WORKDEPT" , "PHONENO" , "HIREDATE" , "JOB" ,
  "EDLEVEL" , "SEX" , "BIRTHDATE" , "SALARY" , "BONUS" , "COMM" )
   VALUES (
-- ENTER VALUES BELOW         COLUMN NAME        DATA TYPE
                          , -- EMPNO             CHAR(6) NOT NULL
                          , -- FIRSTNME          VARCHAR(12) NOT NULL
                          , -- MIDINIT           CHAR(1) NOT NULL
                          , -- LASTNAME          VARCHAR(15) NOT NULL
```

```
                         , -- WORKDEPT          CHAR(3)
                         , -- PHONENO           CHAR(4)
                         , -- HIREDATE          DATE
                         , -- JOB               CHAR(8)
                         , -- EDLEVEL           SMALLINT
                         , -- SEX               CHAR(1)
                         , -- BIRTHDATE         DATE
                         , -- SALARY            DECIMAL(9,2)
                         , -- BONUS             DECIMAL(9,2)
                         ) -- COMM              DECIMAL(9,2)
```

Generate a template for an UPDATE statement that updates values of table DSN8510.EMP. The local subsystem ID is DSN.

The DRAW invocation is:

```
DRAW DSN8510.EMP (TYPE=UPDATE SSID=DSN
```

The output is:

```
UPDATE DSN8510.EMP  SET
-- COLUMN NAME          ENTER VALUES BELOW      DATA TYPE
   "EMPNO"=                                  -- CHAR(6) NOT NULL
 , "FIRSTNME"=                               -- VARCHAR(12) NOT NULL
 , "MIDINIT"=                                -- CHAR(1) NOT NULL
 , "LASTNAME"=                               -- VARCHAR(15) NOT NULL
 , "WORKDEPT"=                               -- CHAR(3)
 , "PHONENO"=                                -- CHAR(4)
 , "HIREDATE"=                               -- DATE
 , "JOB"=                                    -- CHAR(8)
 , "EDLEVEL"=                                -- SMALLINT
 , "SEX"=                                    -- CHAR(1)
 , "BIRTHDATE"=                              -- DATE
 , "SALARY"=                                 -- DECIMAL(9,2)
 , "BONUS"=                                  -- DECIMAL(9,2)
 , "COMM"=                                   -- DECIMAL(9,2)
WHERE
```

Generate a LOAD control statement to load values into table DSN8510.EMP. The local subsystem ID is DSN.

The draw invocation is:

```
DRAW DSN8510.EMP (TYPE=LOAD SSID=DSN
```

The output is:

```
LOAD DATA INDDN SYSREC INTO TABLE DSN8510.EMP
 ( "EMPNO"              POSITION(    1) CHAR(6)
 , "FIRSTNME"           POSITION(    8) VARCHAR
 , "MIDINIT"            POSITION(   21) CHAR(1)
 , "LASTNAME"           POSITION(   23) VARCHAR
 , "WORKDEPT"           POSITION(   39) CHAR(3)
                          NULLIF(   39)='?'
 , "PHONENO"            POSITION(   43) CHAR(4)
                          NULLIF(   43)='?'
 , "HIREDATE"           POSITION(   48) DATE EXTERNAL
                          NULLIF(   48)='?'
 , "JOB"                POSITION(   59) CHAR(8)
                          NULLIF(   59)='?'
 , "EDLEVEL"            POSITION(   68) SMALLINT
                          NULLIF(   68)='?'
 , "SEX"                POSITION(   71) CHAR(1)
                          NULLIF(   71)='?'
 , "BIRTHDATE"          POSITION(   73) DATE EXTERNAL
                          NULLIF(   73)='?'
 , "SALARY"             POSITION(   84) DECIMAL EXTERNAL(9,2)
```

```
                          NULLIF(   84)='?'
 , "BONUS"              POSITION(   90) DECIMAL EXTERNAL(9,2)
                          NULLIF(   90)='?'
 , "COMM"               POSITION(   96) DECIMAL EXTERNAL(9,2)
                          NULLIF(   96)='?'
 )
```

DRAW source code:

```
/* REXX ***************************************************************/
L1 = WHEREAMI()
/*
DRAW creates basic SQL queries by retrieving the description of a
table.  You must specify the name of the table or view to be queried.
You can specify the type of query you want to compose.  You might need
to specify the name of the DB2 subsystem.

>>--DRAW-----tablename-----|---------------------------|-------><
                           |-(-|-Ssid=subsystem-name-|-|
                               |       +-Select-+    |
                               |-Type=-|-Insert-|----|
                                       |-Update-|
                                       +--Load--+

Ssid=subsystem-name
    subsystem-name specified the name of a DB2 subsystem.

Select
    Composes a basic query for selecting data from the columns of a
    table or view.  If TYPE is not specified, SELECT is assumed.
    Using SELECT with the DRAW command produces a query that would
    retrieve all rows and all columns from the specified table.  You
    can then modify the query as needed.

    A SELECT query of EMP composed by DRAW looks like this:

SELECT "EMPNO" , "FIRSTNME" , "MIDINIT" , "LASTNAME" , "WORKDEPT" ,
       "PHONENO" , "HIREDATE" , "JOB" , "EDLEVEL" , "SEX" , "BIRTHDATE" ,
       "SALARY" , "BONUS" , "COMM"
FROM DSN8510.EMP

            If you include a location qualifier, the query looks like this:

SELECT "EMPNO" , "FIRSTNME" , "MIDINIT" , "LASTNAME" , "WORKDEPT" ,
       "PHONENO" , "HIREDATE" , "JOB" , "EDLEVEL" , "SEX" , "BIRTHDATE" ,
       "SALARY" , "BONUS" , "COMM"
FROM STLEC1.DSN8510.EMP
```

Figure 6. REXX sample program DRAW (Part 1 of 10)

```
    To use this SELECT query, type the other clauses you need.  If
    you are selecting from more than one table, use a DRAW command
    for each table name you want represented.

Insert
    Composes a basic query to insert data into the columns of a table
    or view.

    The following example shows an INSERT query of EMP that
    DRAW composed:

INSERT INTO DSN8510.EMP  ( "EMPNO" , "FIRSTNME" , "MIDINIT" , "LASTNAME" ,
       "WORKDEPT" , "PHONENO" , "HIREDATE" , "JOB" , "EDLEVEL" , "SEX" ,
       "BIRTHDATE" , "SALARY" , "BONUS" , "COMM" )
   VALUES (
-- ENTER VALUES BELOW         COLUMN NAME        DATA TYPE
                         , -- EMPNO              CHAR(6) NOT NULL
                         , -- FIRSTNME           VARCHAR(12) NOT NULL
                         , -- MIDINIT            CHAR(1) NOT NULL
                         , -- LASTNAME           VARCHAR(15) NOT NULL
                         , -- WORKDEPT           CHAR(3)
                         , -- PHONENO            CHAR(4)
                         , -- HIREDATE           DATE
                         , -- JOB                CHAR(8)
                         , -- EDLEVEL            SMALLINT
                         , -- SEX                CHAR(1)
                         , -- BIRTHDATE          DATE
                         , -- SALARY             DECIMAL(9,2)
                         , -- BONUS              DECIMAL(9,2)
                         ) -- COMM               DECIMAL(9,2)

    To insert values into EMP, type values to the left of the
    column names.  See SQL Reference for more information on
    INSERT queries.

Update
    Composes a basic query to change the data in a table or view.

    The following example shows an UPDATE query of EMP composed
    by DRAW:
```

Figure 6. REXX sample program DRAW (Part 2 of 10)

```
UPDATE DSN8510.EMP  SET
-- COLUMN NAME           ENTER VALUES BELOW      DATA TYPE
   "EMPNO"=                                   -- CHAR(6) NOT NULL
 , "FIRSTNME"=                                -- VARCHAR(12) NOT NULL
 , "MIDINIT"=                                 -- CHAR(1) NOT NULL
 , "LASTNAME"=                                -- VARCHAR(15) NOT NULL
 , "WORKDEPT"=                                -- CHAR(3)
 , "PHONENO"=                                 -- CHAR(4)
 , "HIREDATE"=                                -- DATE
 , "JOB"=                                     -- CHAR(8)
 , "EDLEVEL"=                                 -- SMALLINT
 , "SEX"=                                     -- CHAR(1)
 , "BIRTHDATE"=                               -- DATE
 , "SALARY"=                                  -- DECIMAL(9,2)
 , "BONUS"=                                   -- DECIMAL(9,2)
 , "COMM"=                                    -- DECIMAL(9,2)
WHERE

    To use this UPDATE query, type the changes you want to make to
    the right of the column names, and delete the lines you don't
    need.  Be sure to complete the WHERE clause.  For information on
    writing queries to update data, refer to SQL Reference.

Load
    Composes a load statement to load the data in a table.

    The following example shows a LOAD statement of EMP composed
    by DRAW:

LOAD DATA INDDN SYSREC INTO TABLE DSN8510.EMP
 ( "EMPNO"              POSITION(    1) CHAR(6)
 , "FIRSTNME"           POSITION(    8) VARCHAR
 , "MIDINIT"            POSITION(   21) CHAR(1)
 , "LASTNAME"           POSITION(   23) VARCHAR
 , "WORKDEPT"           POSITION(   39) CHAR(3)
                          NULLIF(   39)='?'
 , "PHONENO"            POSITION(   43) CHAR(4)
                          NULLIF(   43)='?'
 , "HIREDATE"           POSITION(   48) DATE EXTERNAL
                          NULLIF(   48)='?'
 , "JOB"                POSITION(   59) CHAR(8)
                          NULLIF(   59)='?'
 , "EDLEVEL"            POSITION(   68) SMALLINT
                          NULLIF(   68)='?'
 , "SEX"                POSITION(   71) CHAR(1)
                          NULLIF(   71)='?'
 , "BIRTHDATE"          POSITION(   73) DATE EXTERNAL
                          NULLIF(   73)='?'
 , "SALARY"             POSITION(   84) DECIMAL EXTERNAL(9,2)
                          NULLIF(   84)='?'
 , "BONUS"              POSITION(   90) DECIMAL EXTERNAL(9,2)
                          NULLIF(   90)='?'
 , "COMM"               POSITION(   96) DECIMAL EXTERNAL(9,2)
                          NULLIF(   96)='?'
 )
```

Figure 6. REXX sample program DRAW (Part 3 of 10)

```
    To use this LOAD statement, type the changes you want to make,
    and delete the lines you don't need.  For information on writing
    queries to update data, refer to
    Utility Guide and Reference.

*/
   L2 = WHEREAMI()
/*********************************************************************/
/* TRACE ?R                                                          */
/*********************************************************************/
Address ISPEXEC
"ISREDIT MACRO (ARGS) NOPROCESS"
If ARGS = "" Then
Do
  Do I = L1+2 To L2-2;Say SourceLine(I);End
  Exit (20)
End
Parse Upper Var Args Table "(" Parms
Parms = Translate(Parms," ",",")
Type = "SELECT" /* Default */
SSID = ""       /* Default */
"VGET (DSNEOV01)"
If RC = 0 Then SSID = DSNEOV01
If (Parms <> "") Then
Do Until(Parms = "")
Parse Var Parms Var "=" Value Parms
  If Var = "T" | Var = "TYPE" Then Type = Value
  Else
  If Var = "S" | Var = "SSID" Then SSID = Value
  Else
    Exit (20)
End
"CONTROL ERRORS RETURN"
"ISREDIT (LEFTBND,RIGHTBND) = BOUNDS"
"ISREDIT (LRECL) = DATA_WIDTH" /*LRECL*/
BndSize = RightBnd - LeftBnd + 1
If BndSize > 72 Then BndSize = 72
"ISREDIT PROCESS DEST"
Select
  When rc = 0 Then
    'ISREDIT (ZDEST) = LINENUM .ZDEST'
  When rc <= 8 Then /* No A or B entered */
    Do
       zedsmsg = 'Enter "A"/"B" line cmd'
       zedlmsg = 'DRAW requires an "A" or "B" line command'
       'SETMSG MSG(ISRZ001)'
       Exit 12
    End
  When rc < 20 Then /* Conflicting line commands - edit sets message */
    Exit 12
  When rc = 20 Then
    zdest = 0
  Otherwise
    Exit 12
End
```

Figure 6. REXX sample program DRAW (Part 4 of 10)

```
SQLTYPE. = "UNKNOWN TYPE"
VCHTYPE  = 448; SQLTYPES.VCHTYPE  = 'VARCHAR'
CHTYPE   = 452; SQLTYPES.CHTYPE   = 'CHAR'
LVCHTYPE = 456; SQLTYPES.LVCHTYPE = 'VARCHAR'
VGRTYP   = 464; SQLTYPES.VGRTYP   = 'VARGRAPHIC'
GRTYP    = 468; SQLTYPES.GRTYP    = 'GRAPHIC'
LVGRTYP  = 472; SQLTYPES.LVGRTYP  = 'VARGRAPHIC'
FLOTYPE  = 480; SQLTYPES.FLOTYPE  = 'FLOAT'
DCTYPE   = 484; SQLTYPES.DCTYPE   = 'DECIMAL'
INTYPE   = 496; SQLTYPES.INTYPE   = 'INTEGER'
SMTYPE   = 500; SQLTYPES.SMTYPE   = 'SMALLINT'
DATYPE   = 384; SQLTYPES.DATYPE   = 'DATE'
TITYPE   = 388; SQLTYPES.TITYPE   = 'TIME'
TSTYPE   = 392; SQLTYPES.TSTYPE   = 'TIMESTAMP'

Address TSO "SUBCOM DSNREXX"           /* HOST CMD ENV AVAILABLE? */

IF RC THEN                             /* NO, LET'S MAKE ONE      */
  S_RC = RXSUBCOM('ADD','DSNREXX','DSNREXX') /* ADD HOST CMD ENV  */

Address DSNREXX "CONNECT" SSID
If SQLCODE ˆ= 0 Then Call SQLCA
Address DSNREXX "EXECSQL DESCRIBE TABLE :TABLE INTO :SQLDA"

If SQLCODE ˆ= 0 Then Call SQLCA
Address DSNREXX "EXECSQL COMMIT"
Address DSNREXX "DISCONNECT"
If SQLCODE ˆ= 0 Then Call SQLCA

Select
  When (Left(Type,1) = "S") Then
    Call DrawSelect
  When (Left(Type,1) = "I") Then
    Call DrawInsert
  When (Left(Type,1) = "U") Then
    Call DrawUpdate
  When (Left(Type,1) = "L") Then
    Call DrawLoad
  Otherwise EXIT (20)
End

Do I = LINE.0 To 1 By -1
  LINE = COPIES(" ",LEFTBND-1)||LINE.I
  'ISREDIT LINE_AFTER 'zdest' = DATALINE (Line)'
End
line1 = zdest + 1
'ISREDIT CURSOR = 'line1 0
Exit
```

Figure 6. REXX sample program DRAW (Part 5 of 10)

```
/*********************************************************************/
WHEREAMI:; RETURN SIGL
/*********************************************************************/
/* Draw SELECT                                                       */
/*********************************************************************/
DrawSelect:
  Line.0 = 0
  Line = "SELECT"
  Do I = 1 To SQLDA.SQLD
    If I > 1 Then Line = Line ','
    ColName = '"'SQLDA.I.SQLNAME'"'
    Null = SQLDA.I.SQLTYPE//2
    If Length(Line)+Length(ColName)+LENGTH(" ,") > BndSize THEN
    Do
      L = Line.0 + 1; Line.0 = L
      Line.L = Line
      Line = "      "
    End
    Line = Line ColName
  End I
  If Line ˆ= "" Then
  Do
    L = Line.0 + 1; Line.0 = L
    Line.L = Line
    Line = "      "
  End
  L = Line.0 + 1; Line.0 = L
  Line.L = "FROM" TABLE
  Return
/*********************************************************************/
/* Draw INSERT                                                       */
/*********************************************************************/
DrawInsert:
  Line.0 = 0
  Line = "INSERT INTO" TABLE "("
  Do I = 1 To SQLDA.SQLD
    If I > 1 Then Line = Line ','
    ColName = '"'SQLDA.I.SQLNAME'"'
    If Length(Line)+Length(ColName) > BndSize THEN
    Do
      L = Line.0 + 1; Line.0 = L
      Line.L = Line
      Line = "      "
    End
    Line = Line ColName
    If I = SQLDA.SQLD Then Line = Line ')'
  End I
  If Line ˆ= "" Then
  Do
    L = Line.0 + 1; Line.0 = L
    Line.L = Line
    Line = "      "
  End
```

Figure 6. REXX sample program DRAW (Part 6 of 10)

```
L = Line.0 + 1; Line.0 = L
Line.L = "   VALUES ("
L = Line.0 + 1; Line.0 = L
Line.L = ,
"-- ENTER VALUES BELOW         COLUMN NAME        DATA TYPE"
Do I = 1 To SQLDA.SQLD
  If SQLDA.SQLD > 1 & I < SQLDA.SQLD Then
    Line = "                         , --"
  Else
    Line = "                         ) --"
  Line = Line Left(SQLDA.I.SQLNAME,18)
  Type  = SQLDA.I.SQLTYPE
  Null  = Type//2
  If Null Then Type = Type - 1
  Len   = SQLDA.I.SQLLEN
  Prcsn = SQLDA.I.SQLLEN.SQLPRECISION
  Scale = SQLDA.I.SQLLEN.SQLSCALE
  Select
  When (Type = CHTYPE   ,
       |Type = VCHTYPE  ,
       |Type = LVCHTYPE ,
       |Type = GRTYP    ,
       |Type = VGRTYP   ,
       |Type = LVGRTYP  ) THEN
    Type = SQLTYPES.Type"("STRIP(LEN)")"
  When (Type = FLOTYPE  ) THEN
    Type = SQLTYPES.Type"("STRIP((LEN*4)-11) ")"
  When (Type = DCTYPE   ) THEN
    Type = SQLTYPES.Type"("STRIP(PRCSN)","STRIP(SCALE)")"
  Otherwise
    Type = SQLTYPES.Type
  End
  Line = Line Type
  If Null = 0 Then
  Line = Line "NOT NULL"
  L = Line.0 + 1; Line.0 = L
  Line.L = Line
End I
Return
```

Figure 6. REXX sample program DRAW (Part 7 of 10)

```
/*********************************************************************/
/* Draw UPDATE                                                       */
/*********************************************************************/
DrawUpdate:
  Line.0 = 1
  Line.1 = "UPDATE" TABLE "SET"
  L = Line.0 + 1; Line.0 = L
  Line.L = ,
  "-- COLUMN NAME           ENTER VALUES BELOW      DATA TYPE"
  Do I = 1 To SQLDA.SQLD
    If I = 1 Then
      Line = "  "
    Else
      Line = " ,"
    Line = Line Left('"'SQLDA.I.SQLNAME'"=',21)
    Line = Line Left(" ",20)
    Type  = SQLDA.I.SQLTYPE
    Null  = Type//2
    If Null Then Type = Type - 1
    Len   = SQLDA.I.SQLLEN
    Prcsn = SQLDA.I.SQLLEN.SQLPRECISION
    Scale = SQLDA.I.SQLLEN.SQLSCALE
    Select
    When (Type = CHTYPE   ,
         |Type = VCHTYPE  ,
         |Type = LVCHTYPE ,
         |Type = GRTYP    ,
         |Type = VGRTYP   ,
         |Type = LVGRTYP  ) THEN
      Type = SQLTYPES.Type"("STRIP(LEN)")"
    When (Type = FLOTYPE  ) THEN
      Type = SQLTYPES.Type"("STRIP((LEN*4)-11) ")"
    When (Type = DCTYPE   ) THEN
      Type = SQLTYPES.Type"("STRIP(PRCSN)","STRIP(SCALE)")"
    Otherwise
      Type = SQLTYPES.Type
    End
    Line = Line "--" Type
    If Null = 0 Then
    Line = Line "NOT NULL"
    L = Line.0 + 1; Line.0 = L
    Line.L = Line
  End I
  L = Line.0 + 1; Line.0 = L
  Line.L = "WHERE"
  Return
```

Figure 6. REXX sample program DRAW (Part 8 of 10)

```
/*********************************************************************/
/* Draw LOAD                                                         */
/*********************************************************************/
DrawLoad:
  Line.0 = 1
  Line.1 = "LOAD DATA INDDN SYSREC INTO TABLE" TABLE
  Position = 1
  Do I = 1 To SQLDA.SQLD
    If I = 1 Then
      Line = " ("
    Else
      Line = " ,"
    Line = Line Left('"'SQLDA.I.SQLNAME'"',20)
    Line = Line "POSITION("RIGHT(POSITION,5)")"
    Type  = SQLDA.I.SQLTYPE
    Null  = Type//2
    If Null Then Type = Type - 1
    Len   = SQLDA.I.SQLLEN
    Prcsn = SQLDA.I.SQLLEN.SQLPRECISION
    Scale = SQLDA.I.SQLLEN.SQLSCALE
    Select
    When (Type = CHTYPE   ,
         |Type = GRTYP    ) THEN
      Type = SQLTYPES.Type"("STRIP(LEN)")"
    When (Type = FLOTYPE  ) THEN
      Type = SQLTYPES.Type"("STRIP((LEN*4)-11) ")"
    When (Type = DCTYPE   ) THEN
    Do
      Type = SQLTYPES.Type "EXTERNAL"
      Type = Type"("STRIP(PRCSN)","STRIP(SCALE)")"
      Len  = (PRCSN+2)%2
    End
    When (Type = DATYPE   ,
         |Type = TITYPE   ,
         |Type = TSTYPE   ) THEN
      Type = SQLTYPES.Type "EXTERNAL"
    Otherwise
      Type = SQLTYPES.Type
    End
    If   (Type = GRTYP    ,
         |Type = VGRTYP   ,
         |Type = LVGRTYP  ) THEN
      Len = Len * 2
    If   (Type = VCHTYPE  ,
         |Type = LVCHTYPE ,
         |Type = VGRTYP   ,
         |Type = LVGRTYP  ) THEN
      Len = Len + 2
    Line = Line Type
    L = Line.0 + 1; Line.0 = L
```

Figure 6. REXX sample program DRAW (Part 9 of 10)

```
    Line.L = Line
    If Null = 1 Then
    Do
      Line = "  "
      Line = Line Left('',20)
      Line = Line "  NULLIF("RIGHT(POSITION,5)")='?'"
      L = Line.0 + 1; Line.0 = L
      Line.L = Line
    End
    Position = Position + Len + 1
  End I
  L = Line.0 + 1; Line.0 = L
  Line.L = " )"
  Return
/*********************************************************************/
/* Display SQLCA                                                     */
/*********************************************************************/
SQLCA:
  "ISREDIT LINE_AFTER "zdest" = MSGLINE 'SQLSTATE="SQLSTATE"'"
  "ISREDIT LINE_AFTER "zdest" = MSGLINE 'SQLWARN ="SQLWARN.0",",
            || SQLWARN.1",",
            || SQLWARN.2",",
            || SQLWARN.3",",
            || SQLWARN.4",",
            || SQLWARN.5",",
            || SQLWARN.6",",
            || SQLWARN.7",",
            || SQLWARN.8",",
            || SQLWARN.9",",
            || SQLWARN.10"'"
  "ISREDIT LINE_AFTER "zdest" = MSGLINE 'SQLERRD ="SQLERRD.1",",
            || SQLERRD.2",",
            || SQLERRD.3",",
            || SQLERRD.4",",
            || SQLERRD.5",",
            || SQLERRD.6"'"
  "ISREDIT LINE_AFTER "zdest" = MSGLINE 'SQLERRP ="SQLERRP"'"
  "ISREDIT LINE_AFTER "zdest" = MSGLINE 'SQLERRM ="SQLERRM"'"
  "ISREDIT LINE_AFTER "zdest" = MSGLINE 'SQLCODE ="SQLCODE"'"
  Exit 20
```

Figure 6. REXX sample program DRAW (Part 10 of 10)

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

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:

AIX
BookManager
C++/MVS
CICS
CICS/ESA
CICS/MVS
DATABASE 2
DB2
DB2/2
DB2/6000
DFSMS
DFSMShsm
Distributed Relational Database Architecture
DRDA
IBM
IMS
IMS/ESA
Language Environment
MVS
MVS/ESA
OS/2
OS/390
OS/400
Parallel Sysplex
QMF
RACF
SQL/DS
VTAM

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.