System i

Database
DB2 Universal Database for iSeries Embedded SQL programming

Version 5 Release 4

System i

Database
DB2 Universal Database for iSeries Embedded SQL programming

Version 5 Release 4

Note

Before using this information and the product it supports, read the information in “Notices,” on page 177.

Sixth Edition (February 2006)

This edition applies to version 5, release 4, modification 0 of IBM i5/OS (product number 5722–SS1) and to all subsequent releases and modifications until otherwise indicated in new editions. This version does not run on all reduced instruction set computer (RISC) models nor does it run on CISC models.

Contents

Embedded SQL programming

This topic collection explains how to create database applications in host languages that use DB2 Universal Database™ for iSeries™ SQL statements and functions.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 176.

What's new for V5R4

This topic highlights the changes made to this topic collection for V5R4.

- Support for embedded SQL in RPG free format was added to "Embedding SQL statements in ILE RPG applications that use SQL" on page 94 and some topics within it.
- The rules for "Names in ILE RPG applications that use SQL" on page 96 were updated.

How to see what's new or changed

To help you see where technical changes have been made, this information uses:

- The » image to mark where new or changed information begins.
- The « image to mark where new or changed information ends.

To find other information about what's new or changed this release, see the Memo to users.

Printable PDF

Use this to view and print a PDF of this information.

To view or download the PDF version of this document, select Embedded SQL programming (about 1750 KB).

Saving PDF files

To save a PDF on your workstation for viewing or printing:

1. Right-click the PDF in your browser (right-click the link above).
2. Click the option that saves the PDF locally.
3. Navigate to the directory in which you want to save the PDF.
4. Click **Save**.

Downloading Adobe Reader

You need Adobe Reader installed on your system to view or print these PDFs. You can download a free copy from the Adobe Web site (www.adobe.com/products/acrobat/readstep.html) .

Common concepts and rules for using embedded SQL

Here are some common concepts and rules for using SQL statements in a host language.

Writing applications that use SQL

You can create database applications in host languages that use DB2® UDB for iSeries SQL statements and functions.

To use embedded SQL, you must have the DB2 UDB Query Manager and SQL Development Kit installed. Additionally, you must have the compilers for the host languages you want to use installed.

Related concepts

"Coding SQL statements in C and C++ applications" on page 13
To embed SQL statements in an ILE C or C++ program, you need to be aware of some unique application and coding requirements. This topic also defines the requirements for host structures and host variables.

"Coding SQL statements in COBOL applications" on page 41
There are unique application and coding requirements for embedding SQL statements in a COBOL program. In this topic, requirements for host structures and host variables are defined.

"Coding SQL statements in PL/I applications" on page 66
There are some unique application and coding requirements for embedding SQL statements in a PL/I program. In this topic, requirements for host structures and host variables are defined.

"Coding SQL statements in RPG/400 applications" on page 81
The RPG/400® licensed program supports both RPG II and RPG III programs.

"Coding SQL statements in ILE RPG applications" on page 91
You need to be aware of the unique application and coding requirements for embedding SQL statements in an ILE RPG program. In this topic, the coding requirements for host variables are defined.

"Coding SQL statements in REXX applications" on page 114
REXX procedures do not have to be preprocessed. At run time, the REXX interpreter passes statements that it does not understand to the current active command environment for processing.

"Preparing and running a program with SQL statements" on page 122
This topic describes some of the tasks for preparing and running an application program.

IBM Developer Kit for Java

Using host variables in SQL statements

When your program retrieves data, the values are put into data items that are defined by your program and that are specified with the INTO clause of a SELECT INTO or FETCH statement. The data items are called host variables.

A *host variable* is a field in your program that is specified in an SQL statement, usually as the source or target for the value of a column. The host variable and column must have compatible data types. Host variables cannot be used to identify SQL objects, such as tables or views, except in the DESCRIBE TABLE statement.

A *host structure* is a group of host variables used as the source or target for a set of selected values (for example, the set of values for the columns of a row). A *host structure array* is an array of host structures that is used in the multiple-row FETCH and blocked INSERT statements.

Note: By using a host variable instead of a literal value in an SQL statement, you give the application program the flexibility to process different rows in a table or view.

For example, instead of coding an actual department number in a WHERE clause, you can use a host variable set to the department number you are currently interested in.

Host variables are commonly used in SQL statements in these ways:

- **In a WHERE clause:** You can use a host variable to specify a value in the predicate of a search condition, or to replace a literal value in an expression. For example, if you have defined a field called EMPID that contains an employee number, you can retrieve the name of the employee whose number is 000110 with:

  ```
  MOVE '000110' TO EMPID.
  EXEC SQL
   SELECT LASTNAME
     INTO :PGM-LASTNAME
     FROM CORPDATA.EMPLOYEE
     WHERE EMPNO = :EMPID
  END-EXEC.
  ```

- **As a receiving area for column values (named in an INTO clause):** You can use a host variable to specify a program data area that is to contain the column values of a retrieved row. The INTO clause names one or more host variables that you want to contain column values returned by SQL. For example, suppose you are retrieving the *EMPNO*, *LASTNAME*, and *WORKDEPT* column values from rows in the CORPDATA.EMPLOYEE table. You could define a host variable in your program to hold each column, then name the host variables with an INTO clause. For example:

  ```
  EXEC SQL
   SELECT EMPNO, LASTNAME, WORKDEPT
     INTO :CBLEMPNO, :CBLNAME, :CBLDEPT
     FROM CORPDATA.EMPLOYEE
     WHERE EMPNO = :EMPID
  END-EXEC.
  ```

  In this example, the host variable CBLEMPNO receives the value from EMPNO, CBLNAME receives the value from LASTNAME, and CBLDEPT receives the value from WORKDEPT.

- **As a value in a SELECT clause:** When specifying a list of items in the SELECT clause, you are not restricted to the column names of tables and views. Your program can return a set of column values intermixed with host variable values and literal constants. For example:

  ```
  MOVE '000220' TO PERSON.
  EXEC SQL
   SELECT "A", LASTNAME, SALARY, :RAISE,
        SALARY + :RAISE
     INTO :PROCESS, :PERSON-NAME, :EMP-SAL,
        :EMP-RAISE, :EMP-TTL
     FROM CORPDATA.EMPLOYEE
     WHERE EMPNO = :PERSON
  END-EXEC.
  ```

  The results are:

PROCESS	PERSON-NAME	EMP-SAL	EMP-RAISE	EMP-TTL
A	LUTZ	29840	4476	34316

- **As a value in other clauses of an SQL statement:**
  - The SET clause in an UPDATE statement
  - The VALUES clause in an INSERT statement
  - The CALL statement

  Related concepts

  SQL reference

Assignment rules for host variables in SQL statements

SQL values are assigned to host variables during the running of FETCH, SELECT INTO, SET, and VALUES INTO statements. SQL values are assigned from host variables during the running of INSERT, UPDATE, and CALL statements.

All assignment operations observe the following rules:

- Numbers and strings are compatible:

  - Numbers can be assigned to character or graphic string columns or host variables.
  - Character and graphic strings can be assigned to numeric columns or numeric host variables.
- All character and DBCS graphic strings are compatible with UCS-2 and UTF-16 graphic columns if conversion is supported between the CCSIDs. All graphic strings are compatible if the CCSIDs are compatible. All numeric values are compatible. Conversions are performed by SQL whenever necessary. All character and DBCS graphic strings are compatible with UCS-2 and UTF-16 graphic columns for assignment operations, if conversion is supported between the CCSIDs. For the CALL statement, character and DBCS graphic parameters are compatible with UCS-2 and UTF-16 parameters if conversion is supported.
- Binary strings are only compatible with binary strings.
- A null value cannot be assigned to a host variable that does not have an associated indicator variable.
- Different types of date/time values are not compatible. Dates are only compatible with dates or string representations of dates; times are only compatible with times or string representations of times; and timestamps are only compatible with timestamps or string representations of timestamps.

  A date can be assigned only to a date column, a character column, a DBCS-open or DBCS-either column or variable, or a character variable. The insert or update value of a date column must be a date or a string representation of a date. A DBCS-open or DBCS-either variable is a variable that was declared in the host language by including the definition of an externally described file. DBCS-open variables are also declared if the job CCSID indicates MIXED data, or the DECLARE VARIABLE statement is used and a MIXED CCSID or the FOR MIXED DATA clause is specified.

  A time can be assigned only to a time column, a character column, a DBCS-open or DBCS-either column or variable, or a character variable. The insert or update value of a time column must be a time or a string representation of a time.

  A timestamp can be assigned only to a timestamp column, a character column, a DBCS-open or DBCS-either column or variable, or a character variable. The insert or update value of a timestamp column must be a timestamp or a string representation of a timestamp.

  Related reference

  DECLARE VARIABLE

Rules for string assignment of host variables in SQL statements:

You need to be aware of these rules regarding character string assignment.

- When a character or graphic string is assigned to a column, the length of the string value must not be greater than the length attribute of the column. (Trailing blanks are normally included in the length of the string. However, for string assignment, trailing blanks are not included in the length of the string.)
- When a binary string is assigned to a column, the length of the string value must not be greater than the length attribute of the column. (Hexadecimal zeros are normally included in the length of the string. However, for string assignment, hexadecimal zeros are not included in the length of the string.)
- When a MIXED character result column is assigned to a MIXED column, the value of the MIXED character result column must be a valid MIXED character string.
- When the value of a result column is assigned to a host variable and the string value of the result column is longer than the length attribute of the host variable, the string is truncated on the right by the necessary number of characters. If this occurs, SQLWARN0 and SQLWARN1 (in the SQL communication area (SQLCA)) are set to W.
- When the value of a result column is assigned to a fixed-length character or graphic host variable or when the value of a host variable is assigned to a fixed-length character or graphic result column and the length of the string value is less than the length attribute of the target, the string is padded on the right with the necessary number of blanks.
- When the value of a result column is assigned to a fixed-length binary host variable or when the value of a host variable is assigned to a fixed-length binary result column and the length of the string value is less than the length attribute of the target, the string is padded on the right with the necessary number of hexadecimal zeros.

- When a MIXED character result column is truncated because the length of the host variable into which it was being assigned was less than the length of the string, the shift-in character at the end of the string is preserved. The result, therefore, is still a valid MIXED character string.

Rules for CCSIDs of host variables in SQL statements:

CCSIDs must be considered when you assign one character or graphic value to another. This includes the assignment of host variables. The database manager uses a common set of system services for converting SBCS data, DBCS data, MIXED data, and graphic data.

The rules for CCSIDs are as follows:

- If the CCSID of the source matches the CCSID of the target, the value is assigned without conversion.
- If the sub-type for the source or target is BIT, the value is assigned without conversion.
- If the value is either null or an empty string, the value is assigned without conversion.
- If conversion is not defined between specific CCSIDs, the value is not assigned and an error message is issued.
- If conversion is defined and needed, the source value is converted to the CCSID of the target before the assignment is performed.

Related concepts

i5/OS globalization

Rules for numeric assignment of host variables in SQL statements:

You need to be aware of these rules regarding numeric assignment.

- **The whole part of a number may be altered when converting it to floating-point**. A single-precision floating-point field can only contain seven decimal digits. Any whole part of a number that contains more than seven digits is altered due to rounding. A double-precision floating point field can only contain 16 decimal digits. Any whole part of a number that contains more than 16 digits is altered due to rounding.
- **The whole part of a number is never truncated.** If necessary, the fractional part of a number is truncated. If the number, as converted, does not fit into the target host variable or column, a negative SQLCODE is returned.
- Whenever a **decimal, numeric, or integer number** is assigned to a decimal, numeric, or integer column or host variable, the number is converted, if necessary, to the precision and scale of the target. The necessary number of leading zeros is added or deleted; in the fractional part of the number, the necessary number of trailing zeros is added, or the necessary number of trailing digits is eliminated.
- When an **integer or floating-point number** is assigned to a decimal or numeric column or host variable, the number is first converted to a temporary decimal or numeric number and then converted, if necessary, to the precision and scale of the target.
  - When a **halfword binary integer** (SMALLINT) with 0 scale is converted to decimal or numeric, the temporary result has a precision of 5 and a scale of 0.
  - When a **fullword binary integer** (INTEGER) is converted to decimal or numeric, the temporary result has a precision of 11 and a scale of 0.
  - When a **double fullword binary integer** (BIGINT) is converted to a decimal or numeric, the temporary result has a precision of 19 and a scale of 0.
  - When a **floating-point number** is converted to decimal or numeric, the temporary result has a precision of 31 and the maximum scale that allows the whole part of the number to be represented without loss of either significance or accuracy.

Rules for date, time, and timestamp assignment of host variables in SQL statements:

You need to be aware of these rules for date, time, and timestamp assignment.

When a **date** is assigned to a host variable, the date is converted to the string representation specified by the DATFMT and DATSEP parameters of the CRTSQLxxx command. Leading zeros are not omitted from any part of the date representation. The host variable must be a fixed or variable-length character string variable with a length of at least 10 bytes for *USA, *EUR, *JIS, or *ISO date formats; 8 bytes for *MDY, *DMY, or *YMD date formats; or 6 bytes for the *JUL date format. If the length is greater than 10, the string is padded on the right with blanks. In ILE RPG and ILE COBOL, the host variable can also be a date variable.

When a **time** is assigned to a host variable, the time is converted to the string representation by the TIMFMT and TIMSEP parameters of the CRTSQLxxx command. Leading zeros are not omitted. The host variable must be a fixed or variable-length character string variable. If the length of the host variable is greater than the string representation of the time, the string is padded on the right with blanks. In ILE RPG and ILE COBOL, the host variable can also be a time variable.

- If the *USA format is used, the length of the host variable must not be less than 8.
- If the *HMS, *ISO, *EUR, or *JIS format is used, the length of the host variable must be at least 8 bytes if seconds are to be included, and 5 bytes if only hours and minutes are needed. In this case, SQLWARN0 and SQLWARN1 (in the SQLCA) are set to W, and if an indicator variable is specified, it is set to the actual number of seconds truncated.

When a **timestamp** is assigned to a host variable, the timestamp is converted to its string representation. Leading zeros are not omitted from any part. The host variable must be a fixed or variable-length character string variable with a length of at least 19 bytes. If the length is less than 26, the host variable does not include all the digits of the microseconds. If the length is greater than 26, the host variable is padded on the right with blanks. In ILE RPG and ILE COBOL, the host variable can also be a timestamp variable.

Indicator variables in applications that use SQL

An *indicator variable* is a halfword integer variable used to indicate whether its associated host variable has been assigned a null value.

- If the value for the result column is null, SQL puts a -1 in the indicator variable.
- If you do not use an indicator variable and the result column is a null value, a negative SQLCODE is returned.
- If the value for the result column causes a data mapping error. SQL sets the indicator variable to -2.

You can also use an indicator variable to verify that a retrieved string value has not been truncated. If truncation occurs, the indicator variable contains a positive integer that specifies the original length of the string. If the string represents a large object (LOB), and the original length of the string is greater than 32 767, the value that is stored in the indicator variable is 32 767, because no larger value can be stored in a halfword integer.

When the database manager returns a value from a result column, you can test the indicator variable. If the value of the indicator variable is less than zero, you know the value of the results column is null. When the database manager returns a null value, the host variable will be set to the default value for the result column.

You specify an indicator variable (preceded by a colon) immediately after the host variable or immediately after the keyword INDICATOR. For example:

```
EXEC SQL
   SELECT COUNT(*), AVG(SALARY)
   INTO :PLICNT, :PLISAL:INDNULL
   FROM CORPDATA.EMPLOYEE
   WHERE EDLEVEL < 18
END-EXEC.
```

You can then test INDNULL to see if it contains a negative value. If it does, you know SQL returned a null value.

Always test for NULL in a column by using the ***IS NULL*** predicate. For example:

```
WHERE expression IS NULL
```

Do not test for NULL in this way:

```
MOVE -1 TO HUIND.
EXEC SQL...WHERE column-name = :HUI :HUIND
```

The EQUAL predicate will always be evaluated as false when it compares a null value. The result of this example will select no rows.

The DISTINCT predicate can be used to perform comparisons when null values may exist.

Related reference

Predicates

Indicator variables used with host structures:

You can specify an *indicator structure* (defined as an array of halfword integer variables) to support a host structure.

If the results column values returned to a host structure can be null, you can add an indicator structure name to the host structure name. This allows SQL to notify your program about each null value returned to a host variable in the host structure.

For example, in COBOL:

```
01  SAL-REC.
    10 MIN-SAL           PIC S9(6)V99 USAGE COMP-3.
    10 AVG-SAL           PIC S9(6)V99 USAGE COMP-3.
    10 MAX-SAL           PIC S9(6)V99 USAGE COMP-3.
01  SALTABLE.
02  SALIND               PIC S9999 USAGE COMP-4 OCCURS 3 TIMES.
01  EDUC-LEVEL           PIC S9999 COMP-4.
   ...
    MOVE 20 TO EDUC-LEVEL.
   ...
    EXEC SQL
     SELECT MIN(SALARY), AVG(SALARY), MAX(SALARY)
       INTO :SAL-REC:SALIND
       FROM CORPDATA.EMPLOYEE
       WHERE EDLEVEL>:EDUC-LEVEL
    END-EXEC.
```

In this example, SALIND is an array containing three values, each of which can be tested for a negative value. If, for example, SALIND(1) contains a negative value, then the corresponding host variable in the host structure (that is, MIN-SAL) is not changed for the selected row.

In the preceding example, SQL selects the column values of the row and puts them into a host structure. Therefore, you must use a corresponding structure for the indicator variables to determine which (if any) selected column values are null.

Indicator variables used to set null values:

You can use an indicator variable to set a null value in a column.

When processing UPDATE or INSERT statements, SQL checks the indicator variable (if it exists). If it contains a negative value, the column value is set to null. If it contains a value greater than -1, the associated host variable contains a value for the column.

For example, you can specify that a value be put in a column (using an INSERT or UPDATE statement), but you may not be sure that the value was specified with the input data. To provide the capability to set a column to a null value, you can write the following statement:

```
EXEC SQL
 UPDATE CORPDATA.EMPLOYEE
   SET PHONENO = :NEWPHONE:PHONEIND
   WHERE EMPNO = :EMPID
END-EXEC.
```

When NEWPHONE contains other than a null value, set PHONEIND to zero by preceding the statement with:

```
MOVE 0 to PHONEIND.
```

Otherwise, to tell SQL that NEWPHONE contains a null value, set PHONEIND to a negative value, as follows:

```
MOVE -1 TO PHONEIND.
```

Handling SQL error return codes using the SQLCA

When an SQL statement is processed in your program, SQL places a return code in the SQLCODE and SQLSTATE fields. The return codes indicate the success or failure of the running of your statement.

If SQL encounters an error while processing the statement, the SQLCODE is a negative number and SUBSTR(SQLSTATE,1,2) is not ’00’, ’01’, or ’02’. If SQL encounters an exception but valid condition while processing your statement, the SQLCODE is a positive number and SUBSTR(SQLSTATE,1,2) is ’01’ or ’02’. If your SQL statement is processed without encountering an error or warning condition, the SQLCODE is zero and the SQLSTATE is ’00000’.

Note: There are situations when a zero SQLCODE is returned to your program and the result might not be satisfactory. For example, if a value was truncated as a result of running your program, the SQLCODE returned to your program is zero. However, one of the SQL warning flags (SQLWARN1) indicates truncation. In this case, the SQLSTATE is not ’00000’.

Attention: If you do not test for negative SQLCODEs or specify a WHENEVER SQLERROR statement, your program will continue to the next statement. Continuing to run after an error can produce unpredictable results.

The main purpose for SQLSTATE is to provide common return codes for common return conditions among the different IBM® relational database systems. SQLSTATEs are particularly useful when handling problems with distributed database operations.

Because the SQLCA is a valuable problem-diagnosis tool, it is a good idea to include in your application programs the instructions necessary to display some of the information contained in the SQLCA. Especially important are the following SQLCA fields:

SQLCODE
Return code.

SQLSTATE
Return code.

SQLERRD(3)
The number of rows updated, inserted, or deleted by SQL.

SQLWARN0

If set to W, at least one of the SQL warning flags (SQLWARN1 through SQLWARNA) is set.

Related concepts

SQL reference

SQL messages and codes

Using the SQL diagnostics area

The SQL diagnostics area is used to keep the returned information for an SQL statement that has been run in a program. It contains all the information that is available to you as an application programmer through the SQLCA.

There are additional values available to provide more detailed information about your SQL statement including connection information. More than one condition can be returned from a single SQL statement. The information in the SQL diagnostics area is available for the previous SQL statement until the next SQL statement is run.

To access the information from the diagnostics area, use the GET DIAGNOSTICS statement. In this statement, you can request multiple pieces of information at one time about the previously run SQL statement. Each item is returned in a host variable. You can also request to get a string that contains all the diagnostic information that is available. Running the GET DIAGNOSTICS statement does not clear the diagnostics area.

Related reference

GET DIAGNOSTICS

Updating applications to use the SQL diagnostics area

You might consider changing your applications to use the SQL diagnostics area instead of the SQL communications area (SQLCA), because the SQL diagnostics area provides some significant advantages over the SQLCA.

One of the best reasons is that the SQLERRM field in the SQLCA is only 70 bytes in length. This is often insufficient for returning meaningful error information to the calling application. Additional reasons for considering the SQL diagnostics area are multiple row operations, and long column and object names. Reporting even simple warnings is sometimes difficult within the restrictions of the 136 byte SQLCA. Quite often, the returned tokens are truncated to fit the restrictions of the SQLCA.

Current applications include the SQLCA definition by using the following:

```
EXEC SQL INCLUDE SQLCA; /* Existing SQLCA */
```

With the conversion to using the SQL diagnostics area, the application would first declare a stand-alone SQLSTATE variable:

```
char SQLSTATE[6]; /* Stand-alone sqlstate */
```

And possibly a stand-alone SQLCODE variable:

```
 long int SQLCODE; /* Stand-alone sqlcode */
```

The completion status of the SQL statement is verified by checking the stand-alone SQLSTATE variable. If upon the completion of the current SQL statement, the application chooses to retrieve diagnostics, the application would run the SQL GET DIAGNOSTICS statement:

```
char hv1[256];
long int hv2;

EXEC SQL GET DIAGNOSTICS :hv1 = COMMAND_FUNCTION,
  :hv2 = COMMAND_FUNCTION_CODE;
```

i5/OS programming model

In the i5/OS® Integrated Language Environment® (ILE), the SQL diagnostics area is scoped to a thread and an activation group. This means that for each activation group in which a thread runs SQL statements, a separate diagnostics area exists for the activation.

Additional notes on using the SQL diagnostics area

In an application program, the SQLCA is replaced with an implicit or a stand-alone SQLSTATE variable, which must be declared in the program.

With multiple condition areas existing in the SQL diagnostics area, the most severe error or warning is returned in the first diagnostics area. There is no specific ordering of the multiple conditions, except that the first diagnostics area will contain the information for the SQLSTATE that is also returned in the SQLSTATE variable.

With the SQLCA, the application program provides the storage for the SQLCA that is used to communicate the results of the run of an SQL statement. With the SQL diagnostics area, the database manager manages the storage for the diagnostics, and the GET DIAGNOSTICS statement is provided to retrieve the contents of the diagnostics area.

Note that the SQLCA will continue to be supported for application programs. Also, the GET DIAGNOSTICS statement can be used in an application program that uses the SQLCA.

Example: SQL routine exception

In this application example, a stored procedure signals an error when an input value is out of range.

```
EXEC SQL CREATE PROCEDURE check_input (IN p1 INT)
LANGUAGE SQL READS SQL DATA
test: BEGIN
 IF p1< 0 THEN
  SIGNAL SQLSTATE VALUE '99999'
   SET MESSAGE_TEXT = 'Bad input value';
  END IF
END test;
```

The calling application checks for a failure and retrieves the information about the failure from the SQL diagnostics area:

```
char SQLSTATE[6]; /* Stand-alone sqlstate */
long int SQLCODE; /* Stand-alone sqlcode */

long int hv1;
char hv2[6];
char hv3[256];

hv1 = -1;
EXEC SQL CALL check_input(:hv1);

if (strncmp(SQLSTATE, "99999", 5) == 0)
{
  EXEC SQL GET DIAGNOSTICS CONDITION 1
    :hv2 = RETURNED_SQLSTATE,
    :hv3 = MESSAGE_TEXT;
}
else
{
}
```

Example: Logging items from the SQL diagnostics area

In this example, an application needs to log all errors for security reasons. The log can be used to monitor the health of a system or to monitor for inappropriate use of a database.

For each SQL error that occurs, an entry is placed in the log. The entry includes when the error occurred, what user was using the application, what type of SQL statement was run, the returned SQLSTATE value, and the message number and corresponding complete message text.

```
char stmt_command[256];
long int error_count;
long int condition_number;
char auth_id[256];
char error_state[6];
char msgid[128];
char msgtext[1024];

EXEC SQL WHENEVER SQLERROR GOTO error;

(application code)


error:
EXEC SQL GET DIAGNOSTICS :stmt_command = COMMAND_FUNCTION,
                         :error_count = NUMBER;

for (condition_number=1;i<=error_count;++condition_number)
{
  EXEC SQL GET DIAGNOSTICS CONDITION :condition_number
    :auth_id = DB2_AUTHORIZATION_ID,
    :error_state = RETURNED_SQLSTATE,
    :msgid = DB2_MESSAGE_ID,
    :msgtext = DB2_MESSAGE_TEXT;

  EXEC SQL INSERT INTO error_log VALUES(CURRENT_TIMESTAMP,
    :stmt_command,
    :condition_number,
    :auth_id,
    :error_state,
    :msgid,
    :msgtext);
}
```

Related reference

GET DIAGNOSTICS

Handling exception conditions with the WHENEVER statement

The WHENEVER statement causes SQL to check the SQLSTATE and SQLCODE and continue processing your program, or branch to another area in your program if an error, exception, or warning exists as a result of running an SQL statement.

An exception condition handling subroutine (part of your program) can then examine the SQLCODE or SQLSTATE field to take an action specific to the error or exception situation.

Note: The WHENEVER statement is not allowed in REXX procedures.

The WHENEVER statement allows you to specify what you want to do whenever a general condition is true. You can specify more than one WHENEVER statement for the same condition. When you do this, the first WHENEVER statement applies to all subsequent SQL statements in the source program until another WHENEVER statement is specified.

The WHENEVER statement looks like this:

```
EXEC SQL
WHENEVER condition action
END-EXEC.
```

There are three conditions you can specify:

SQLWARNING

Specify SQLWARNING to indicate what you want done when SQLWARN0 = W or SQLCODE contains a positive value other than 100 (SUBSTR(SQLSTATE,1,2) =’01’).

Note: SQLWARN0 could be set for several different reasons. For example, if the value of a column was truncated when it was moved into a host variable, your program might not regard this as an error.

SQLERROR

Specify SQLERROR to indicate what you want done when an error code is returned as the result of an SQL statement (SQLCODE < 0) (SUBSTR(SQLSTATE,1,2) > ’02’).

NOT FOUND

Specify NOT FOUND to indicate what you want done when an SQLCODE of +100 and a SQLSTATE of ’02000’ is returned because:

- After a single-row SELECT is issued or after the first FETCH is issued for a cursor, the data the program specifies does not exist.
- After a subsequent FETCH, no more rows satisfying the cursor select-statement are left to retrieve.
- After an UPDATE, a DELETE, or an INSERT, no row meets the search condition.

You can also specify the action you want taken:

CONTINUE

This causes your program to continue to the next statement.

GO TO label

This causes your program to branch to an area in the program. The label for that area may be preceded with a colon. The WHENEVER ... GO TO statement:

- Must be a section name or an unqualified paragraph name in COBOL
- Is a label in PL/I and C
- Is the label of a TAG in RPG

For example, if you are retrieving rows using a cursor, you expect that SQL will eventually be unable to find another row when the FETCH statement is issued. To prepare for this situation, specify a WHENEVER NOT FOUND GO TO ... statement to cause SQL to branch to a place in the program where you issue a CLOSE statement in order to close the cursor properly.

Note: A WHENEVER statement affects all subsequent *source* SQL statements until another WHENEVER is encountered.

In other words, all SQL statements coded between two WHENEVER statements (or following the first, if there is only one) are governed by the first WHENEVER statement, regardless of the path the program takes.

Because of this, the WHENEVER statement *must precede* the first SQL statement it is to affect. If the WHENEVER *follows* the SQL statement, the branch is not taken on the basis of the value of the SQLCODE and SQLSTATE set by that SQL statement. However, if your program checks the SQLCODE or SQLSTATE directly, the check must be done after the SQL statement is run.

The WHENEVER statement does not provide a CALL to a subroutine option. For this reason, you might want to examine the SQLCODE or SQLSTATE value after each SQL statement is run and call a subroutine, rather than use a WHENEVER statement.

Related concepts

"Coding SQL statements in REXX applications" on page 114
REXX procedures do not have to be preprocessed. At run time, the REXX interpreter passes statements that it does not understand to the current active command environment for processing.

Coding SQL statements in C and C++ applications

To embed SQL statements in an ILE C or C++ program, you need to be aware of some unique application and coding requirements. This topic also defines the requirements for host structures and host variables.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 176.

Related concepts

"Writing applications that use SQL" on page 2
You can create database applications in host languages that use DB2 UDB for iSeries SQL statements and functions.

"Error and warning messages during a compile of application programs that use SQL" on page 132
These conditions might produce an error or warning message during an attempted compile process.

Related reference

"Example programs: Using DB2 UDB for iSeries statements" on page 136
Here is a sample application that shows how to code SQL statements in each of the languages that DB2 UDB for iSeries supports.

Defining the SQL communications area in C and C++ applications that use SQL

A C or C++ program can be written to use the SQLCA to check return status for embedded SQL statements, or the program can use the SQL diagnostics area to check return status.

When using the SQLCA, a C or C++ program that contains SQL statements must include one or both of the following:

- An SQLCODE variable declared as long SQLCODE
- An SQLSTATE variable declared as char SQLSTATE[6]

Or,

- An SQLCA (which contains an SQLCODE and SQLSTATE variable).

The SQLCODE and SQLSTATE values are set by the database manager after each SQL statement is run. An application can check the SQLCODE or SQLSTATE value to determine whether the last SQL statement was successful.

You can code the SQLCA in a C or C++ program directly or by using the SQL INCLUDE statement. When coding it directly, initialize the SQLCA using the following statement:

```
struct sqlca sqlca = {0x0000000000000000};
```

Using the SQL INCLUDE statement requests the inclusion of a standard declaration:

```
  EXEC SQL INCLUDE SQLCA ;
```

A standard declaration includes a structure definition and a data area that are named sqlca.

The SQLCODE, SQLSTATE, and SQLCA variables must appear before any executable statements. The scope of the declaration must include the scope of all SQL statements in the program.

The included C and C++ source statements for the SQLCA are:

```
#ifndef SQLCODE
struct sqlca {
              unsigned char sqlcaid[8];
              long          sqlcabc;
              long          sqlcode;
              short         sqlerrml;
              unsigned char sqlerrmc[70];
              unsigned char sqlerrp[8];
              long          sqlerrd[6];
              unsigned char sqlwarn[11];
              unsigned char sqlstate[5];
             };
#define SQLCODE sqlca.sqlcode
#define SQLWARN0 sqlca.sqlwarn[0]
#define SQLWARN1 sqlca.sqlwarn[1]
#define SQLWARN2 sqlca.sqlwarn[2]
#define SQLWARN3 sqlca.sqlwarn[3]
#define SQLWARN4 sqlca.sqlwarn[4]
#define SQLWARN5 sqlca.sqlwarn[5]
#define SQLWARN6 sqlca.sqlwarn[6]
#define SQLWARN7 sqlca.sqlwarn[7]
#define SQLWARN8 sqlca.sqlwarn[8]
#define SQLWARN9 sqlca.sqlwarn[9]
#define SQLWARNA sqlca.sqlwarn[10]
#define SQLSTATE sqlca.sqlstate
#endif
struct sqlca sqlca = {0x0000000000000000};
```

When a declare for SQLCODE is found in the program and the precompiler provides the SQLCA, SQLCADE replaces SQLCODE. When a declare for SQLSTATE is found in the program and the precompiler provides the SQLCA, SQLSTOTE replaces SQLSTATE.

Note: Many SQL error messages contain message data that is of varying length. The lengths of these data fields are embedded in the value of the SQLCA `sqlerrmc` field. Because of these lengths, printing the value of `sqlerrmc` from a C or C++ program might give unpredictable results.

Related concepts

"Using the SQL diagnostics area" on page 9
The SQL diagnostics area is used to keep the returned information for an SQL statement that has been run in a program. It contains all the information that is available to you as an application programmer through the SQLCA.

Related reference

SQL communication area

GET DIAGNOSTICS

Defining SQL descriptor areas in C and C++ applications that use SQL

There are two types of SQL descriptor areas. One is defined with the ALLOCATE DESCRIPTOR statement. The other is defined using the SQL descriptor area (SQLDA) structure. In this topic, only the SQLDA form is discussed.

The following statements can use an SQLDA:

- EXECUTE...USING DESCRIPTOR *descriptor-name*
- FETCH...USING DESCRIPTOR *descriptor-name*
- OPEN...USING DESCRIPTOR *descriptor-name*
- DESCRIBE *statement-name* INTO *descriptor-name*
- DESCRIBE INPUT *statement-name* INTO *descriptor-name*
- DESCRIBE TABLE *host-variable* INTO *descriptor-name*
- PREPARE *statement-name* INTO *descriptor-name*

- CALL...USING DESCRIPTOR *descriptor-name*

Unlike the SQLCA, more than one SQLDA can be in the program, and an SQLDA can have any valid name. The following list includes the statements that require a SQLDA. You can code an SQLDA in a C or C++ program either directly or by using the SQL INCLUDE statement. Using the SQL INCLUDE statement requests the inclusion of a standard SQLDA declaration:

```
  EXEC SQL INCLUDE SQLDA;
```

A standard declaration includes only a structure definition with the name 'sqlda'.

C and C++ declarations that are included for the SQLDA are:

```
  #ifndef SQLDASIZE
  struct sqlda {
                unsigned char sqldaid[8];
                long sqldabc;
                short sqln;
                short sqld;
                struct sqlvar {
                              short sqltype;
                              short sqllen;
                              unsigned char *sqldata;
                              short *sqlind;
                              struct sqlname {
                                             short length;
                                             unsigned char data[30];
                                             } sqlname;
                             } sqlvar[1];
                };
  #define SQLDASIZE(n) (sizeof(struct sqlda) + (n-1)* sizeof(struct sqlvar))
  #endif
```

One benefit from using the INCLUDE SQLDA SQL statement is that you also get the following macro definition:

```
#define SQLDASIZE(n) (sizeof(struct sqlda) + (n-1)* sizeof(struc sqlvar))
```

This macro makes it easy to allocate storage for an SQLDA with a specified number of SQLVAR elements. In the following example, the SQLDASIZE macro is used to allocate storage for an SQLDA with 20 SQLVAR elements.

```
  #include <stdlib.h>
  EXEC SQL INCLUDE SQLDA;

  struct sqlda *mydaptr;
  short numvars = 20;
    .
    .
  mydaptr = (struct sqlda *) malloc(SQLDASIZE(numvars));
  mydaptr->sqln = 20;
```

Here are other macro definitions that are included with the INCLUDE SQLDA statement:

GETSQLDOUBLED(daptr)
Returns 1 if the SQLDA pointed to by daptr has been doubled, or 0 if it has not been doubled. The SQLDA is doubled if the seventh byte in the SQLDAID field is set to '2'.

SETSQLDOUBLED(daptr, newvalue)
Sets the seventh byte of SQLDAID to a new value.

GETSQLDALONGLEN(daptr,n)
Returns the length attribute of the nth entry in the SQLDA to which daptr points. Use this only if the SQLDA was doubled and the nth SQLVAR entry has a LOB data type.

SETSQLDALONGLEN(daptr,n,len)
Sets the SQLLONGLEN field of the SQLDA to which daptr points to len for the nth entry. Use this only if the SQLDA was doubled and the nth SQLVAR entry has a LOB datatype.

GETSQLDALENPTR(daptr,n)
Returns a pointer to the actual length of the data for the nth entry in the SQLDA to which daptr points. The SQLDATALEN pointer field returns a pointer to a long (4 byte) integer. If the SQLDATALEN pointer is zero, a NULL pointer is returned. Use this only if the SQLDA has been doubled.

SETSQLDALENPTR(daptr,n,ptr)
Sets a pointer to the actual length of the data for the nth entry in the SQLDA to which daptr points. Use this only if the SQLDA has been doubled.

When you have declared an SQLDA as a pointer, you must reference it exactly as declared when you use it in an SQL statement, just as you would for a host variable that was declared as a pointer. To avoid compiler errors, the type of the value that is assigned to the sqldata field of the SQLDA must be a pointer of unsigned character. This helps avoid compiler errors. The type casting is only necessary for the EXECUTE, OPEN, CALL, and FETCH statements where the application program is passing the address of the host variables in the program. For example, if you declared a pointer to an SQLDA called mydaptr, you would use it in a PREPARE statement as:

```
EXEC SQL PREPARE mysname INTO :*mydaptr FROM :mysqlstring;
```

SQLDA declarations can appear wherever a structure definition is allowed. Normal C scope rules apply.

Dynamic SQL is an advanced programming technique. With dynamic SQL, your program can develop and then run SQL statements while the program is running. A SELECT statement with a variable SELECT list (that is a list of the data to be returned as part of the query) that runs dynamically requires an SQL descriptor area (SQLDA). This is because you will not know in advance how many or what type of variables to allocate in order to receive the results of the SELECT.

Related concepts

Dynamic SQL applications

Related reference

SQL descriptor area

Embedding SQL statements in C and C++ applications that use SQL

SQL statements can be coded in a C or C++ program wherever executable statements can appear.

Each SQL statement must begin with EXEC SQL and end with a semicolon (;). The EXEC SQL keywords must be on one line. The remaining part of the SQL statement can be on more than one line.

Example: An UPDATE statement coded in a C or C++ program might be coded in the following way:

```
EXEC SQL
  UPDATE DEPARTMENT
  SET MGRNO = :MGR_NUM
  WHERE DEPTNO = :INT_DEPT ;
```

Comments in C and C++ applications that use SQL

In addition to using SQL comments (--), you can include C comments (/*...*/) within embedded SQL statements whenever a blank is allowed, except between the keywords EXEC and SQL.

Comments can span any number of lines. You cannot nest comments. You can use single-line comments (comments that start with //) in C++, but you cannot use them in C.

Continuation for SQL statements in C and C++ applications that use SQL

SQL statements can be contained in one or more lines.

You can split an SQL statement wherever a blank can appear. The backslash (\) can be used to continue a string constant or delimited identifier. Identifiers that are not delimited cannot be continued.

Constants containing DBCS data may be continued across multiple lines in two ways:

- If the character at the right margin of the continued line is a shift-in and the character at the left margin of the continuation line is a shift-out, then the shift characters located at the left and right margin are removed.

  This SQL statement has a valid graphic constant of G'<AABBCCDDEEFFGGHHIIJJKK>'. The redundant shifts at the margin are removed.

  ```
  *...+....1....+....2....+....3....+....4....+....5....+....6....+....7....*....8
  EXEC SQL SELECT * FROM GRAPHTAB           WHERE GRAPHCOL =  G'<AABBCCDDEEFFGGHH>
  <IIJJKK>';
  ```

- It is possible to place the shift characters outside of the margins. For this example, assume the margins are 5 and 75. This SQL statement has a valid graphic constant of G'<AABBCCDDEEFFGGHHIIJJKK>'.

  ```
  *...(....1....+....2....+....3....+....4....+....5....+....6....+....7....)....8
      EXEC SQL SELECT * FROM GRAPHTAB           WHERE GRAPHCOL =  G'<AABBCCDD>
     <EEFFGGHHIIJJKK>';
  ```

Including code in C and C++ applications that use SQL

You can include SQL statements, C, or C++ statements by embedding the following SQL statement in the source code.

```
EXEC SQL INCLUDE member-name;
```

You cannot use C and C++ #include statements to include SQL statements or declarations of C or C++ host variables that are referred to in SQL statements.

Margins in C and C++ applications that use SQL

You must code SQL statements within the margins that are specified by the MARGINS parameter on the CRTSQLCI or CRTSQLCPPI command.

If the MARGINS parameter is specified as *SRCFILE, the record length of the source file will be used. If a value is specified for the right margin and that value is larger than the source record length, the entire record will be read. The value will also apply to any included members. For example, if a right margin of 200 is specified and the source file has a record length of 80, only 80 columns of data will be read from the source file. If an included source member in the same precompile has a record length of 200, the entire 200 from the include will be read.

If EXEC SQL does not start within the specified margins, the SQL precompiler does not recognize the SQL statement.

Related concepts

"CL command descriptions for host language precompilers" on page 174
The DB2 UDB for iSeries database provides commands for precompiling programs coded in these programming languages.

Names in C and C++ applications that use SQL

You can use any valid C or C++ variable name for a host variable. It is subject to these restrictions.

Do not use host variable names or external entry names that begin with SQL, RDI, or DSN in any combination of uppercase or lowercase letters. These names are reserved for the database manager. The length of host variable names is limited to 128.

| If the name **SQL** in any combination of uppercase or lowercase letters is used, unpredictable results
| might occur.

NULLs and NULs in C and C++ applications that use SQL

C, C++, and SQL use the word null, but for different meanings.

The C and C++ languages have a null character (NUL), a null pointer (NULL), and a null statement (just a semicolon (;)). The C NUL is a single character that compares equal to 0. The C NULL is a special reserved pointer value that does not point to any valid data object. The SQL null value is a special value that is distinct from all non-null values and denotes the absence of a (non-null) value.

Statement labels in C and C++ applications that use SQL

Executable SQL statements can be preceded with a label.

Preprocessor sequence for C and C++ applications that use SQL

You must run the SQL preprocessor before the C or C++ preprocessor. You cannot use C or C++ preprocessor directives within SQL statements.

Trigraphs in C and C++ applications that use SQL

Some characters from the C and C++ character set are not available on all keyboards. You can enter these characters into a C or C++ source program by using a sequence of three characters that is called a *trigraph*.

The following trigraph sequences are supported within host variable declarations:

- ??(left bracket
- ??) right bracket
- ??< left brace
- ??> right brace
- ??= pound
- ??/ backslash

WHENEVER statement in C and C++ applications that use SQL

The target for the GOTO clause in an SQL WHENEVER statement must be within the scope of any SQL statements affected by the WHENEVER statement.

Using host variables in C and C++ applications that use SQL

All host variables used in SQL statements must be explicitly declared prior to their first use.

In C, the C statements that are used to define the host variables should be preceded by a BEGIN DECLARE SECTION statement and followed by an END DECLARE SECTION statement. If a BEGIN DECLARE SECTION and END DECLARE SECTION are specified, all host variable declarations used in SQL statements must be between the BEGIN DECLARE SECTION and the END DECLARE SECTION statements. Host variables declared using a typedef identifier also require a BEGIN DECLARE SECTION and END DECLARE SECTION; however, the typedef declarations do not need to be between these two sections.

In C++, the C++ statements that are used to define the host variables must be preceded by a BEGIN DECLARE SECTION statement and followed by an END DECLARE SECTION statement. You cannot use any variable that is not between the BEGIN DECLARE SECTION statement and the END DECLARE SECTION statement as a host variable.

All host variables within an SQL statement must be preceded by a colon (:).

The names of host variables must be unique within the program, even if the host variables are in different blocks or procedures.

An SQL statement that uses a host variable must be within the scope of the statement in which the variable was declared.

Host variables cannot be union elements.

Host variables cannot contain continuation characters within the name.

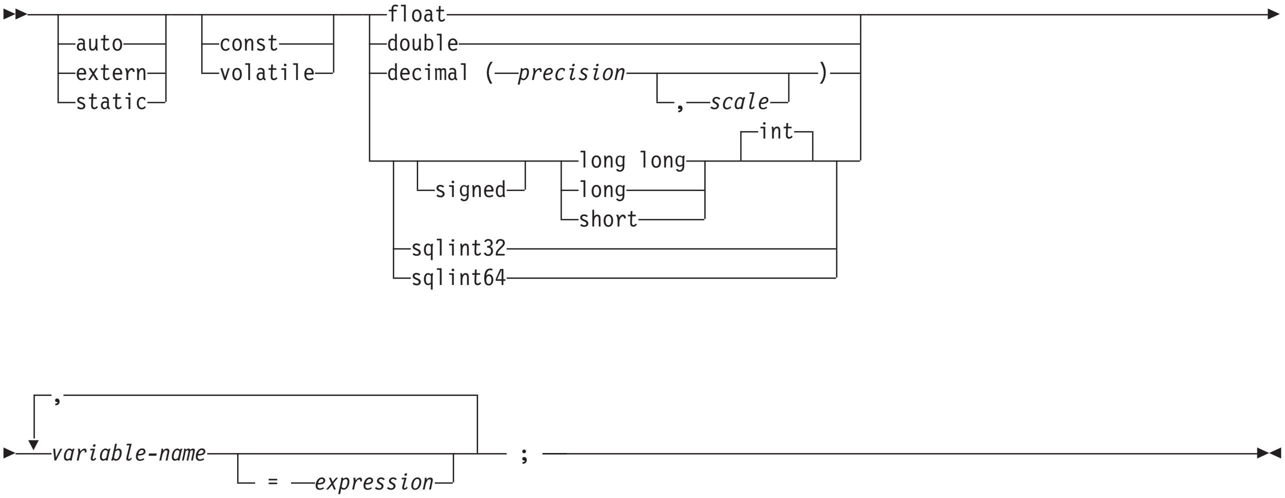
Declaring host variables in C and C++ applications that use SQL

The C and C++ precompilers recognize only a subset of valid C and C++ declarations as valid host variable declarations.

Numeric host variables in C and C++ applications that use SQL:

This figure shows the syntax for valid numeric host variable declarations.

Numeric



Notes:

1. Precision and scale must be integer constants. Precision may be in the range from 1 to 63. Scale may be in the range from 0 to the precision.
2. If using the decimal data type, the header file decimal.h must be included.
3. If using sqlint32 or sqlint64, the header file sqlsystm.h must be included.

Character host variables in C and C++ applications that use SQL:

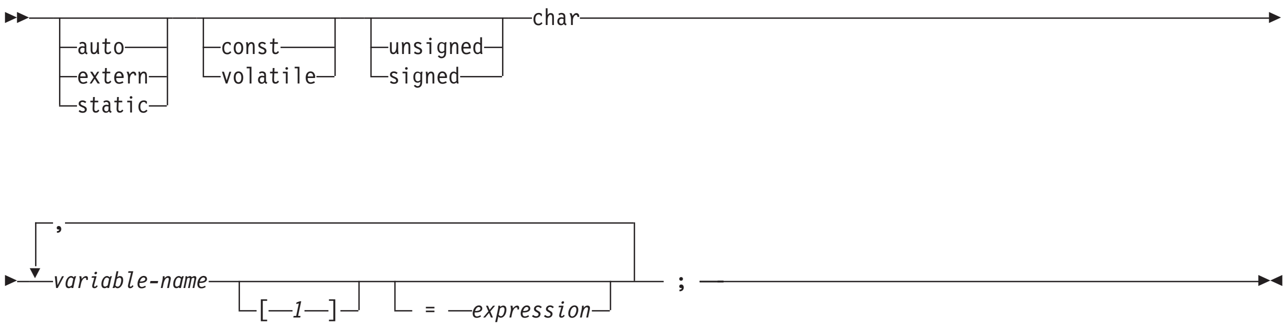
There are three valid forms for character host variables.

These forms are:
- Single-character form
- NUL-terminated character form
- VARCHAR structured form

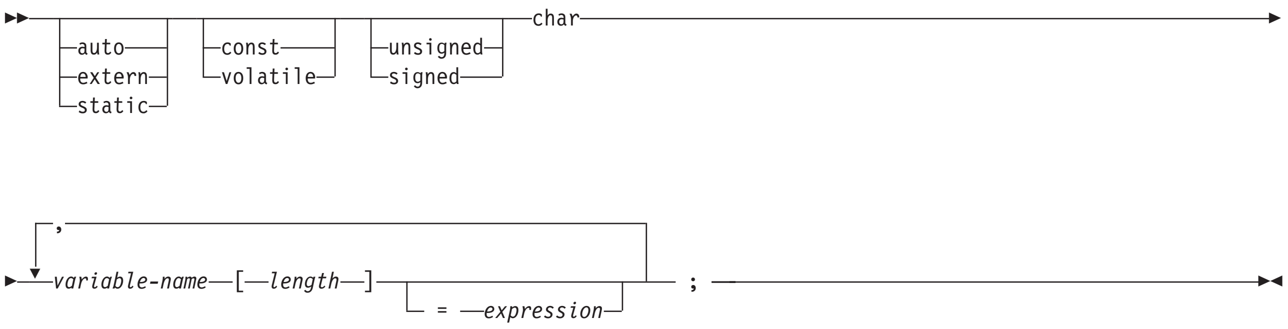
In addition, an SQL VARCHAR declare can be used to define a varchar host variable.

All character types are treated as unsigned.

Single-character form



NUL-terminated character form



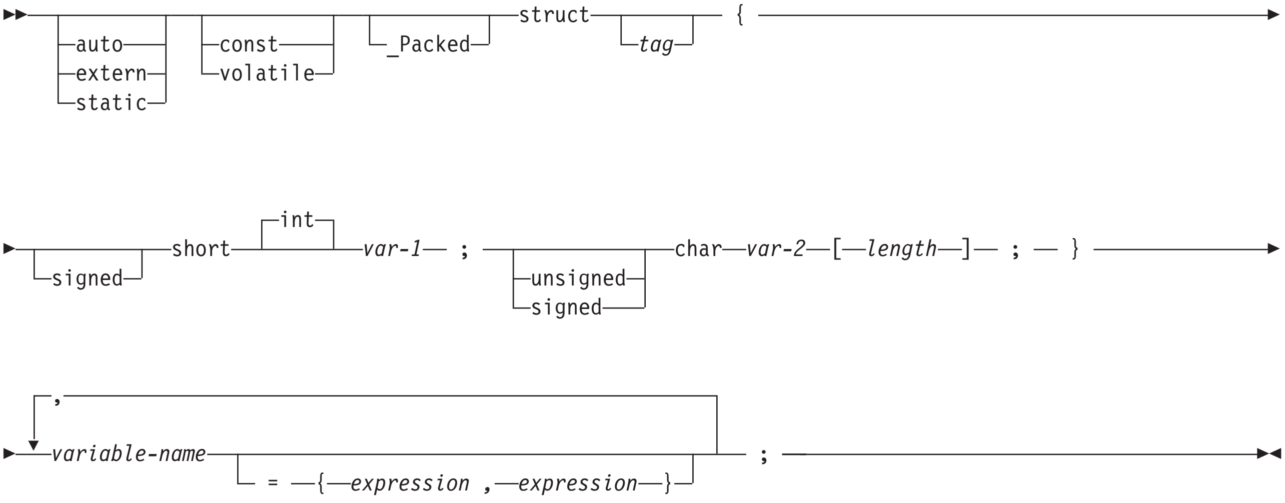
Notes:

1. The length must be an integer constant that is greater than 1 and not greater than 32 741.
2. If the *CNULRQD option is specified on the CRTSQLCI or CRTSQLCPPI command, the input host variables must contain the NUL-terminator. Output host variables are padded with blanks, and the last character is the NUL-terminator. If the output host variable is too small to contain both the data and the NUL-terminator, the following actions are taken:
   - The data is truncated
   - The last character is the NUL-terminator
   - SQLWARN1 is set to ’W’
3. If the *NOCNULRQD option is specified on the CRTSQLCI or CRTSQLCPPI command, the input variables do not need to contain the NUL-terminator.

   The following applies to output host variables.
   - If the host variable is large enough to contain the data and the NUL-terminator, then the following actions are taken:
     - The data is returned, but the data is not padded with blanks
     - The NUL-terminator immediately follows the data
   - If the host variable is large enough to contain the data but not the NUL-terminator, then the following actions are taken:
     - The data is returned
     - A NUL-terminator is not returned
     - SQLWARN1 is set to ’N’
   - If the host variable is not large enough to contain the data, the following actions are taken:
     - The data is truncated

- A NUL-terminator is not returned
- SQLWARN1 is set to 'W'

VARCHAR structured form



Notes:

1. *length* must be an integer constant that is greater than 0 and not greater than 32 740.
2. *var-1* and *var-2* must be simple variable references and cannot be used individually as integer and character host variables.
3. The struct tag can be used to define other data areas, but these cannot be used as host variables.
4. The VARCHAR structured form should be used for bit data that may contain the NULL character. The VARCHAR structured form will not be ended using the nul-terminator.
5. _Packed must not be used in C++. Instead, specify #pragma pack(1) prior to the declaration and #pragma pack() after the declaration.

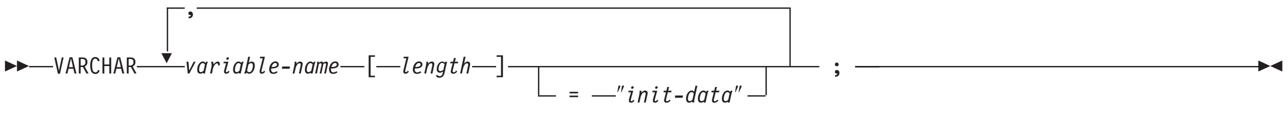
   Note: You can use #pragma pack (reset) instead of #pragma pack() because they are the same.

   ```
   #pragma pack(1)
    struct VARCHAR {
         short len;
         char s[10];
         } vstring;
   #pragma pack()
   ```

Example:

```
EXEC SQL BEGIN DECLARE SECTION;

   /* valid declaration of host variable vstring */

   struct VARCHAR {
      short len;
      char s[10];
      } vstring;

   /* invalid declaration of host variable wstring */

   struct VARCHAR wstring;
```

SQL VARCHAR form



Notes:

1. VARCHAR can be in mixed case.
2. Length must be an integer constant that is greater than 0 and not greater than 32 740.
3. The SQL VARCHAR form should be used for bit data that may contain the NULL character. The SQL VARCHAR form will not be ended using the nul-terminator.

Example

The following declaration:

```
VARCHAR vstring[528]="mydata";
```

Results in the generation of the following structure:

```
_Packed struct { short len;
                 char data[528];}
 vstring={6, "mydata"};
```

The following declaration:

```
VARCHAR vstring1[111],
        vstring2[222]="mydata",
        vstring3[333]="more data";
```

Results in the generation of the following structures:

```
_Packed struct { short len;
                 char data[111];}
vstring1;

_Packed struct { short len;
                 char data[222];}
vstring2={6,"mydata"};

_Packed struct { short len;
                 char data[333};}
vstring3={9,"more data"};
```

Graphic host variables in C and C++ applications that use SQL:

There are three valid forms for graphic host variables.

- Single-graphic form
- NUL-terminated graphic form
- VARGRAPHIC structured form

Single-graphic form

```
                                         ┌─,──────────────────────────────┐
                                         ▼                                │
►►──┬────────┬──┬──────────┬──wchar_t────variable-name──┬────────────────┬┴── ; ──────────────►◄
    ├─auto───┤  ├─const────┤                            └─ = ─expression─┘
    ├─extern─┤  └─volatile─┘
    └─static─┘
```

NUL-terminated graphic form

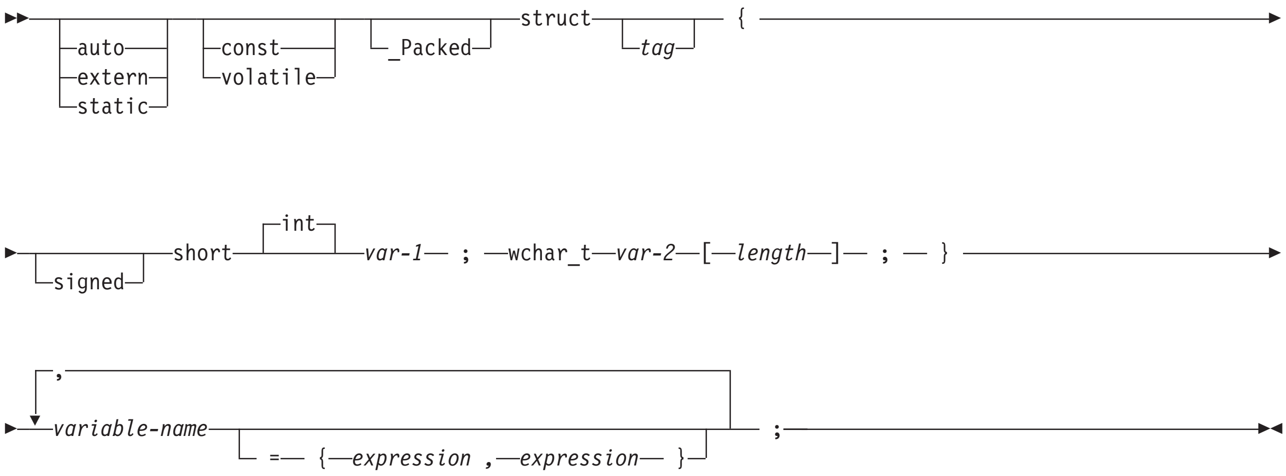
```
                                         ┌─,─────────────────────────────────────────────────┐
                                         ▼                                                   │
►►──┬────────┬──┬──────────┬──wchar_t────variable-name──[──length──]──┬────────────────┬─────┴── ; ────►◄
    ├─auto───┤  ├─const────┤                                          └─ = ─expression─┘
    ├─extern─┤  └─volatile─┘
    └─static─┘
```

Notes:

1. *length* must be an integer constant that is greater than 1 and not greater than 16371.
2. If the *CNULRQD option is specified on the CRTSQLCI or CRTSQLCPPI command, then input host variables must contain the graphic NUL-terminator (/0/0). Output host variables are padded with DBCS blanks, and the last character is the graphic NUL-terminator. If the output host variable is too small to contain both the data and the NUL-terminator, the following actions are taken:
   - The data is truncated
   - The last character is the graphic NUL-terminator
   - SQLWARN1 is set to 'W'

   If the *NOCNULRQD option is specified on the CRTSQLCI or CRTSQLCPPI command, the input host variables do not need to contain the graphic NUL-terminator. The following is true for output host variables.
   - If the host variable is large enough to contain the data and the graphic NUL-terminator, the following actions are taken:
     – The data is returned, but is not padded with DBCS blanks
     – The graphic NUL-terminator immediately follows the data
   - If the host variable is large enough to contain the data but not the graphic NUL-terminator, the following actions are taken:
     – The data is returned
     – A graphic NUL-terminator is not returned
     – SQLWARN1 is set to 'N'
   - If the host variable is not large enough to contain the data, the following actions are taken:
     – The data is truncated
     – A graphic NUL-terminator is not returned
     – SQLWARN1 is set to 'W'

VARGRAPHIC structured form



Notes:

1. *length* must be an integer constant that is greater than 0 and not greater than 16370.
2. *var-1* and *var-2* must be simple variable references and cannot be used as host variables.
3. The struct tag can be used to define other data areas, but these cannot be used as host variables.
4. _Packed must not be used in C++. Instead, specify #pragma pack(1) prior to the declaration and #pragma pack() after the declaration.

```
#pragma pack(1)
 struct VARGRAPH {
      short len;
      wchar_t s[10];
      } vstring;
#pragma pack()
```

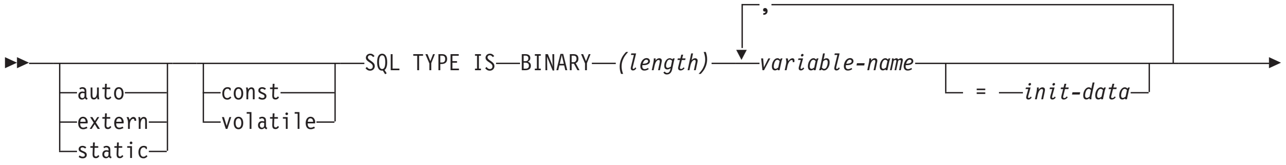
Example

```
EXEC SQL BEGIN DECLARE SECTION;

  /* valid declaration of host variable graphic string */

  struct VARGRAPH {
     short len;
     wchar_t s[10];
     } vstring;

  /* invalid declaration of host variable wstring */

  struct VARGRAPH wstring;
```

Binary host variables in C and C++ applications that use SQL:

C and C++ do not have variables that correspond to the SQL binary data types. To create host variables that can be used with these data types, use the SQL TYPE IS clause. The SQL precompiler replaces this declaration with a C language structure in the output source member.

BINARY



```
►─ ; ──────────────────────────────────────────────────────────────────►◄
```

VARBINARY

```
►►──┬────────┬──┬──────────┬──SQL TYPE IS──┬─VARBINARY──────┬──(length)──────────────►
    ├─auto───┤  ├─const────┤               └─BINARY VARYING─┘
    ├─extern─┤  └─volatile─┘
    └─static─┘
```

```
   ┌─,──────────────────────────────────────────────┐
   ▼                                                │
►──variable-name──┬────────────────────────────────┬┴── ; ─────────────►◄
                  ├─ = ──{──init-len,"init-data" ──}──┤
                  └─ = ──SQL_VARBINARY_INIT("init-data") ─┘
```

Notes:

1. For BINARY host variables, the length must be in the range 1 to 32766.
2. For VARBINARY and BINARY VARYING host variables, the length must in the range 1 to 32740.
3. SQL TYPE IS, BINARY, VARBINARY, and BINARY VARYING can be in mixed case.

BINARY example

The following declaration:

```
SQL TYPE IS BINARY(4) myBinField;
```

Results in the generation of the following code:

```
unsigned char myBinField[4];
```

VARBINARY example

The following declaration:

```
SQL TYPE IS VARBINARY(12) myVarBinField;
```

Results in the generation of the following structure:

```
_Packed struct myVarBinField_t {
 short length;
 char  data[12]; }
myVarBinField;
```

LOB host variables in C and C++ applications that use SQL:

C and C++ do not have variables that correspond to the SQL data types for LOBs (large objects). To create host variables that can be used with these data types, use the SQL TYPE IS clause. The SQL precompiler replaces this declaration with a C language structure in the output source member.

LOB host variable

```
►►──┬────────┬──┬──────────┬──SQL TYPE IS──┬─CLOB───┬──(──length──┬───┬──)──────────────────────►
    ├─auto───┤  ├─const────┤               ├─DBCLOB─┤             ├─K─┤
    ├─extern─┤  └─volatile─┘               └─BLOB───┘             ├─M─┤
    └─static─┘                                                    └─G─┘

   ┌─,──────────────────────────────────────────────┐
   ▼                                                │
►──variable-name──┬─────────────────────────────────┬┴── ; ──────────────────────────────────────►◄
                  ├─ = ─{─init-len,"init-data"─}────┤
                  ├─ = ─SQL_CLOB_INIT("init-data")──┤
                  ├─ = ─SQL_DBCLOB_INIT("init-data")┤
                  └─ = ─SQL_BLOB_INIT("init-data")──┘
```

Notes:

1. K multiplies *length* by 1024. M multiplies *length* by 1 048 576. G multiplies *length* by 1 073 741 824.
2. For BLOB and CLOB, 1 ≤ *length* ≤ 2 147 483 647
3. For DBCLOB, 1 ≤ *length* ≤ 1 073 741 823
4. SQL TYPE IS, BLOB, CLOB, DBCLOB, K, M, G can be in mixed case.
5. The maximum length allowed for the initialization string is 32 766 bytes.
6. The initialization length, *init-len*, must be a numeric constant (that is, it cannot include K, M, or G).
7. If the LOB is not initialized within the declaration, then no initialization will be done within the precompiler generated code.
8. The precompiler generates a structure tag which can be used to cast to the host variable's type.
9. Pointers to LOB host variables can be declared, with the same rules and restrictions as for pointers to other host variable types.
10. CCSID processing for LOB host variables will be the same as the processing for other character and graphic host variable types.
11. If a DBCLOB is initialized, it is the user's responsibility to prefix the string with an 'L' (indicating a wide-character string).

CLOB example

The following declaration:

```
SQL TYPE IS CLOB(128K) var1, var2 = {10, "data2data2"};
```

The precompiler will generate for C:

```
_Packed struct var1_t {
 unsigned long length;
 char data[131072];
 } var1,var2={10,"data2data2"};
```

DBCLOB example

The following declaration:

```
SQL TYPE IS DBCLOB(128K) my_dbclob;
```

The precompiler will then generate:

```
_Packed struct my_dbclob_t {
 unsigned long length;
 wchar_t data[131072]; } my_dbclob;
```

BLOB example

The following declaration:

```
static SQL TYPE IS BLOB(128K)
  my_blob=SQL_BLOB_INIT("mydata");
```

Results in the generation of the following structure:

```
static struct my_blob_t {
   unsigned long length;
   char          data[131072];
} my_blob=SQL_BLOB_INIT("my_data");
```

LOB locator

```
►►──┬────────┬──┬──────────┬──SQL TYPE IS──┬─CLOB_LOCATOR───┬──────────────────────────►
    ├─auto───┤  ├─const────┤               ├─DBCLOB_LOCATOR─┤
    ├─extern─┤  └─volatile─┘               └─BLOB_LOCATOR───┘
    └─static─┘


   ┌─,─────────────────────────────┐
   ▼                               │
►──variable-name──┬──────────────┬─┴─ ; ─────────────────────────────────────────────►◄
                  └─ = ─init-value─┘
```

Notes:

1. SQL TYPE IS, BLOB_LOCATOR, CLOB_LOCATOR, DBCLOB_LOCATOR can be in mixed case.
2. *init-value* permits the initialization of pointer locator variables. Other types of initialization will have no meaning.
3. Pointers to LOB locators can be declared with the same rules and restrictions as for pointers to other host variable types.

CLOB locator example

The following declaration:

```
static SQL TYPE IS CLOB_LOCATOR my_locator;
```

Results in the following generation:

```
static long int unsigned my_locator;
```

BLOB and DBCLOB locators have similar syntax.

LOB file reference variable

```
►►──┬────────┬──┬──────────┬──SQL TYPE IS──┬─CLOB_FILE───┬─────────────────────────────►
    ├─auto───┤  ├─const────┤               ├─DBCLOB_FILE─┤
    ├─extern─┤  └─volatile─┘               └─BLOB_FILE───┘
    └─static─┘
```

```
   ┌─,──────────────────────────────────┐
   ▼                                    │
►──variable-name──┬──────────────────┬──┴── ; ──────────────────────────────────────────►◄
                  └─ = ──init-value──┘
```

Notes:

1. SQL TYPE IS, BLOB_FILE, CLOB_FILE, DBCLOB_FILE can be in mixed case.
2. Pointers to LOB File Reference Variables can be declared, with the same rules and restrictions as for pointers to other host variable types.

CLOB file reference example

The following declaration:

```
static SQL TYPE IS CLOB_FILE my_file;
```

Results in the generation of the following structure:

```
static _Packed struct {
     unsigned long    name_length;
     unsigned long    data_length;
     unsigned long    file_options;
              char    name[255];
} my_file;
```

BLOB and DBCLOB file reference variables have similar syntax.

The precompiler generates declarations for the following file option constants. You can use these constants to set the file_options variable when you use file reference host variables.

- SQL_FILE_READ (2)
- SQL_FILE_CREATE (8)
- SQL_FILE_OVERWRITE (16)
- SQL_FILE_APPEND (32)

Related reference

LOB file reference variables

ROWID host variables in C and C++ applications that use SQL:

C and C++ do not have a variable that corresponds to the SQL data type ROWID. To create host variables that can be used with this data type, use the SQL TYPE IS clause. The SQL precompiler replaces this declaration with a C language structure in the output source member.

ROWID

```
                    ┌─,─────────────┐
                    ▼               │
►►──SQL TYPE IS ROWID───variable-name─┴── ; ──────────────────────────────────────────►◄
```

Note: SQL TYPE IS ROWID can be in mixed case.

ROWID example

The following declaration:

```
SQL TYPE IS ROWID myrowid, myrowid2;
```

Results in the generation of the following structure:

```
_Packed struct { short len;
                 char data[40];}
myrowid1, myrowid2;
```

Using host structures in C and C++ applications that use SQL

In C and C++ programs, you can define a *host structure*, which is a named set of elementary C or C++ variables.

Host structures have a maximum of two levels, even though the host structure might itself occur within a multilevel structure. An exception is the declaration of a varying-length string, which requires another structure.

A host structure name can be a group name whose subordinate levels name elementary C or C++ variables. For example:

```
   struct {
            struct {
                     char c1;
                     char c2;
                    } b_st;
          } a_st;
```

In this example, b_st is the name of a host structure consisting of the elementary items c1 and c2.

You can use the structure name as a shorthand notation for a list of scalars, but only for a two-level structure. You can qualify a host variable with a structure name (for example, structure.field). Host structures are limited to two levels. (For example, in the above host structure example, the a_st cannot be referred to in SQL.) A structure cannot contain an intermediate level structure. In the previous example, a_st could not be used as a host variable or referred to in an SQL statement. A host structure for SQL data has two levels and can be thought of as a named set of host variables. After the host structure is defined, you can refer to it in an SQL statement instead of listing the several host variables (that is, the names of the host variables that make up the host structure).

For example, you can retrieve all column values from selected rows of the table CORPDATA.EMPLOYEE with:

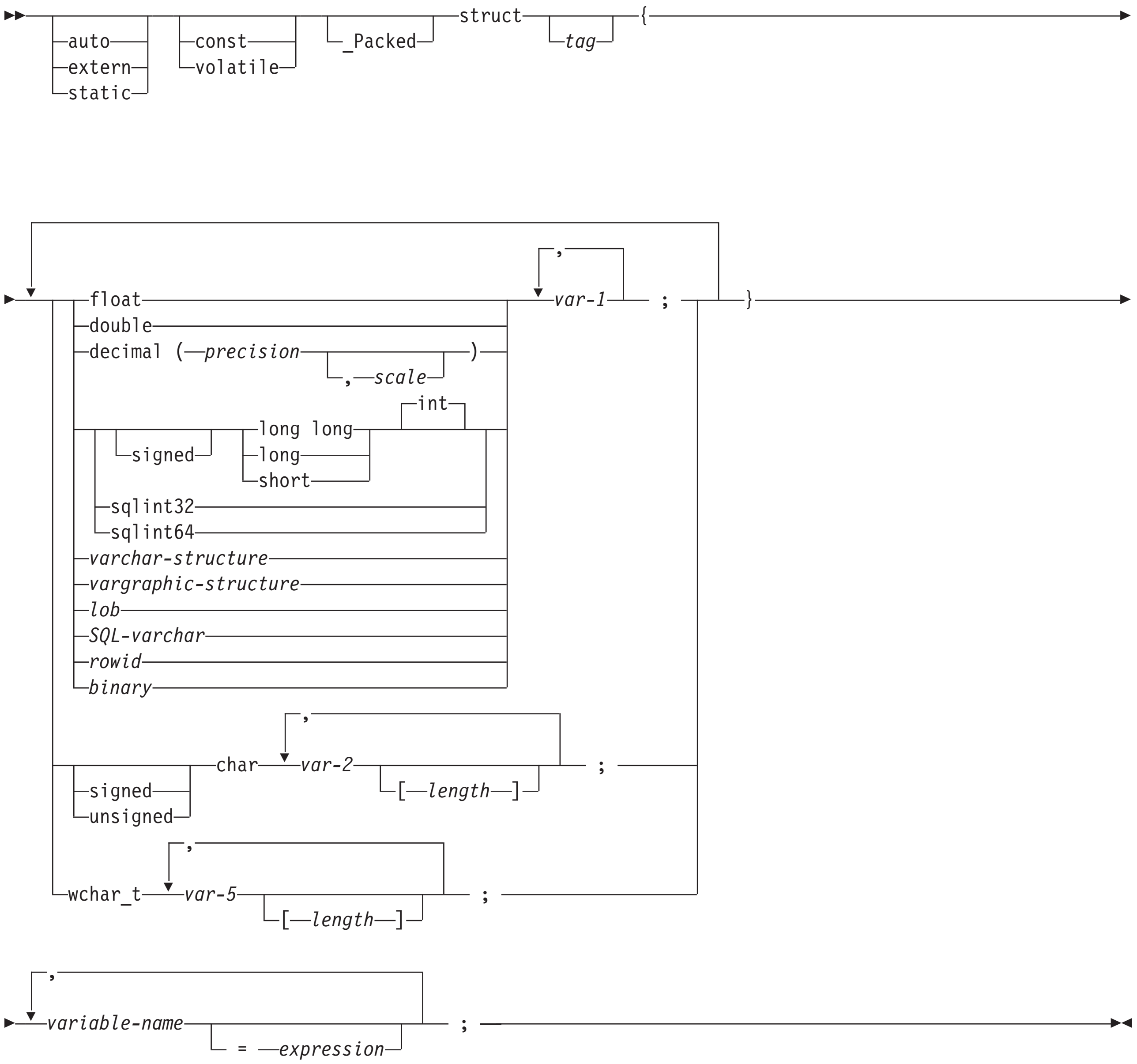
```
  struct { char empno[7];
                struct          { short int firstname_len;
                                  char firstname_text[12];
                                } firstname;
                char midint,
                struct          { short int lastname_len;
                                  char lastname_text[15];
                                } lastname;
                char workdept[4];
                } pemp1;
    .....
  strcpy("000220",pemp1.empno);
  .....
  exec sql
    SELECT *
      INTO :pemp1
      FROM corpdata.employee
      WHERE empno=:pemp1.empno;
```

Notice that in the declaration of pemp1, two varying-length string elements are included in the structure: firstname and lastname.

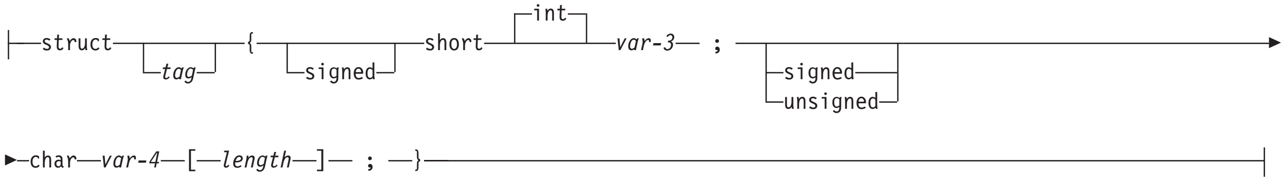
Host structure declarations in C and C++ applications that use SQL

These figures show the valid syntax for host structure declarations.

Host structures



varchar-structure:



Host structures (continued)

vargraphic-structure:

```
                                        ┌─int─┐
├──struct──┬─────┬──{──┬────────┬──short──┴─────┴──var-6── ; ──wchar_t──var-7──[──length──]── ; ──}────┤
           └─tag─┘     └─signed─┘
```

lob:

```
├──SQL TYPE IS──┬──┬─CLOB───┬──(──length──┬───┬──)──┬──────────────────────────────────┤
                │  ├─DBCLOB─┤             ├─K─┤     │
                │  └─BLOB───┘             ├─M─┤     │
                │                         └─G─┘     │
                ├──┬─CLOB_LOCATOR───┬───────────────┤
                │  ├─DBCLOB_LOCATOR─┤               │
                │  └─BLOB_LOCATOR───┘               │
                └──┬─CLOB_FILE───┬──────────────────┘
                   ├─DBCLOB_FILE─┤
                   └─BLOB_FILE───┘
```

SQL-varchar:

```
├──VARCHAR──variable-name──[──length──]──────────────────────────────────────────────┤
```

rowid:

```
├──SQL TYPE IS ROWID─────────────────────────────────────────────────────────────────┤
```

binary:

```
├──SQL TYPE IS──┬─BINARY─────────┬──(──length──)─────────────────────────────────────┤
                ├─VARBINARY──────┤
                └─BINARY VARYING─┘
```

Notes:

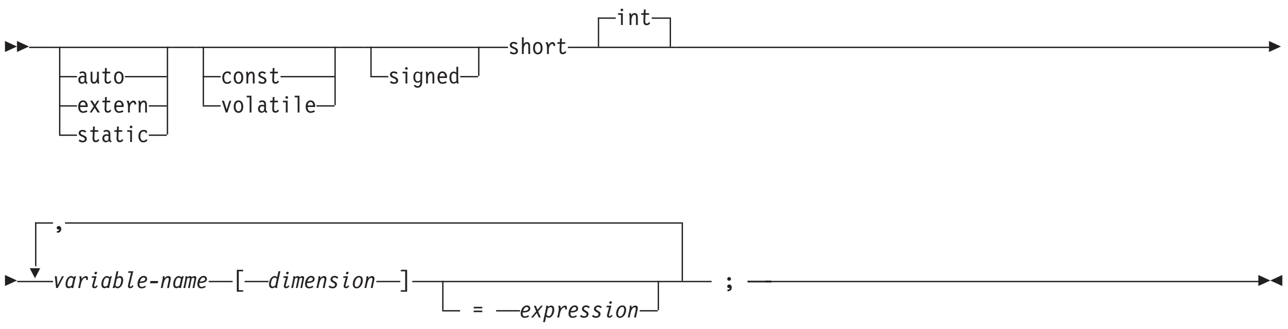
1. For details on declaring numeric, character, graphic, LOB, ROWID, and binary host variables, see the notes under numeric, character, graphic, LOB, ROWID, and binary host variables.
2. A structure of a short int followed by either a char or wchar_t array is always interpreted by the SQL C and C++ precompilers as either a VARCHAR or VARGRAPHIC structure.
3. _Packed must not be used in C++. Instead, specify #pragma pack(1) prior to the declaration and #pragma pack() after the declaration.

   ```
   #pragma pack(1)
    struct {
          short myshort;
          long mylong;
          char mychar[5];
           } a_st;
   #pragma pack()
   ```
4. If using sqlint32 or sqlint64, the header file sqlsystm.h must be included.

Host structure indicator array in C and C++ applications that use SQL

This figure shows the valid syntax for host structure indicator array declarations.

Host structure indicator array



Note: Dimension must be an integer constant between 1 and 32767.

Using arrays of host structures in C and C++ applications that use SQL

In C and C++ programs, you can define a host structure array that has the dimension attribute. Host structure arrays have a maximum of two levels, even though the array might occur within a multiple-level structure. Another structure is not needed if a varying-length character string or a varying-length graphic string is not used.

In this C example,

```
struct {
       _Packed struct{
                      char c1_var[20];
                      short c2_var;
                     } b_array[10];
       } a_struct;
```

and in this C++ example,

```
#pragma pack(1)
struct {
       struct{
                      char c1_var[20];
                      short c2_var;
                     } b_array[10];
       } a_struct;
#pragma pack()
```

the following are true:

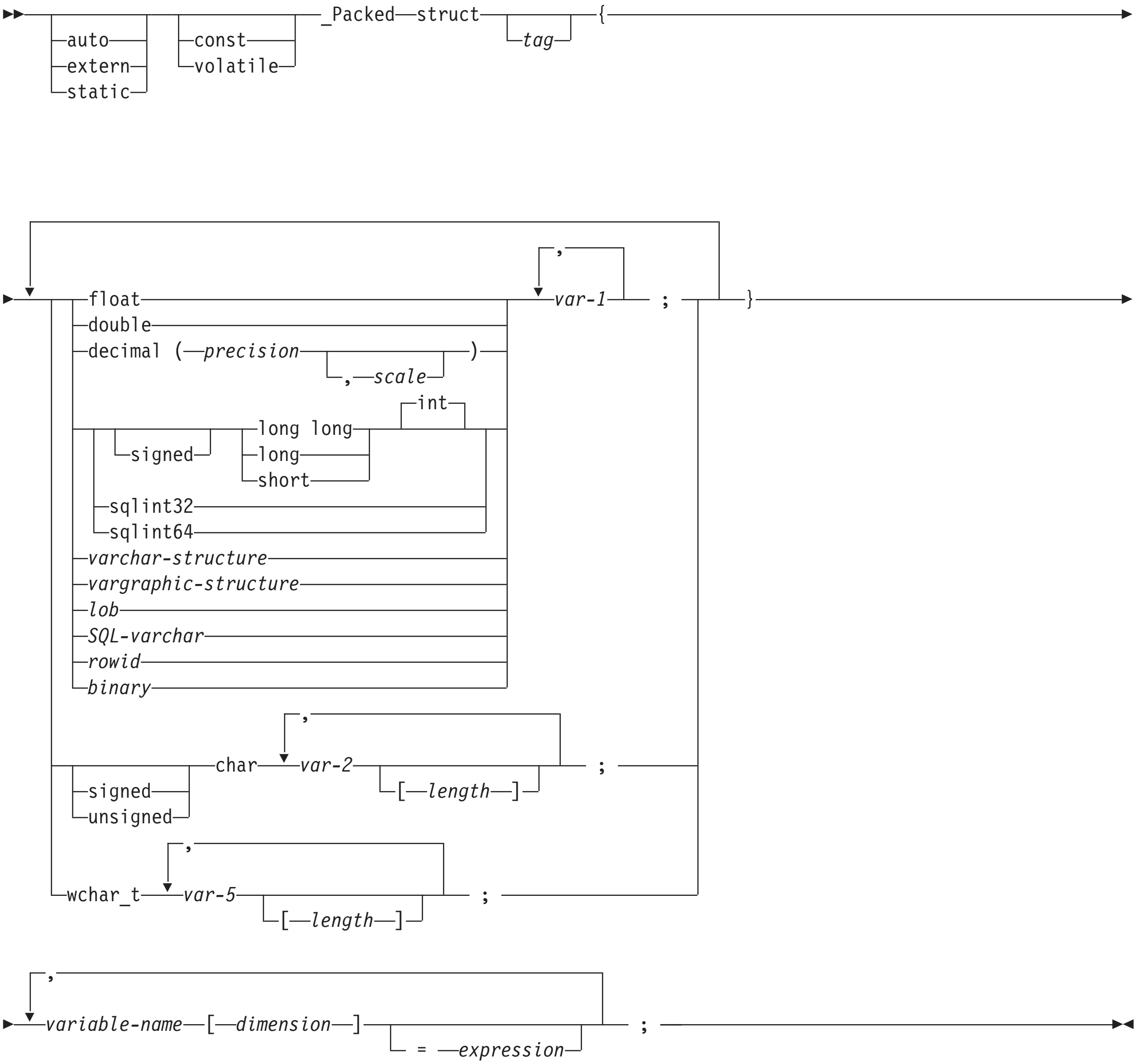
- All of the members in b_array must be valid variable declarations.
- The _Packed attribute must be specified for the struct tag.
- b_array is the name of an array of host structures containing the members c1_var and c2_var.
- b_array may only be used on the blocked forms of FETCH statements and INSERT statements.
- c1_var and c2_var are not valid host variables in any SQL statement.
- A structure cannot contain an intermediate level structure.

For example, in C you can retrieve 10 rows from the cursor with:

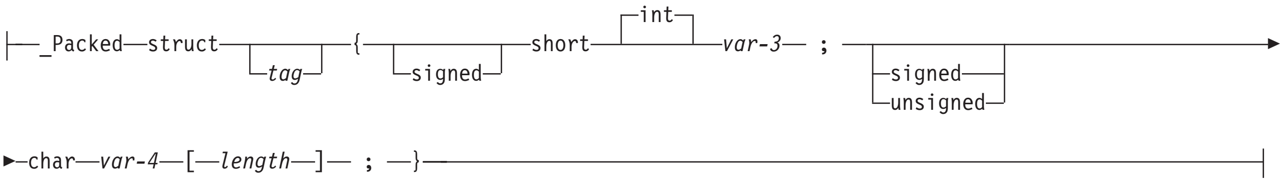
```
_Packed struct {char first_initial;
                char middle_initial;
                _Packed struct {short lastname_len;
                                char lastname_data[15];
                               } lastname;
                double total_salary;
                } employee_rec[10];
struct { short inds[4];
       } employee_inds[10];
...
EXEC SQL DECLARE C1 CURSOR FOR
 SELECT SUBSTR(FIRSTNME,1,1), MIDINIT, LASTNAME,
              SALARY+BONUS+COMM
        FROM CORPDATA.EMPLOYEE;
EXEC SQL OPEN C1;
EXEC SQL FETCH C1 FOR 10 ROWS INTO :employee_rec:employee_inds;
...
```

Host structure array in C and C++ applications that use SQL

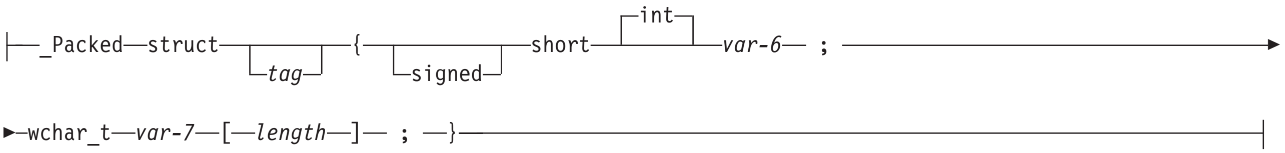
The figure shows the valid syntax for host structure array declarations.



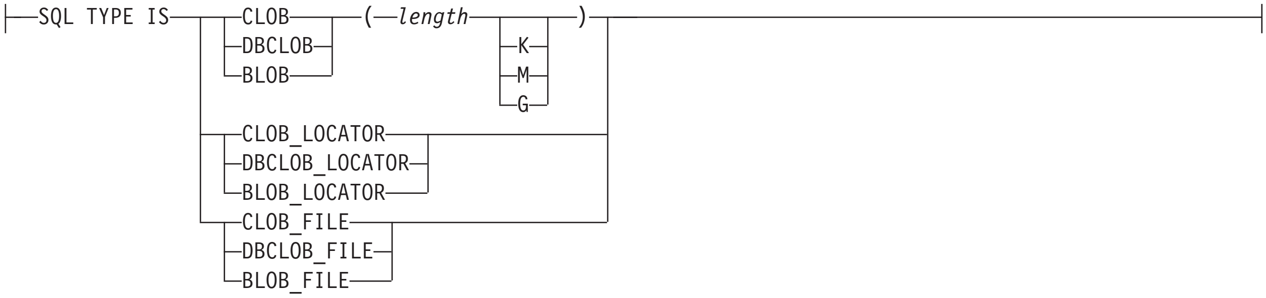
varchar-structure:



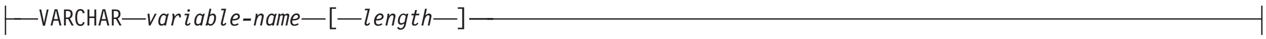
vargraphic-structure:



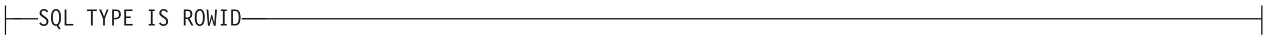
lob:



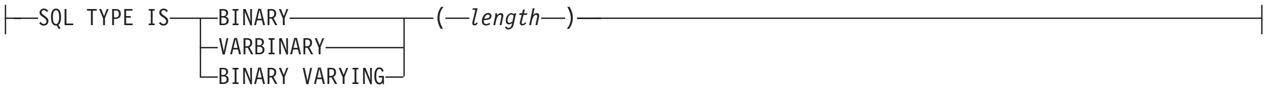
SQL-varchar:



rowid:



binary:



Notes:

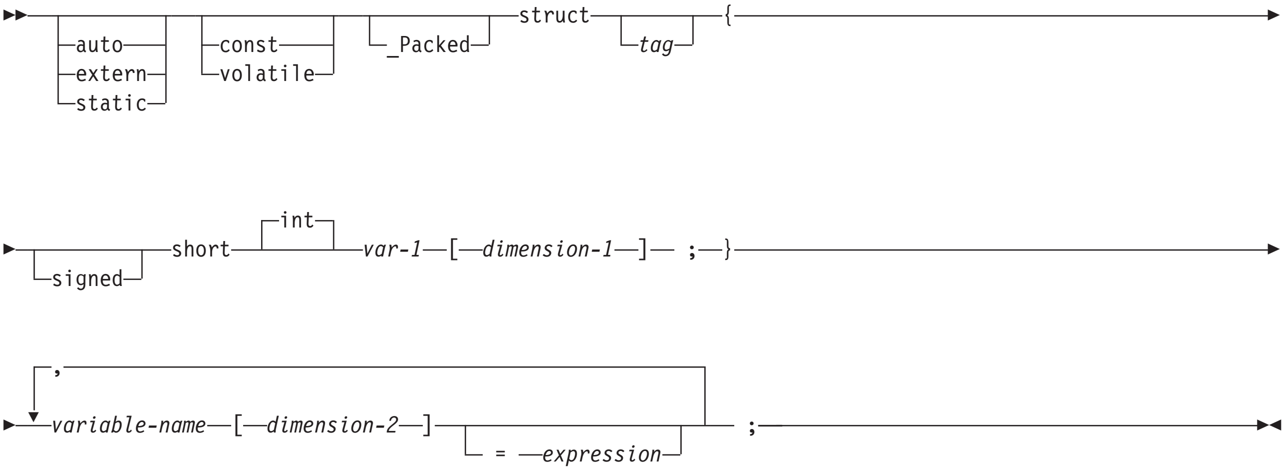
1. For details on declaring numeric, character, graphic, LOB, ROWID, and binary host variables, see the notes under numeric-host variables, character-host, graphic-host variables, LOB host variables, ROWID host variables, and binary host variables.
2. The struct tag can be used to define other data areas, but these cannot be used as host variables.

3. Dimension must be an integer constant between 1 and 32767.
4. _Packed must not be used in C++. Instead, specify #pragma pack(1) prior to the declaration and #pragma pack() after the declaration.
5. If using sqlint32 or sqlint64, the header file sqlsystm.h must be included.

Host structure array indicator structure in C and C++ applications that use SQL

The figure shows the valid syntax for host structure array indicator structure declarations.

Host Structure Array Indicator Structure



Notes:

1. The struct tag can be used to define other data areas, but they cannot be used as host variables.
2. dimension-1 and dimension-2 must both be integer constants between 1 and 32767.
3. _Packed must not be used in C++. Instead, specify #pragma pack(1) prior to the declaration and #pragma pack() after the declaration.

Using pointer data types in C and C++ applications that use SQL

You can also declare host variables that are pointers to the supported C and C++ data types, with the following restrictions.

- If a host variable is declared as a pointer, then that host variable must be declared with asterisks followed by a host variable. The following examples are all valid:

```
short *mynum;               /* Ptr to an integer                  */
long **mynumptr;            /* Ptr to a ptr to a long integer     */
char *mychar;               /* Ptr to a single character          */
char(*mychara)[20];          /* Ptr to a char array of 20 bytes    */
struct {                    /* Ptr to a variable char array of 30 */
   short mylen;             /*     bytes.                         */
   char mydata[30];
  } *myvarchar;
```

Note: Parentheses are only allowed when declaring a pointer to a NUL-terminated character array, in which case they are required. If the parentheses were not used, you would be declaring an array of pointers rather than the desired pointer to an array. For example:

```
char (*a)[10];          /* pointer to a null-terminated char array */
char *a[10];            /* pointer to an array of pointers         */
```

- If a host variable is declared as a pointer, then no other host variable can be declared with that same name within the same source file. For example, the second declaration below would be invalid:

```
char *mychar;               /* This declaration is valid            */
char mychar;                /* But this one is invalid              */
```

- When a host variable is referenced within an SQL statement, that host variable must be referenced exactly as declared, with the exception of pointers to NUL-terminated character arrays. For example, the following declaration required parentheses:

  ```
  char (*mychara)[20];        /* ptr to char array of 20 bytes        */
  ```

  However, the parentheses are not allowed when the host variable is referenced in an SQL statement, such as a SELECT:

  ```
  EXEC SQL SELECT name INTO  :*mychara FROM mytable;
  ```

- Only the asterisk can be used as an operator over a host variable name.
- The maximum length of a host variable name is affected by the number of asterisks specified, as these asterisks are considered part of the name.
- Pointers to structures are not usable as host variables except for variable character structures. Also, pointer fields in structures are not usable as host variables.
- SQL requires that all specified storage for based host variables be allocated. If the storage is not allocated, unpredictable results can occur.

Using typedef in C and C++ applications that use SQL

You can also use the typedef declarations to define your own identifiers that will be used in place of C type specifiers such as short, float, and double.

The typedef identifiers used to declare host variables must be unique within the program, even if the typedef declarations are in different blocks or procedures. If the program contains BEGIN DECLARE SECTION and END DECLARE SECTION statements, the typedef declarations do not need to be contained with the BEGIN DECLARE SECTION and END DECLARE SECTION. The typedef identifier will be recognized by the SQL precompiler within the BEGIN DECLARE SECTION. The C and C++ precompilers recognize only a subset of typedef declarations, the same as with host variable declarations.

Examples of valid typedef statements:

- Declaring a long typedef and then declaring host variables which reference the typedef.

  ```
  typedef long int LONG_T;
  LONG_T I1, *I2;
  ```

- The character array length may be specified in either the typedef or on the host variable declaration but not in both.

  ```
  typedef char NAME_T[30];
  typedef char CHAR_T;
  CHAR_T name1[30];   /* Valid */
  NAME_T name2;       /* Valid */
  NAME_T name3[10];   /* Not valid for SQL use */
  ```

- The SQL TYPE IS statement may be used in a typedef.

  ```
  typedef SQL TYPE IS CLOB(5K) CLOB_T;
  CLOB_T clob_var1;
  ```

- Storage class (auto, extern, static), volatile, or const qualifiers may be specified on the host variable declaration.

  ```
  typdef short INT_T;
  typdef short INT2_T;
  static INT_T i1;
  volatile INT2_T i2;
  ```

- typedefs of structures are supported.

  ```
  typedef _Packed struct {char dept[3];
                          char deptname[30];
                          long Num_employees;} DEPT_T;
  DEPT_T dept_rec;
  DEPT_T dept_array[20];  /* use for blocked insert or fetch */
  ```

Using ILE C compiler external file descriptions in C and C++ applications that use SQL

You can use the C or C++ #pragma mapinc directive with the #include directive to include external file descriptions in your program.

When used with SQL, only a particular format of the #pragma mapinc directive is recognized by the SQL precompiler. If all of the required elements are not specified, the precompiler ignores the directive and does not generate host variable structures. The required elements are:

- Include name
- Externally described file name
- Format name or a list of format names
- Options
- Conversion options

The library name, union name, conversion options, and prefix name are optional. Although typedef statements coded by the user are not recognized by the precompiler, those created by the #pragma mapinc and #include directives are recognized. SQL supports input, output, both, and key values for the options parameter. For the conversion options, the supported values are D, p, z, _P, and 1BYTE_CHAR. These options may be specified in any order except that both D and p cannot be specified. Unions declared using the typedef union created by the #pragma mapinc and #include directive cannot be used as host variables in SQL statements; the members of the unions can be used. Structures that contain the typedef structure cannot be used in SQL statements; the structure declared using the typedef can be used.

To retrieve the definition of the sample table DEPARTMENT described in DB2 UDB for iSeries sample tables in the DB2 UDB for iSeries SQL Programming topic collection, you can code the following:

```
#pragma mapinc ("dept","CORPDATA/DEPARTMENT(*ALL)","both")
#include "dept"
CORPDATA_DEPARTMENT_DEPARTMENT_both_t Dept_Structure;
```

A host structure named Dept_Structure is defined with the following elements: DEPTNO, DEPTNAME, MGRNO, and ADMRDEPT. These field names can be used as host variables in SQL statements.

Note: DATE, TIME, and TIMESTAMP columns generate character host variable definitions. They are treated by SQL with the same comparison and assignment rules as a DATE, TIME, and TIMESTAMP column. For example, a date host variable can be compared only against a DATE column or a character string that is a valid representation of a date.

If the GRAPHIC or VARGRAPHIC column has a UCS-2 CCSID, the generated host variable will have the UCS-2 CCSID assigned to it. If the GRAPHIC or VARGRAPHIC column has a UTF-16 CCSID, the generated host variable will have the UTF-16 CCSID assigned to it.

Although zoned, binary (with nonzero scale fields), and, optionally, decimal are mapped to character fields in ILE C, SQL will treat these fields as numeric. By using the extended program model (EPM) routines, you can manipulate these fields to convert zoned and packed decimal data.

For more information, see the ILE C/C++ Language Reference topic.

Determining equivalent SQL and C or C++ data types

The precompiler determines the base SQLTYPE and SQLLEN of host variables based on the table. If a host variable appears with an indicator variable, the SQLTYPE is the base SQLTYPE plus one.

Table 1. C or C++ declarations mapped to typical SQL data types

C or C++ data type	SQLTYPE of host variable	SQLLEN of host variable	SQL data type
short int	500	2	SMALLINT
long int	496	4	INTEGER
long long int	492	8	BIGINT
decimal(p,s)	484	p in byte 1, s in byte 2	DECIMAL (p,s)
float	480	4	FLOAT (single precision)
double	480	8	FLOAT (double precision)
single-character form	452	1	CHAR(1)
NUL-terminated character form	460	length	VARCHAR (length - 1)
VARCHAR structured form	448	length	VARCHAR (length)
single-graphic form	468	1	GRAPHIC(1)
NUL-terminated single-graphic form	400	length	VARGRAPHIC (length - 1)
VARGRAPHIC structured form	464	length	VARGRAPHIC (length)

You can use the following table to determine the C or C++ data type that is equivalent to a given SQL data type.

Table 2. SQL data types mapped to typical C or C++ declarations

SQL data type	C or C++ data type	Notes
SMALLINT	short int	
INTEGER	long int	
BIGINT	long long int	
DECIMAL(p,s)	decimal(p,s)	p is a positive integer from 1 to 63, and s is a positive integer from 0 to 63.
NUMERIC(p,s) or nonzero scale binary	No exact equivalent	Use DECIMAL (p,s).
FLOAT (single precision)	float	
FLOAT (double precision)	double	
CHAR(1)	single-character form	
CHAR(n)	No exact equivalent	If *n*>1, use NUL-terminated character form.
VARCHAR(n)	NUL-terminated character form	Allow at least *n*+1 to accommodate the NUL-terminator. If data can contain character NULs (\0), use VARCHAR structured form or SQL VARCHAR. *n* is a positive integer. The maximum value of *n* is 32740.
	VARCHAR structured form	The maximum value of *n* is 32740. The SQL VARCHAR form may also be used.

Table 2. SQL data types mapped to typical C or C++ declarations (continued)

SQL data type	C or C++ data type	Notes
CLOB	None	Use SQL TYPE IS to declare a CLOB in C or C++.
GRAPHIC (1)	single-graphic form	
GRAPHIC (n)	No exact equivalent	
VARGRAPHIC(n)	NUL-terminated graphic form	If *n* > 1, use NUL-terminated graphic form.
	VARGRAPHIC structured form	If data can contain graphic NUL values (/0/0), use VARGRAPHIC structured form. Allow at least *n* + 1 to accommodate the NUL-terminator. *n* is a positive integer. The maximum value of *n* is 16370.
DBCLOB	None	Use SQL TYPE IS to declare a DBCLOB in C or C++.
BINARY	None	Use SQL TYPE IS to declare a BINARY in C or C++.
VARBINARY	None	Use SQL TYPE IS to declare a VARBINARY in C or C++.
BLOB	None	Use SQL TYPE IS to declare a BLOB in C or C++.
DATE	NUL-terminated character form	If the format is *USA, *ISO, *JIS, or *EUR, allow at least 11 characters to accommodate the NUL-terminator. If the format is *MDY, *YMD, or *DMY, allow at least 9 characters to accommodate the NUL-terminator. If the format is *JUL, allow at least 7 characters to accommodate the NUL-terminator.
	VARCHAR structured form	If the format is *USA, *ISO, *JIS, or *EUR, allow at least 10 characters. If the format is *MDY, *YMD, or *DMY, allow at least 8 characters. If the format is *JUL, allow at least 6 characters.
TIME	NUL-terminated character form	Allow at least 7 characters (9 to include seconds) to accommodate the NUL-terminator.
	VARCHAR structured form	Allow at least 6 characters; 8 to include seconds.

Table 2. SQL data types mapped to typical C or C++ declarations (continued)

SQL data type	C or C++ data type	Notes
TIMESTAMP	NUL-terminated character form	Allow at least 20 characters (27 to include microseconds at full precision) to accommodate the NUL-terminator. If n is less than 27, truncation occurs on the microseconds part.
	VARCHAR structured form	Allow at least 19 characters. To include microseconds at full precision, allow 26 characters. If the number of characters is less than 26, truncation occurs on the microseconds part.
DATALINK	Not supported	
ROWID	None	Use SQL TYPE IS to declare a ROWID in C or C++.

Notes on C and C++ variable declaration and usage

Single quotation marks (′) and quotation marks (″) have different meanings in C, C++, and SQL.

C and C++ use quotation marks to delimit string constants and single quotation marks to delimit character constants. In contrast, SQL uses quotation marks for delimited identifiers and uses single quotation marks to delimit character string constants. Character data in SQL is distinct from integer data.

Using indicator variables in C and C++ applications that use SQL

An indicator variable is a two-byte integer (short int).

You can also specify an indicator structure (defined as an array of halfword integer variables) to support a host structure. On retrieval, an indicator variable is used to show if its associated host variable has been assigned a null value. On assignment to a column, a negative indicator variable is used to indicate that a null value should be assigned.

Indicator variables are declared in the same way as host variables. The declarations of the two can be mixed in any way that seems appropriate to you.

Example

Given the statement:

```
EXEC SQL FETCH CLS_CURSOR INTO :ClsCd,
                               :Day :DayInd,
                               :Bgn :BgnInd,
                               :End :EndInd;
```

Variables can be declared as follows:

```
EXEC SQL BEGIN DECLARE SECTION;
char  ClsCd[8];
char  Bgn[9];
char  End[9];
short Day, DayInd, BgnInd, EndInd;
EXEC SQL END DECLARE SECTION;
```

Related reference

References to variables

Coding SQL statements in COBOL applications

There are unique application and coding requirements for embedding SQL statements in a COBOL program. In this topic, requirements for host structures and host variables are defined.

The System i™ products support more than one COBOL compiler. The DB2 UDB Query Manager and SQL Development Kit licensed program only supports the OPM COBOL and ILE COBOL programming languages.

Note: By using the code examples, you agree to the terms of the “Code license and disclaimer information” on page 176.

Related concepts

“Writing applications that use SQL” on page 2
You can create database applications in host languages that use DB2 UDB for iSeries SQL statements and functions.

“Error and warning messages during a compile of application programs that use SQL” on page 132
These conditions might produce an error or warning message during an attempted compile process.

Related reference

“Example programs: Using DB2 UDB for iSeries statements” on page 136
Here is a sample application that shows how to code SQL statements in each of the languages that DB2 UDB for iSeries supports.

Defining the SQL communication area in COBOL applications that use SQL

A COBOL program can be written to use the SQL communication area (SQLCA) to check return status for embedded SQL statements, or the program can use the SQL diagnostics area to check return status.

To use the SQL diagnostics area instead of the SQLCA, use the SET OPTION SQL statement with the option SQLCA = *NO.

When using the SQLCA, a COBOL program that contains SQL statements must include one or both of the following:

- An SQLCODE variable declared as PICTURE S9(9) BINARY, PICTURE S9(9) COMP-4, or PICTURE S9(9) COMP.
- An SQLSTATE variable declared as PICTURE X(5).

Or,

- An SQLCA (which contains an SQLCODE and SQLSTATE variable).

The SQLCODE and SQLSTATE values are set by the database manager after each SQL statement is run. An application can check the SQLCODE or SQLSTATE value to determine whether the last SQL statement was successful.

The SQLCA can be coded in a COBOL program either directly or by using the SQL INCLUDE statement. When coding it directly, make sure it is initialized. Using the SQL INCLUDE statement requests the inclusion of a standard declaration:

```
EXEC SQL INCLUDE SQLCA END-EXEC.
```

The SQLCODE, SQLSTATE, and SQLCA variable declarations must appear in the WORKING-STORAGE SECTION or LINKAGE SECTION of your program and can be placed wherever a record description entry can be specified in those sections.

When you use the INCLUDE statement, the SQL COBOL precompiler includes COBOL source statements for the SQLCA:

```
01 SQLCA.
   05 SQLCAID       PIC X(8). VALUE X"0000000000000000".
   05 SQLCABC       PIC S9(9) BINARY.
   05 SQLCODE       PIC S9(9) BINARY.
   05 SQLERRM.
      49 SQLERRML   PIC S9(4) BINARY.
      49 SQLERRMC   PIC X(70).
   05 SQLERRP       PIC X(8).
   05 SQLERRD       OCCURS 6 TIMES
                    PIC S9(9) BINARY.
   05 SQLWARN.
      10 SQLWARN0   PIC X.
      10 SQLWARN1   PIC X.
      10 SQLWARN2   PIC X.
      10 SQLWARN3   PIC X.
      10 SQLWARN4   PIC X.
      10 SQLWARN5   PIC X.
      10 SQLWARN6   PIC X.
      10 SQLWARN7   PIC X.
      10 SQLWARN8   PIC X.
      10 SQLWARN9   PIC X.
      10 SQLWARNA   PIC X.
   05 SQLSTATE      PIC X(5).
```

For ILE COBOL, the SQLCA is declared using the GLOBAL clause. SQLCODE is replaced with SQLCADE when a declaration for SQLCODE is found in the program and the SQLCA is provided by the precompiler. SQLSTATE is replaced with SQLSTOTE when a declaration for SQLSTATE is found in the program and the SQLCA is provided by the precompiler.

Related concepts

"Using the SQL diagnostics area" on page 9
The SQL diagnostics area is used to keep the returned information for an SQL statement that has been run in a program. It contains all the information that is available to you as an application programmer through the SQLCA.

Related reference

SQL communication area

Defining SQL descriptor areas in COBOL applications that use SQL

There are two types of SQL descriptor areas (SQLDAs). One is defined with the ALLOCATE DESCRIPTOR statement. The other is defined using the SQLDA structure. In this topic, only the SQLDA form is discussed.

The following statements can use an SQLDA:

- EXECUTE...USING DESCRIPTOR *descriptor-name*
- FETCH...USING DESCRIPTOR *descriptor-name*
- OPEN...USING DESCRIPTOR *descriptor-name*
- CALL...USING DESCRIPTOR *descriptor-name*
- DESCRIBE *statement-name* INTO *descriptor-name*
- DESCRIBE INPUT *statement-name* INTO *descriptor-name*
- DESCRIBE TABLE *host-variable* INTO *descriptor-name*
- PREPARE *statement-name* INTO *descriptor-name*

Unlike the SQLCA, there can be more than one SQLDA in a program. The SQLDA can have any valid name. An SQLDA can be coded in a COBOL program directly or added with the INCLUDE statement. Using the SQL INCLUDE statement requests the inclusion of a standard SQLDA declaration:

```
EXEC SQL INCLUDE SQLDA END-EXEC.
```

The COBOL declarations included for the SQLDA are:

```
1 SQLDA.
  05 SQLDAID     PIC X(8).
  05 SQLDABC     PIC S9(9) BINARY.
  05 SQLN        PIC S9(4) BINARY.
  05 SQLD        PIC S9(4) BINARY.
  05 SQLVAR OCCURS 0 TO 409 TIMES DEPENDING ON SQLD.
     10 SQLTYPE   PIC S9(4) BINARY.
     10 SQLLEN    PIC S9(4) BINARY.
     10 FILLER  REDEFINES SQLLEN.
        15 SQLPRECISION PIC X.
        15 SQLSCALE     PIC X.
     10 SQLRES    PIC X(12).
     10 SQLDATA   POINTER.
     10 SQLIND    POINTER.
     10 SQLNAME.
        49 SQLNAMEL PIC S9(4) BINARY.
        49 SQLNAMEC PIC X(30).
```

Figure 1. INCLUDE SQLDA declarations for COBOL

SQLDA declarations must appear in the WORKING-STORAGE SECTION or LINKAGE SECTION of your program and can be placed wherever a record description entry can be specified in those sections. For ILE COBOL, the SQLDA is declared using the GLOBAL clause.

Dynamic SQL is an advanced programming technique. With dynamic SQL, your program can develop and then run SQL statements while the program is running. A SELECT statement with a variable SELECT list (that is, a list of the data to be returned as part of the query) that runs dynamically requires an SQL descriptor area (SQLDA). This is because you cannot know in advance how many or what type of variables to allocate in order to receive the results of the SELECT.

Related concepts

Dynamic SQL applications

Related reference

SQL descriptor area

Embedding SQL statements in COBOL applications that use SQL

SQL statements can be coded in COBOL program sections as in this table.

SQL statement	Program section
BEGIN DECLARE SECTION	WORKING-STORAGE SECTION or LINKAGE SECTION
END DECLARE SECTION	
DECLARE VARIABLE	
DECLARE STATEMENT	

SQL statement	Program section
INCLUDE SQLCA	WORKING-STORAGE SECTION or LINKAGE SECTION
INCLUDE SQLDA	
INCLUDE member-name	DATA DIVISION or PROCEDURE DIVISION
Other	PROCEDURE DIVISION

Each SQL statement in a COBOL program must begin with EXEC SQL and end with END-EXEC. If the SQL statement appears between two COBOL statements, the period is optional and might not be appropriate. The EXEC SQL keywords must appear all on one line, but the remainder of the statement can appear on the next and subsequent lines.

Example

An UPDATE statement coded in a COBOL program might be coded as follows:

```
EXEC SQL
  UPDATE DEPARTMENT
  SET MGRNO = :MGR-NUM
  WHERE DEPTNO = :INT-DEPT
END-EXEC.
```

Comments in COBOL applications that use SQL

In addition to SQL comments (--), you can include COBOL comment lines (* or / in column 7) within embedded SQL statements except between the keywords EXEC and SQL. COBOL debugging lines (D in column 7) are treated as comment lines by the precompiler.

Continuation for SQL statements in COBOL applications that use SQL

The line continuation rules for SQL statements are the same as those for other COBOL statements, except that EXEC SQL must be specified within one line.

If you continue a string constant from one line to the next, the first nonblank character in the next line must be either an apostrophe or a quotation mark. If you continue a delimited identifier from one line to the next, the first nonblank character in the next line must be either an apostrophe or a quotation mark.

Constants containing DBCS data can be continued across multiple lines by placing the shift-in character in column 72 of the continued line and the shift-out after the first string delimiter of the continuation line.

This SQL statement has a valid graphic constant of G'<AABBCCDDEEFFGGHHIIJJKK>'. The redundant shifts are removed.

```
*...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
EXEC SQL
SELECT * FROM GRAPHTAB           WHERE GRAPHCOL =  G'<AABB>
-       '<CCDDEEFFGGHHIIJJKK>'
END-EXEC.
```

Including code in COBOL applications that use SQL

SQL statements or COBOL host variable declaration statements can be included by embedding the following SQL statement in the source code where the statements are to be embedded.

```
  EXEC SQL INCLUDE member-name END-EXEC.
```

COBOL COPY statements cannot be used to include SQL statements or declarations of COBOL host variables that are referenced in SQL statements.

Margins in COBOL applications that use SQL

You must code SQL statements in columns 12 through 72. If EXEC SQL starts before the specified margin (that is, before column 12), the SQL precompiler does not recognize the statement.

Sequence numbers in COBOL applications that use SQL

The source statements generated by the SQL precompiler are generated with the same sequence number as the SQL statement.

Names in COBOL applications that use SQL

Any valid COBOL variable name can be used for a host variable and is subject to the following restrictions:

Do not use host variable names or external entry names that begin with 'SQL', 'RDI', or 'DSN'. These names are reserved for the database manager.

Using structures that contain FILLER may not work as expected in an SQL statement. It is recommended that all fields within a COBOL structure be named to avoid unexpected results.

COBOL compile-time options in COBOL applications that use SQL

The COBOL PROCESS statement can be used to specify the compile-time options for the COBOL compiler.

Although the PROCESS statement will be recognized by the COBOL compiler when it is called by the precompiler to create the program; the SQL precompiler itself does not recognize the PROCESS statement. Therefore, options that affect the syntax of the COBOL source such as APOST and QUOTE should not be specified in the PROCESS statement. Instead *APOST and *QUOTE should be specified in the OPTION parameter of the CRTSQLCBL and CRTSQLCBLI commands.

Statement labels in COBOL applications that use SQL

Executable SQL statements in the PROCEDURE DIVISION can be preceded by a paragraph name.

WHENEVER statement in COBOL applications that use SQL

The target for the GOTO clause in an SQL WHENEVER statement must be a section name or unqualified paragraph name in the PROCEDURE DIVISION.

Multiple source COBOL programs and the SQL COBOL precompiler

The SQL COBOL precompiler does not support precompiling multiple source programs separated with the PROCESS statement.

Using host variables in COBOL applications that use SQL

All host variables used in SQL statements must be explicitly declared prior to their first use.

The COBOL statements that are used to define the host variables should be preceded by a BEGIN DECLARE SECTION statement and followed by an END DECLARE SECTION statement. If a BEGIN DECLARE SECTION and END DECLARE SECTION are specified, all host variable declarations used in SQL statements must be between the BEGIN DECLARE SECTION and the END DECLARE SECTION statements.

All host variables within an SQL statement must be preceded by a colon (:).

Host variables cannot be records or elements.

To accommodate using dashes within a COBOL host variable name, blanks must precede and follow a minus sign.

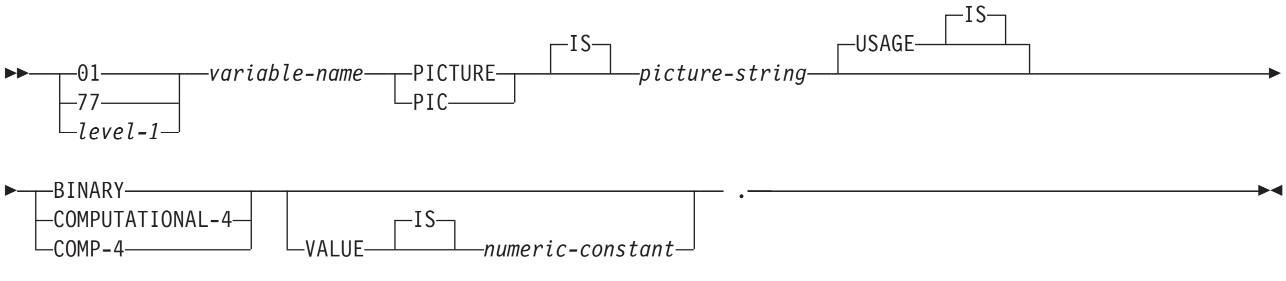
Declaring host variables in COBOL applications that use SQL

The COBOL precompiler only recognizes a subset of valid COBOL declarations as valid host variable declarations.

Numeric host variables in COBOL applications that use SQL:

This figure shows the syntax for valid integer host variable declarations.

BIGINT and INTEGER and SMALLINT

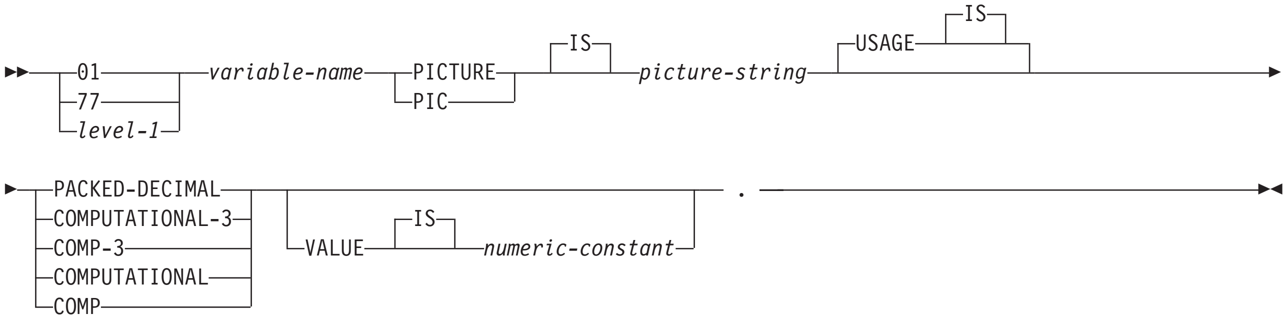


Notes:

1. BINARY, COMPUTATIONAL-4, and COMP-4 are equivalent. A portable application should code BINARY, because COMPUTATIONAL-4 and COMP-4 are IBM extensions that are not supported in International Organization for Standardization (ISO)/ANSI COBOL. The *picture-string* associated with these types must have the form S9(i)V9(d) (or S9...9V9...9, with *i* and *d* instances of 9). i + d must be less than or equal to 18.
2. level-1 indicates a COBOL level between 2 and 48.

The following figure shows the syntax for valid decimal host variable declarations.

DECIMAL



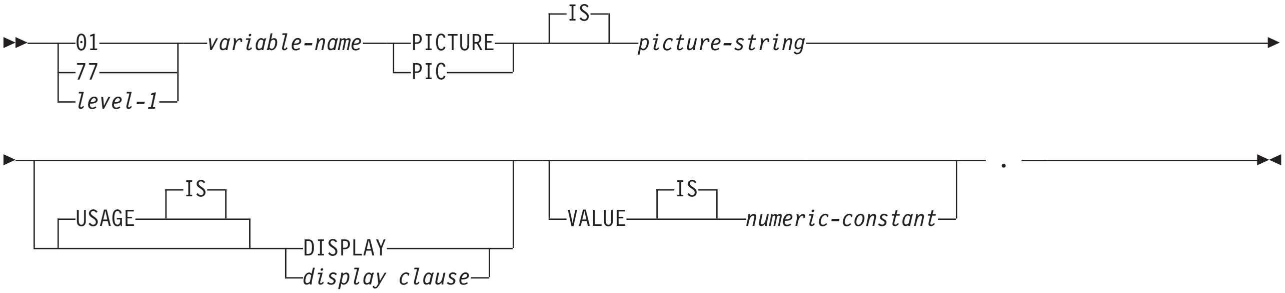
Notes:

1. PACKED-DECIMAL, COMPUTATIONAL-3, and COMP-3 are equivalent. A portable application should code PACKED-DECIMAL, because COMPUTATIONAL-3 and COMP-3 are IBM extensions that are not supported in ISO/ANS COBOL. The *picture-string* associated with these types must have the form S9(i)V9(d) (or S9...9V9...9, with *i* and *d* instances of 9). i + d must be less than or equal to 63.

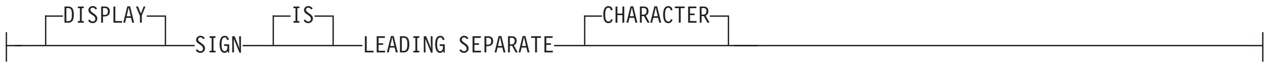
2. COMPUTATIONAL and COMP are equivalent. The picture strings associated with these and the data types they represent are product-specific. Therefore, COMP and COMPUTATIONAL should not be used in a portable application. In an OPM COBOL program, the *picture-string* associated with these types must have the form S9(i)V9(d) (or S9...9V9...9, with *i* and *d* instances of 9). i + d must be less than or equal to 63.
3. level-1 indicates a COBOL level between 2 and 48.

The following figure shows the syntax for valid numeric host variable declarations.

Numeric



display clause:



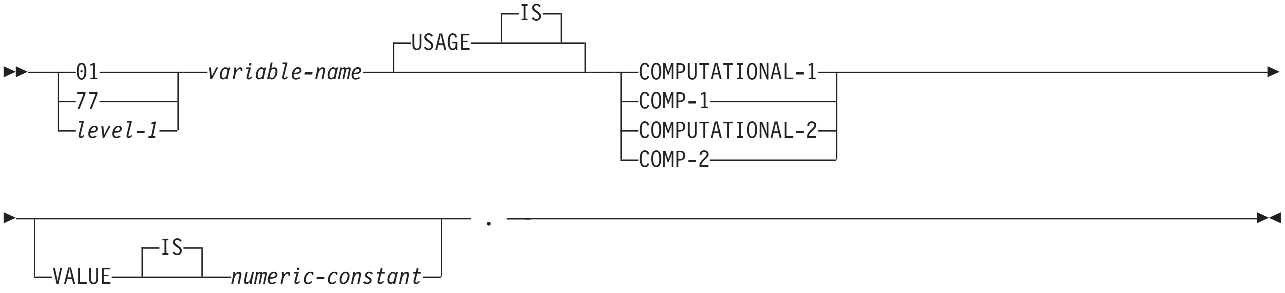
Notes:

1. The *picture-string* associated with SIGN LEADING SEPARATE and DISPLAY must have the form S9(i)V9(d) (or S9...9V9...9, with *i* and *d* instances of 9). i + d must be less than or equal to 18.
2. level-1 indicates a COBOL level between 2 and 48.

Floating-point host variables in COBOL applications that use SQL:

This figure shows the syntax for valid floating-point host variable declarations. Floating-point host variables are only supported for ILE COBOL.

Floating-point



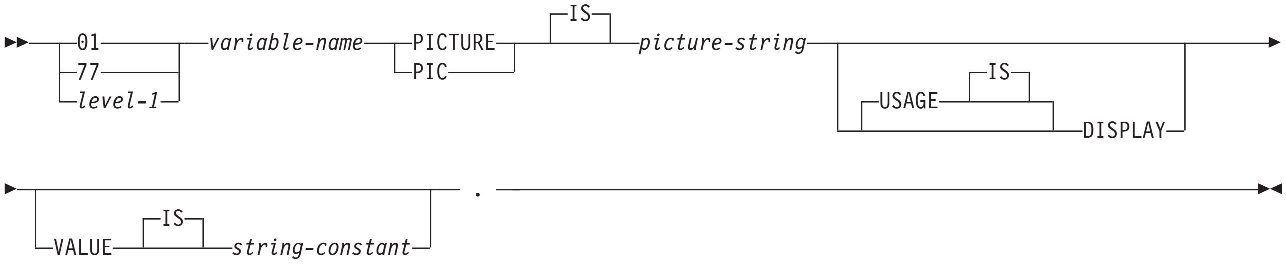
Notes:

1. COMPUTATIONAL-1 and COMP-1 are equivalent. COMPUTATIONAL-2 and COMP-2 are equivalent.
2. level-1 indicates a COBOL level between 2 and 48.

Character host variables in COBOL applications that use SQL:

There are two valid forms of character host variables: fixed-length strings and varying-length strings.

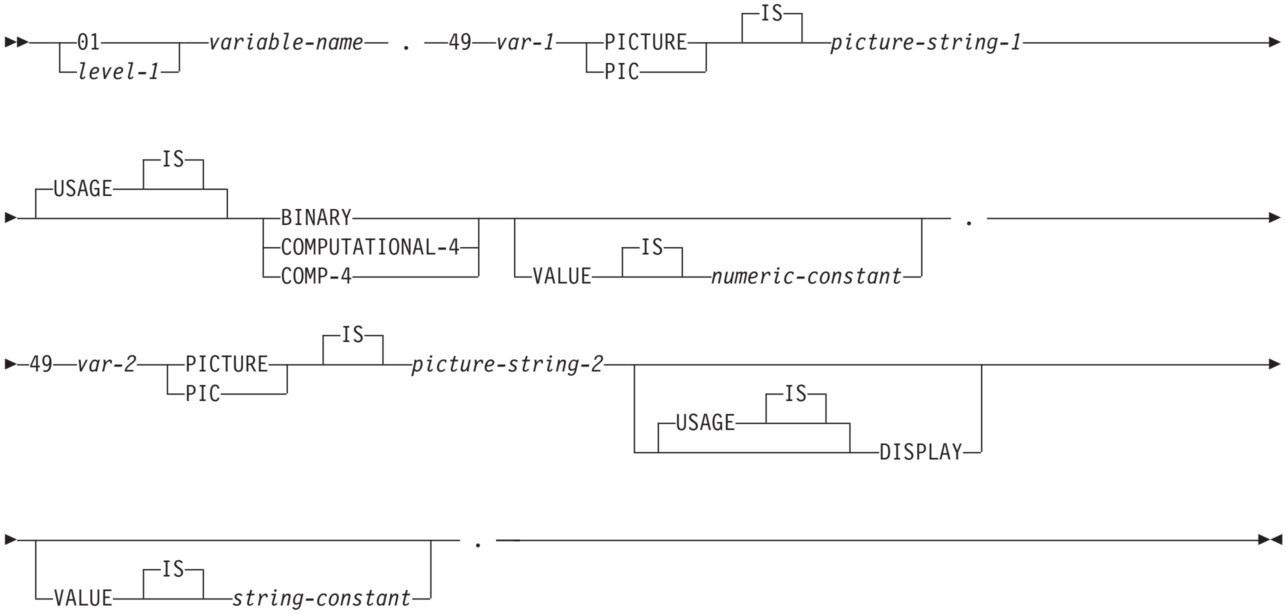
Fixed-length character strings



Notes:

1. The *picture string* associated with these forms must be X(m) (or XXX...X, with m instance of X) with 1 ≤ m ≤ 32 766.
2. level-1 indicates a COBOL level between 2 and 48.

Varying-length character strings



Notes:

1. The *picture-string-1* associated with these forms must be S9(m) or S9...9 with m instances of 9. m must be from 1 to 4.

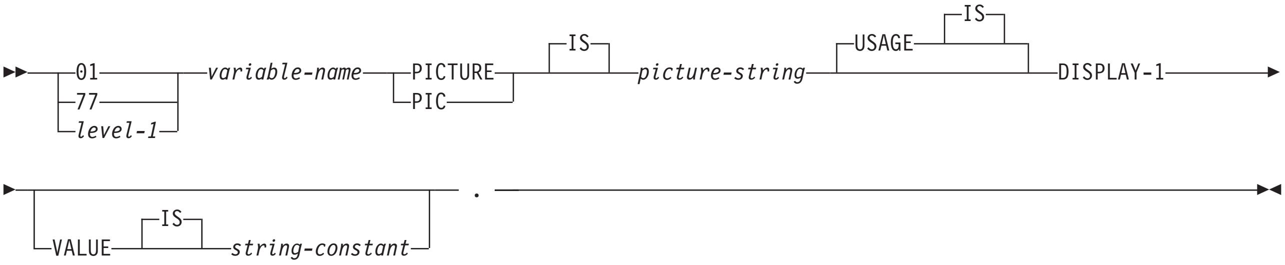
   Note that the database manager uses the full size of the S9(m) variable even though OPM COBOL only recognizes values up to the specified precision. This can cause data truncation errors when COBOL statements are being run, and might effectively limit the maximum length of variable-length character strings to the specified precision.
2. The *picture-string-2* associated with these forms must be either X(m), or XX...X, with m instances of X, and with 1 ≤ m ≤ 32 740.
3. *var-1* and *var-2* cannot be used as host variables.
4. level-1 indicates a COBOL level between 2 and 48.

Graphic host variables in COBOL applications that use SQL:

Graphic host variables are only supported in ILE COBOL.

There are two valid forms of graphic host variables:
- Fixed-Length Graphic Strings
- Varying-Length Graphic Strings

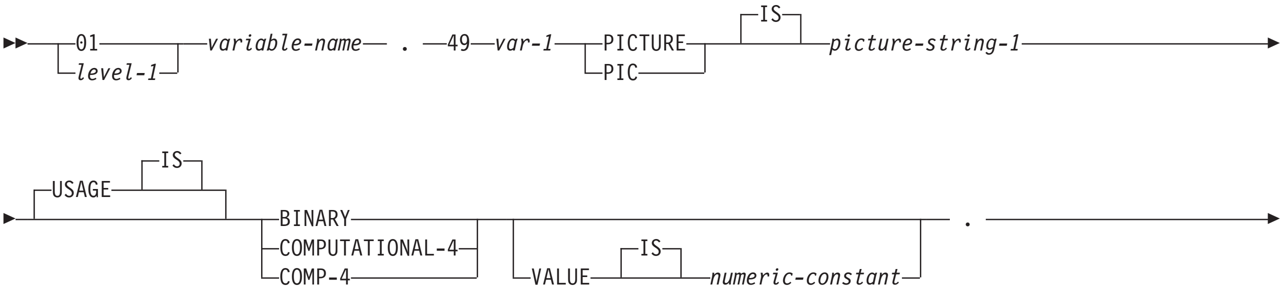
Fixed-length graphic strings



Notes:

1. The *picture string* associated with these forms must be G(m) (or GGG...G, with m instance of G) or N(m) (or NNN...N, with m instance of N) with 1 ≤ m ≤ 16 383.
2. level-1 indicates a COBOL level between 2 and 48.

Varying-length graphic strings



```
                                                      ┌─IS─┐
                        ┌─IS─┐                  ┌─USAGE─┴────┴─┐
►─49─var-2─┬─PICTURE─┬─┴────┴─picture-string-2─┴──────────────┴─DISPLAY-1──────────►
           └─PIC─────┘

►─┬──────────────────────────────────┬─ . ───────────────────────────────────────►◄
  │        ┌─IS─┐                    │
  └─VALUE──┴────┴──string-constant───┘
```

Notes:

1. The *picture-string-1* associated with these forms must be S9(m) or S9...9 with m instances of 9. m must be from 1 to 4.

   Note that the database manager uses the full size of the S9(m) variable even though OPM COBOL only recognizes values up to the specified precision. This can cause data truncation errors when COBOL statements are being run, and might effectively limit the maximum length of variable-length graphic strings to the specified precision.
2. The *picture-string-2* associated with these forms must be G(m), GG...G with m instances of G, N(m), or NN...N with m instances of N, and with 1 ≤ m ≤ 16 370.
3. *var-1* and *var-2* cannot be used as host variables.
4. level-1 indicates a COBOL level between 2 and 48.

Binary host variables in COBOL applications that use SQL:

COBOL does not have variables that correspond to the SQL binary data types. To create host variables that can be used with these data types, use the SQL TYPE IS clause. The SQL precompiler replaces this declaration with a COBOL language structure in the output source member.

BINARY and VARBINARY

```
                               ┌─IS─┐
                        ┌─USAGE─┴────┴─┐
►►─01─variable-name─────┴──────────────┴─SQL TYPE IS─┬─BINARY─────────┬─(─length─)─ . ──────►◄
                                                      ├─VARBINARY──────┤
                                                      └─BINARY VARYING─┘
```

Notes:

1. For BINARY host variables, the length must be in the range 1 to 32766.
2. For VARBINARY host variables, the length must be in the range 1 to 32740.
3. SQL TYPE IS, BINARY, VARBINARY, and BINARY VARYING can be in mixed case.

BINARY Example

The following declaration:

```
01 MY-BINARY SQL TYPE IS BINARY(200).
```

Results in the generation of the following code:

```
01 MY-BINARY PIC X(200).
```

VARBINARY Example

The following declaration:

```
01 MY-VARBINARY SQL TYPE IS VARBINARY(250).
```

Results in the generation of the following structure:

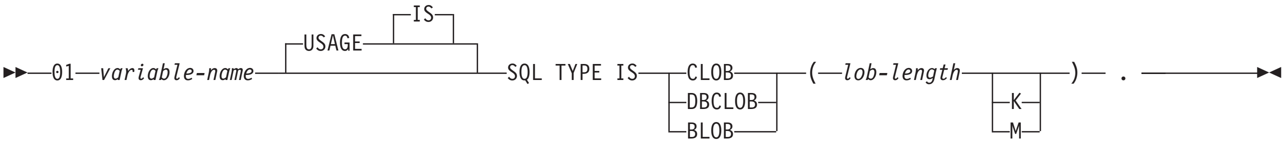
```
01 MY-VARBINARY.
 49 MY-VARBINARY-LENGTH PIC 9(5) BINARY.
 49 MY-VARBINARY-DATA PIC X(250).
```

LOB host variables in COBOL applications that use SQL:

COBOL does not have variables that correspond to the SQL data types for LOBs (large objects). To create host variables that can be used with these data types, use the SQL TYPE IS clause. The SQL precompiler replaces this declaration with a COBOL language structure in the output source member.

LOB host variables are only supported in ILE COBOL.

LOB host variables



Notes:

1. For BLOB and CLOB, 1 ≤ lob-length ≤ 15,728,640
2. For DBCLOB, 1 ≤ lob-length ≤ 7,864,320
3. SQL TYPE IS, BLOB, CLOB, DBCLOB can be in mixed case.

CLOB example

The following declaration:

```
01 MY-CLOB SQL TYPE IS CLOB(16384).
```

Results in the generation of the following structure:

```
01 MY-CLOB.
   49 MY-CLOB-LENGTH PIC 9(9) BINARY.
   49 MY-CLOB-DATA PIC X(16384).
```

DBCLOB example

The following declaration:

```
01 MY-DBCLOB SQL TYPE IS DBCLOB(8192).
```

Results in the generation of the following structure:

```
01 MY-DBCLOB.
   49 MY-DBCLOB-LENGTH PIC 9(9) BINARY.
   49 MY-DBCLOB-DATA PIC G(8192) DISPLAY-1.
```

BLOB example

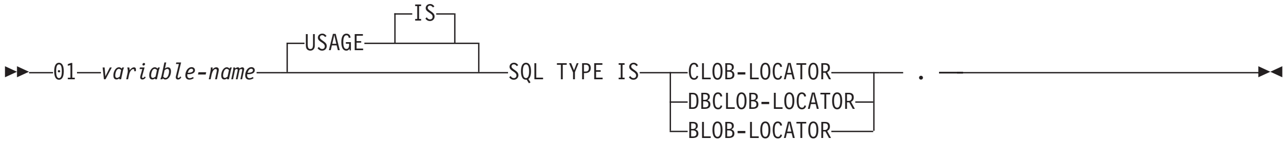
The following declaration:

```
01 MY-BLOB SQL TYPE IS BLOB(16384).
```

Results in the generation of the following structure:

```
01 MY-BLOB.
   49 MY-BLOB-LENGTH PIC 9(9) BINARY.
   49 MY-BLOB-DATA PIC X(16384).
```

LOB locator



Notes:

1. SQL TYPE IS, BLOB-LOCATOR, CLOB-LOCATOR, DBCLOB-LOCATOR can be in mixed case.
2. LOB locators cannot be initialized in the SQL TYPE IS statement.

CLOB and DBCLOB locators have similar syntax.

BLOB locator example

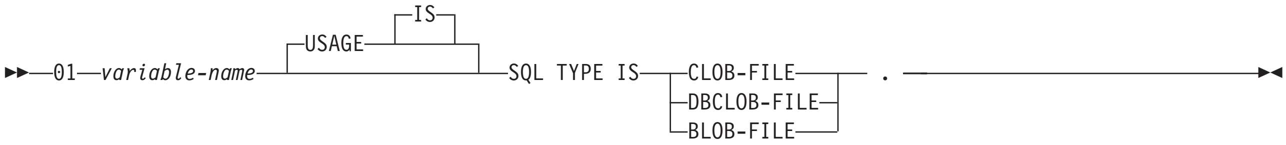
The following declaration:

```
01 MY-LOCATOR SQL TYPE IS BLOB_LOCATOR.
```

Results in the following generation:

```
01 MY-LOCATOR PIC 9(9) BINARY.
```

LOB file reference variable



Note: SQL TYPE IS, BLOB-FILE, CLOB-FILE, DBCLOB-FILE can be in mixed case.

BLOB file reference example

The following declaration:

```
01 MY-FILE SQL TYPE IS BLOB-FILE.
```

Results in the generation of the following structure:

```
01 MY-FILE.
   49 MY-FILE-NAME-LENGTH PIC S9(9) COMP-5.
   49 MY-FILE-DATA-LENGTH PIC S9(9) COMP-5.
   49 MY-FILE-FILE-OPTIONS PIC S9(9) COMP-5.
   49 MY-FILE-NAME PIC X(255).
```

CLOB and DBCLOB file reference variables have similar syntax.

The precompiler generates declarations for the following file option constants. You can use these constants to set the xxx-FILE-OPTIONS variable when you use file reference host variables.

- SQL_FILE_READ (2)
- SQL_FILE_CREATE (8)
- SQL_FILE_OVERWRITE (16)
- SQL_FILE_APPEND (32)

Related reference

LOB file reference variables

Datetime host variables in COBOL applications that use SQL:

This figure shows the syntax for valid date, time, and timestamp host variable declarations. Datetime host variables are supported only for ILE COBOL.

Datetime host variable

```
                                       ┌─OF─┐                      ┌─IS─┐
►►──┬─01──────┬──variable-name──FORMAT─┴────┴──┬─DATE──────┬──────┴────┴──format-options──────────────►◄
    ├─77──────┤                                ├─TIME──────┤
    └─level-1─┘                                └─TIMESTAMP─┘
```

Notes:

1. *level-1* indicates a COBOL level between 2 and 48.
2. *format-options* indicates valid datetime options that are supported by the COBOL compiler. See the ILE COBOL Language Reference manual for details.

ROWID host variables in COBOL applications that use SQL:

COBOL does not have a variable that corresponds to the SQL data type ROWID. To create host variables that can be used with this data type, use the SQL TYPE IS clause. The SQL precompiler replaces this declaration with a COBOL language structure in the output source member.

ROWID

```
►►──01──variable-name──SQL TYPE IS ROWID── . ─────────────────────────────────────────────────────────►◄
```

Note: SQL TYPE IS ROWID can be in mixed case.

ROWID example

The following declaration:

```
01 MY-ROWID SQL TYPE IS ROWID.
```

Results in the generation of the following structure:

```
01 MY-ROWID.
   49 MY-ROWID-LENGTH PIC 9(2) BINARY.
   49 MY-ROWID-DATA PIC X(40).
```

Using host structures in COBOL applications that use SQL

A *host structure* is a named set of host variables that is defined in your program's DATA DIVISION.

Host structures have a maximum of two levels, even though the host structure might itself occur within a multilevel structure. An exception is the declaration of a varying-length character string, which requires another level that must be level 49.

A host structure name can be a group name whose subordinate levels name basic data items. For example:

```
01 A
  02 B
    03 C1 PICTURE ...
    03 C2 PICTURE ...
```

In this example, B is the name of a host structure consisting of the basic items C1 and C2.

When writing an SQL statement using a qualified host variable name (for example, to identify a field within a structure), use the name of the structure followed by a period and the name of the field. For example, specify B.C1 rather than C1 OF B or C1 IN B. However, this guideline applies only to qualified names within SQL statements; you cannot use this technique for writing qualified names in COBOL statements.

A host structure is considered complete if any of the following items are found:

- A COBOL item that must begin in area A
- Any SQL statement (except SQL INCLUDE)

After the host structure is defined, you can refer to it in an SQL statement instead of listing the several host variables (that is, the names of the data items that comprise the host structure).

For example, you can retrieve all column values from selected rows of the table CORPDATA.EMPLOYEE with:

```
01 PEMPL.
    10 EMPNO              PIC X(6).
    10 FIRSTNME.
       49 FIRSTNME-LEN    PIC S9(4) USAGE BINARY.
       49 FIRSTNME-TEXT   PIC X(12).
    10 MIDINIT            PIC X(1).
    10 LASTNAME.
       49 LASTNAME-LEN    PIC S9(4) USAGE BINARY.
       49 LASTNAME-TEXT   PIC X(15).
    10 WORKDEPT           PIC X(3).
...
MOVE "000220" TO EMPNO.
...
EXEC SQL
 SELECT *
   INTO :PEMPL
   FROM CORPDATA.EMPLOYEE
   WHERE EMPNO = :EMPNO
END-EXEC.
```

Notice that in the declaration of PEMPL, two varying-length string elements are included in the structure: FIRSTNME and LASTNAME.

Host structure in COBOL applications that use SQL

This figure shows the syntax for the valid host structure.

```
►►──level-1──variable-name──.──────────────────────────────────────────────────►


   ┌─────────────────────────────────────────────────────────────────┐
                                        ┌─IS─┐                        │
►──▼─level-2──var-1─────┬─PICTURE─┬────┴────┴─picture-string──usage-clause──.─┬─┴──────────────►◄
                        │ └─PIC─────┘                                       │
                        ├─floating-point──.─────────────────────────────────┤
                        ├─.──varchar-string──.──────────────────────────────┤
                        ├─.──vargraphic-string──.───────────────────────────┤
                        ├─lob──.────────────────────────────────────────────┤
                        ├─datetime──.───────────────────────────────────────┤
                        ├─rowid──.──────────────────────────────────────────┤
                        └─binary──.─────────────────────────────────────────┘
```

floating-point:

```
├──┬──────────────────────────────────────┬──┬───────────────────────────┬──────────────────┤
   │          ┌─IS─┐                        │  │          ┌─IS─┐           │
   │ ┌─USAGE──┴────┴─┐                      │  └─VALUE────┴────┴─constant─┘
   └─┴───────────────┴─┬─COMPUTATIONAL-1─┬──┘
                       ├─COMP-1──────────┤
                       ├─COMPUTATIONAL-2─┤
                       └─COMP-2──────────┘
```

usage-clause:

```
├──┬───────────────────────────────────────────┬──┬───────────────────────────┬──────────────┤
   │          ┌─IS─┐                             │  │          ┌─IS─┐           │
   │ ┌─USAGE──┴────┴─┐                           │  └─VALUE────┴────┴─constant─┘
   └─┴───────────────┴─┬─┬─BINARY──────────┬─┬───┘
                       │ ├─COMPUTATIONAL-4─┤ │
                       │ └─COMP-4──────────┘ │
                       ├─┬─PACKED-DECIMAL──┬─┤
                       │ ├─COMPUTATIONAL-3─┤ │
                       │ ├─COMP-3──────────┤ │
                       │ ├─COMPUTATIONAL───┤ │
                       │ └─COMP────────────┘ │
                       ├─┬─DISPLAY────────┬──┤
                       │ └─display-clause─┘  │
                       └─DISPLAY-1───────────┘
```

display-clause:

```
   ┌─DISPLAY─┐          ┌─IS─┐                               ┌─CHARACTER─┐
├──┴─────────┴──SIGN────┴────┴──LEADING──SEPARATE────┴───────────┴──────────────────┤
```

varchar-string:

```
                             .-IS-.
|--49--var-2--+-PICTURE-+--+----+--picture-string-1------------------------------->
              '-PIC-----'

               .-IS-.
   .-USAGE--+----+-.
>--+------------------+--+-BINARY----------+--+----------------------------------+--.-->
                          +-COMPUTATIONAL-4-+  |           .-IS-.                 |
                          '-COMP-4----------'  '-VALUE--+----+--numeric-constant-'

                           .-IS-.
>--49--var-3--+-PICTURE-+--+----+--picture-string-2------------------------------->
              '-PIC-----'

>--+--------------------------------+--+-------------------------+----------------|
   |            .-IS-.              |  |          .-IS-.         |
   |  .-USAGE--+----+-.             |  '-VALUE--+----+--constant-'
   '--+------------------+--DISPLAY-'
```

vargraphic-string:

```
                                                              .-IS-.
                             .-IS-.                  .-USAGE--+----+-.
|--49--var-2--+-PICTURE-+--+----+--picture-string-1--+------------------+--+-BINARY----------+-->
              '-PIC-----'                                                   +-COMPUTATIONAL-4-+
                                                                            '-COMP-4----------'

>--+----------------------------------+--.---------------------------------------->
   |          .-IS-.                  |
   '-VALUE--+----+--numeric-constant-'

                                                              .-IS-.
                           .-IS-.                   .-USAGE--+----+-.
>--49--var-3--+-PICTURE-+--+----+--picture-string-2--+------------------+--DISPLAY-1-->
              '-PIC-----'

>--+--------------------------+-------------------------------------------------|
   |          .-IS-.          |
   '-VALUE--+----+--constant-'
```

lob:

```
            ┌─IS─┐
   ┌─USAGE──┴────┘┐
├──┴──────────────┴──SQL TYPE IS──┬─┬─CLOB───┬──(──lob-length──┬───┬──)─┬────────────────────┤
                                  │ ├─DBCLOB─┤                 ├─K─┤    │
                                  │ └─BLOB───┘                 └─M─┘    │
                                  ├─┬─CLOB-LOCATOR───┬──────────────────┤
                                  │ ├─DBCLOB-LOCATOR─┤                  │
                                  │ └─BLOB-LOCATOR───┘                  │
                                  └─┬─CLOB-FILE───┬─────────────────────┘
                                    ├─DBCLOB-FILE─┤
                                    └─BLOB-FILE───┘
```

datetime:

```
                           ┌─OF─┐             ┌─IS─┐
├──variable-name──FORMAT───┴────┴──┬─DATE──────┬──┴────┴──format-options───────────────────────┤
                                   ├─TIME──────┤
                                   └─TIMESTAMP─┘
```

rowid:

```
├──SQL TYPE IS ROWID───────────────────────────────────────────────────────────────────────────┤
```

binary:

```
            ┌─IS─┐
   ┌─USAGE──┴────┘┐
├──┴──────────────┴──SQL TYPE IS──┬─BINARY─────────┬──(──length──)──────────────────────────────┤
                                  ├─VARBINARY──────┤
                                  └─BINARY VARYING─┘
```

Notes:

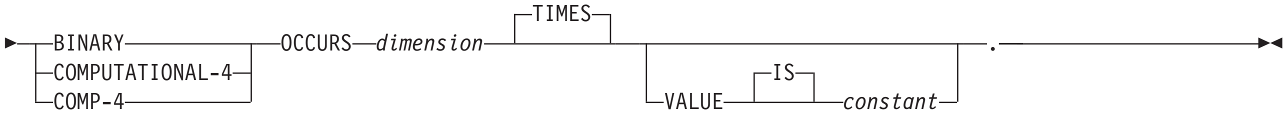
1. level-1 indicates a COBOL level between 1 and 47.
2. level-2 indicates a COBOL level between 2 and 48 where level-2 > level-1.
3. Graphic host variables, LOB host variables, and floating-point host variables are only supported for ILE COBOL.
4. For details on declaring numeric, character, graphic, LOB, ROWID, and binary host variables, see the notes under numeric-host variables, character-host variables, graphic-host variables, LOB host variables, ROWID, and binary host variables.
5. *format-options* indicates valid datetime options that are supported by the COBOL compiler. See the ILE COBOL Language Reference manual for details.

Host structure indicator array in COBOL applications that use SQL

This figure shows the syntax for valid host structure indicator array declarations.

Host structure indicator array

```
                                                                        ┌─IS─┐
                                          ┌─IS─┐                 ┌─USAGE──┴────┘┐
►►──level-1──variable-name──┬─PICTURE─┬──┴────┴──picture-string──┴──────────────┴────────────►
                            └─PIC─────┘
```



Notes:

1. Dimension must be an integer between 1 and 32767.
2. level-1 must be an integer between 2 and 48.
3. BINARY, COMPUTATIONAL-4, and COMP-4 are equivalent. A portable application should code BINARY because COMPUTATIONAL-4 and COMP-4 are IBM extensions that are not supported in ISO/ANSI COBOL. The *picture-string* associated with these types must have the form S9(*i*) (or S9...9, with *i* instances of 9). *i* must be less than or equal to 4.

Using host structure arrays in COBOL applications that use SQL

A host structure array is a named set of host variables that is defined in the program's Data Division and has an OCCURS clause.

Host structure arrays have a maximum of two levels, even though the host structure can occur within a multiple level structure. A varying-length string requires another level, level 49. A host structure array name can be a group name whose subordinate levels name basic data items.

In these examples, the following are true:

- All members in B-ARRAY must be valid.
- B-ARRAY cannot be qualified.
- B-ARRAY can only be used on the blocked form of the FETCH and INSERT statements.
- B-ARRAY is the name of an array of host structures containing items C1-VAR and C2-VAR.
- The SYNCHRONIZED attribute must not be specified.
- C1-VAR and C2-VAR are not valid host variables in any SQL statement. A structure cannot contain an intermediate level structure.

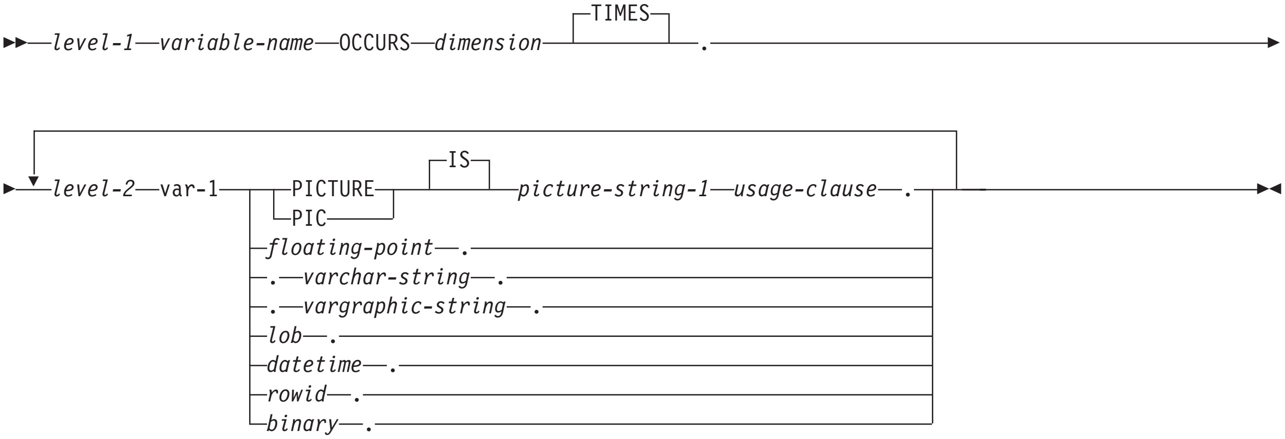
```
01  A-STRUCT.
    02 B-ARRAY OCCURS 10 TIMES.
       03 C1-VAR PIC X(20).
       03 C2-VAR PIC S9(4).
```

To retrieve 10 rows from the CORPDATA.DEPARTMENT table, use the following example:

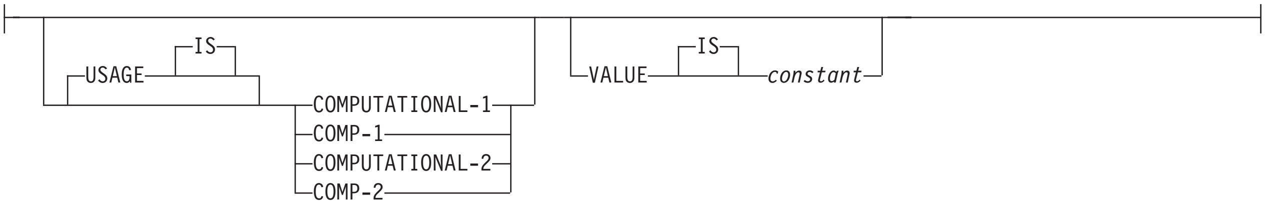
```
01  TABLE-1.
    02 DEPT OCCURS 10 TIMES.
       05 DEPTNO PIC X(3).
       05 DEPTNAME.
          49 DEPTNAME-LEN PIC S9(4) BINARY.
          49 DEPTNAME-TEXT PIC X(29).
      05 MGRNO PIC X(6).
      05 ADMRDEPT PIC X(3).
01  TABLE-2.
    02 IND-ARRAY OCCURS 10 TIMES.
       05 INDS PIC S9(4) BINARY OCCURS 4 TIMES.
....
EXEC SQL
DECLARE C1 CURSOR FOR
   SELECT *
   FROM CORPDATA.DEPARTMENT
END-EXEC.
....
EXEC SQL
   FETCH C1 FOR 10 ROWS INTO :DEPT :IND-ARRAY
END-EXEC.
```

Host structure array in COBOL applications that use SQL

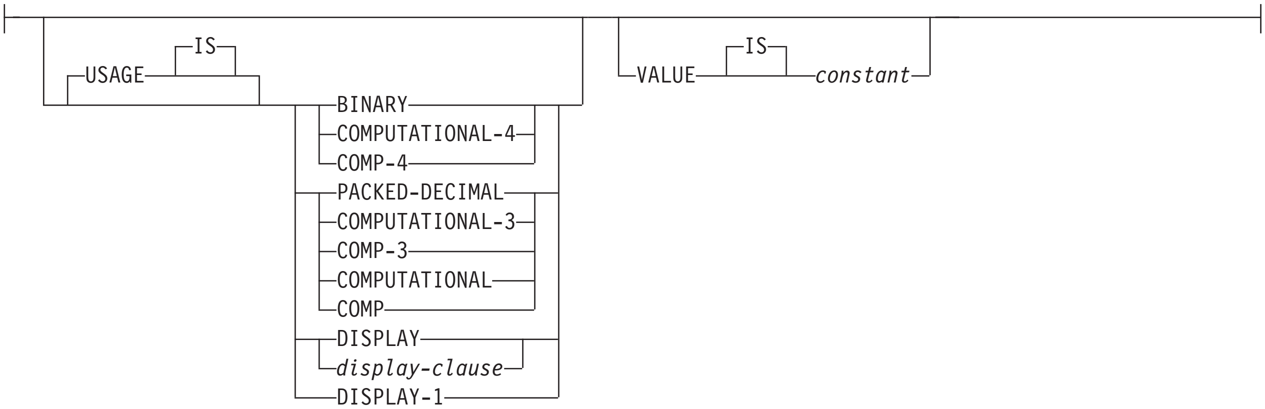
These figures show the syntax for valid host structure array declarations.



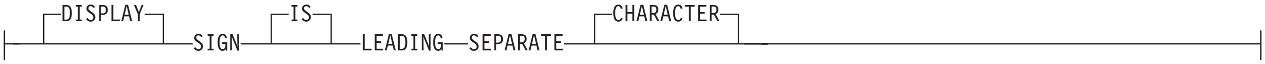
floating-point:



usage-clause:



display-clause:



varchar-string:

```
                                                          ┌─IS─┐
                          ┌─IS─┐                  ┌─USAGE─┴────┴─┐
├──49──var-2──┬─PICTURE─┬─┴────┴─picture-string-2─┴──────────────┴─┬─BINARY──────────┬──────────►
              └─PIC─────┘                                          ├─COMPUTATIONAL-4─┤
                                                                   └─COMP-4──────────┘

►──┬────────────────────────────────┬─.──────────────────────────────────────────────────────────►
   │        ┌─IS─┐                  │
   └─VALUE──┴────┴─numeric-constant─┘

                          ┌─IS─┐
►──49──var-3──┬─PICTURE─┬─┴────┴─picture-string-3─┬──────────────────────────┬───────────────────►
              └─PIC─────┘                         │         ┌─IS─┐           │
                                                  │ ┌─USAGE─┴────┴─┐         │
                                                  └─┴──────────────┴─DISPLAY─┘

►──┬────────────────────────┬───────────────────────────────────────────────────────────────────┤
   │        ┌─IS─┐          │
   └─VALUE──┴────┴─constant─┘
```

vargraphic-string:

```
                                                          ┌─IS─┐
                          ┌─IS─┐                  ┌─USAGE─┴────┴─┐
├──49──var-2──┬─PICTURE─┬─┴────┴─picture-string-2─┴──────────────┴─┬─BINARY──────────┬──────────►
              └─PIC─────┘                                          ├─COMPUTATIONAL-4─┤
                                                                   └─COMP-4──────────┘

►──┬────────────────────────────────┬─.──────────────────────────────────────────────────────────►
   │        ┌─IS─┐                  │
   └─VALUE──┴────┴─numeric-constant─┘

                                                          ┌─IS─┐
                          ┌─IS─┐                  ┌─USAGE─┴────┴─┐
►──49──var-3──┬─PICTURE─┬─┴────┴─picture-string-3─┴──────────────┴─DISPLAY-1─────────────────────►
              └─PIC─────┘

►──┬────────────────────────┬───────────────────────────────────────────────────────────────────┤
   │        ┌─IS─┐          │
   └─VALUE──┴────┴─constant─┘
```

lob:

```
               ┌─IS─┐
     ┌─USAGE───┴────┴─┐
├────┴────────────────┴──SQL TYPE IS──┬─┬─CLOB────┬──(─lob-length─┬───┬─)─┬──────────────────────┤
                                      │ ├─DBCLOB──┤               ├─K─┤   │
                                      │ └─BLOB────┘               └─M─┘   │
                                      ├─┬─CLOB-LOCATOR────┬───────────────┤
                                      │ ├─DBCLOB-LOCATOR──┤               │
                                      │ └─BLOB-LOCATOR────┘               │
                                      └─┬─CLOB-FILE────┬──────────────────┘
                                        ├─DBCLOB-FILE──┤
                                        └─BLOB-FILE────┘
```

datetime:

```
                               ┌─OF─┐                       ┌─IS─┐
├──variable-name──FORMAT───────┴────┴──┬─DATE────────┬──────┴────┴──format-options──────────────┤
                                       ├─TIME────────┤
                                       └─TIMESTAMP───┘
```

rowid:

```
├──SQL TYPE IS ROWID──────────────────────────────────────────────────────────────────────────┤
```

binary:

```
               ┌─IS─┐
     ┌─USAGE───┴────┴─┐
├────┴────────────────┴──SQL TYPE IS──┬─BINARY──────────┬──(──length──)─────────────────────────┤
                                      ├─VARBINARY───────┤
                                      └─BINARY VARYING──┘
```

Notes:

1. level-1 indicates a COBOL level between 2 and 47.
2. level-2 indicates a COBOL level between 3 and 48 where level-2 > level-1.
3. Graphic host variables, LOB host variables, and floating-point host variables are only supported for ILE COBOL.
4. For details on declaring numeric, character, graphic, LOB, ROWID, and binary host variables, see the notes under numeric-host variables, character-host variables, graphic-host variables, LOB, ROWID, and binary host variables.
5. Dimension must be an integer constant between 1 and 32767.
6. *format-options* indicates valid datetime options that are supported by the COBOL compiler. See the ILE COBOL Language Reference manual for details.

Host array indicator structure in COBOL applications that use SQL

This figure shows the valid syntax for host structure array indicators.

```
                                                ┌─TIMES─┐
►►──level-1──variable-name──OCCURS──dimension───┴───────┴──.──────────────────────────────────►
```

```
                                                          ┌─IS─┐
                           ┌─IS─┐                ┌─USAGE──┴────┴─┐
►─level-2─var-1──┬─PICTURE─┬──┴────┴──picture-string──┴───────────────┴──┬─BINARY──────────┬──►
                 └─PIC─────┘                                           ├─COMPUTATIONAL-4─┤
                                                                       └─COMP-4──────────┘

►──┬─────────────────────────┬─.──────────────────────────────────────────────────────────►◄
   │        ┌─IS─┐           │
   └─VALUE──┴────┴──constant─┘
```

Notes:

1. level-1 indicates a COBOL level between 2 and 48.
2. level-2 indicates a COBOL level between 3 and 48 where level-2 > level-1.
3. Dimension must be an integer constant between 1 and 32767.
4. BINARY, COMPUTATIONAL-4, and COMP-4 are equivalent. A portable application should code BINARY, because COMPUTATIONAL-4 and COMP-4 are IBM extensions that are not supported in ISO/ANSI COBOL. The *picture-string* associated with these types must have the form S9(i) (or S9...9, with i instances of 9). i must be less than or equal to 4.

Using external file descriptions in COBOL applications that use SQL

SQL uses the COPY DD-format-name, COPY DD-ALL-FORMATS, COPY DDS-format-name, COPY DDR-format-name, COPY DDR-ALL-FORMATS, COPY DDSR-format-name, COPY DDS-ALL-FORMATS, and COPY DDSR-ALL-FORMATS to retrieve host variables from the file definitions.

If the REPLACING option is specified, only complete name replacing is done. Var-1 is compared against the format name and the field name. If they are equal, var-2 is used as the new name.

Note: You cannot retrieve host variables from file definitions that have field names which are COBOL reserved words. You must place the COPY DDx-format statement within a COBOL host structure.

To retrieve the definition of the sample table DEPARTMENT described in DB2 UDB for iSeries sample tables in the DB2 UDB for iSeries SQL programming concepts topic collection, you can code the following:

```
01   DEPARTMENT-STRUCTURE.
     COPY DDS-ALL-FORMATS OF DEPARTMENT.
```

A host structure named DEPARTMENT-STRUCTURE is defined with an 05 level field named DEPARTMENT-RECORD that contains four 06 level fields named DEPTNO, DEPTNAME, MGRNO, and ADMRDEPT. These field names can be used as host variables in SQL statements.

For more information about the COBOL COPY verb, see the COBOL/400® User’s Guide and ILE COBOL Language Reference manuals.

Using external file descriptions for host structure arrays in COBOL applications that use SQL

Because COBOL creates an extra level when including externally described data, the OCCURS clause must be placed on the preceding 04 level. The structure cannot contain any additional declares at the 05 level.

If the file contains fields that are generated as FILLER, the structure cannot be used as a host structure array.

For device files, if INDARA is not specified and the file contains indicators, the declaration cannot be used as a host structure array. The indicator area is included in the generated structure and causes the storage for records to not be contiguous.

For example, the following shows how to use COPY–DDS to generate a host structure array and fetch 10 rows into the host structure array:

```
01  DEPT.
    04 DEPT-ARRAY OCCURS 10 TIMES.
    COPY DDS-ALL-FORMATS OF DEPARTMENT.
     ...

EXEC SQL DECLARE C1 CURSOR FOR
     SELECT * FROM CORPDATA.DEPARTMENT
END EXEC.

EXEC SQL OPEN C1
END-EXEC.

EXEC SQL FETCH C1 FOR 10 ROWS INTO :DEPARTMENT
END-EXEC.
```

Note: DATE, TIME, and TIMESTAMP columns will generate character host variable definitions that are treated by SQL with the same comparison and assignment rules as the DATE, TIME, or TIMESTAMP column. For example, a date host variable can only be compared against a DATE column or a character string which is a valid representation of a date.

Although GRAPHIC and VARGRAPHIC are mapped to character variables in OPM COBOL, SQL considers these GRAPHIC and VARGRAPHIC variables. If the GRAPHIC or VARGRAPHIC column has a UCS-2 CCSID, the generated host variable has the UCS-2 CCSID assigned to it. If the GRAPHIC or VARGRAPHIC column has a UTF-16 CCSID, the generated host variable has the UTF-16 CCSID assigned to it.

Determining equivalent SQL and COBOL data types

The precompiler determines the base SQLTYPE and SQLLEN of host variables based on this table. If a host variable appears with an indicator variable, the SQLTYPE is the base SQLTYPE plus one.

Table 3. COBOL declarations mapped to typical SQL data types

COBOL data type	SQLTYPE of host variable	SQLLEN of host variable	SQL data type
S9(i)V9(d) COMP-3 or S9(i)V9(d) COMP or S9(i)V9(d) PACKED-DECIMAL	484	i+d in byte 1, d in byte 2	DECIMAL(i+d,d)
S9(i)V9(d) DISPLAY SIGN LEADING SEPARATE	504	i+d in byte 1, d in byte 2	No exact equivalent use DECIMAL(i+d,d) or NUMERIC (i+d,d)
S9(i)V9(d)DISPLAY	488	i+d in byte 1, d in byte 2	NUMERIC(i+d,d)
S9(i) BINARY or S9(i) COMP-4 where i is from 1 to 4	500	2	SMALLINT
S9(i) BINARY or S9(i) COMP-4 where i is from 5 to 9	496	4	INTEGER
S9(i) BINARY or S9(i) COMP-4 where i is from 10 to 18. Not supported by OPM COBOL.	492	8	BIGINT

Table 3. COBOL declarations mapped to typical SQL data types (continued)

COBOL data type	SQLTYPE of host variable	SQLLEN of host variable	SQL data type
S9(i)V9(d) BINARY or S9(i)V9(d) COMP-4 where i+d ≤ 4	500	i+d in byte 1, d in byte 2	No exact equivalent use DECIMAL(i+d,d) or NUMERIC (i+d,d)
S9(i)V9(d) BINARY or S9(i)V9(d) COMP-4 where 4 < i+d ≤ 9	496	i+d in byte 1, d in byte 2	No exact equivalent use DECIMAL(i+d,d) or NUMERIC (i+d,d)
COMP-1 Not supported by OPM COBOL.	480	4	FLOAT(single precision)
COMP-2 Not supported by OPM COBOL.	480	8	FLOAT(double precision)
Fixed-length character data	452	m	CHAR(m)
Varying-length character data	448	m	VARCHAR(m)
Fixed-length graphic data Not supported by OPM COBOL.	468	m	GRAPHIC(m)
Varying-length graphic data Not supported by OPM COBOL.	464	m	VARGRAPHIC(m)
DATE Not supported by OPM COBOL.	384		DATE
TIME Not supported by OPM COBOL.	388		TIME
TIMESTAMP Not supported by OPM COBOL.	392	26	TIMESTAMP

The following table can be used to determine the COBOL data type that is equivalent to a given SQL data type.

Table 4. SQL data types mapped to typical COBOL declarations

SQL data type	COBOL data type	Notes
SMALLINT	S9(m) COMP-4	m is from 1 to 4
INTEGER	S9(m) COMP-4	m is from 5 to 9
BIGINT	S9(m) COMP-4 for ILE COBOL. Not supported by OPM COBOL.	m is from 10 to 18
DECIMAL(p,s)	If p<64: S9(p-s)V9(s) PACKED-DECIMAL or S9(p-s)V9(s) COMP or S9(p-s)V9(s) COMP-3. If p>63: Not supported	*p* is precision; *s* is scale. 0<=s<=p<=63. If s=0, use S9(p) or S9(p)V. If s=p, use SV9(s).
NUMERIC(p,s)	If p<19: S9(p-s)V9(s) DISPLAY If p>18: Not supported	*p* is precision; *s* is scale. 0<=s<=p<=18. If s=0, use S9(p) or S9(p)V. If s=p, use SV9(s).

Table 4. SQL data types mapped to typical COBOL declarations (continued)

SQL data type	COBOL data type	Notes
FLOAT(single precision)	COMP-1 for ILE COBOL. Not supported by OPM COBOL.	
FLOAT(double precision)	COMP-2 for ILE COBOL. Not supported by OPM COBOL.	
CHAR(n)	Fixed-length character string	32766≥n≥1
VARCHAR(n)	Varying-length character string	32740≥n≥1
CLOB	None	Use SQL TYPE IS to declare a CLOB for ILE COBOL. Not supported by OPM COBOL.
GRAPHIC(n)	Fixed-length graphic string for ILE COBOL. Not supported by OPM COBOL.	16383≥n≥1
VARGRAPHIC(n)	Varying-length graphic string for ILE COBOL. Not supported by OPM COBOL.	16370≥n≥1
DBCLOB	None Not supported by OPM COBOL.	Use SQL TYPE IS to declare a DBCLOB for ILE COBOL.
BINARY	None	Use SQL TYPE IS to declare a BINARY.
VARBINARY	None	Use SQL TYPE IS to declare a VARBINARY.
BLOB	None Not supported by OPM COBOL.	Use SQL TYPE IS to declare a BLOB.
DATE	Fixed-length character string or DATE for ILE COBOL.	If the format is *USA, *JIS, *EUR, or *ISO, allow at least 10 characters. If the format is *YMD, *DMY, or *MDY, allow at least 8 characters. If the format is *JUL, allow at least 6 characters.
TIME	Fixed-length character string or TIME for ILE COBOL.	Allow at least 6 characters; 8 to include seconds.
TIMESTAMP	Fixed-length character string or TIMESTAMP for ILE COBOL.	n must be at least 19. To include microseconds at full precision, n must be 26. If n is less than 26, truncation occurs on the microseconds part.
DATALINK	Not supported	
ROWID	None	Use SQL TYPE IS to declare a ROWID.

Notes on COBOL variable declaration and usage

Any level 77 data description entry can be followed by one or more REDEFINES entries. However, the names in these entries cannot be used in SQL statements.

Unpredictable results may occur when a structure contains levels defined below a FILLER item.

The COBOL declarations for SMALLINT, INTEGER, and BIGINT data types are expressed as a number of decimal digits. The database manager uses the full size of the integers and can place larger values in the host variable than would be allowed in the specified number of digits in the COBOL declaration. However, this can cause data truncation or size errors when COBOL statements are being run. Ensure that the size of numbers in your application is within the declared number of digits.

Using indicator variables in COBOL applications that use SQL

An *indicator variable* is a two-byte integer (PIC S9(m) USAGE BINARY, where m is from 1 to 4).

You can also specify an indicator structure (defined as an array of halfword integer variables) to support a host structure. On retrieval, an indicator variable is used to show whether its associated host variable has been assigned a null value. On assignment to a column, a negative indicator variable is used to indicate that a null value should be assigned.

Indicator variables are declared in the same way as host variables, and the declarations of the two can be mixed in any way that seems appropriate to the programmer.

Example

Given the statement:

```
EXEC SQL FETCH CLS_CURSOR INTO :CLS-CD,
                               :NUMDAY :NUMDAY-IND,
                               :BGN :BGN-IND,
                               :ENDCLS :ENDCLS-IND
END-EXEC.
```

The variables can be declared as follows:

```
EXEC SQL BEGIN DECLARE SECTION END-EXEC.
77 CLS-CD     PIC X(7).
77 NUMDAY     PIC S9(4) BINARY.
77 BGN        PIC X(8).
77 ENDCLS     PIC X(8).
77 NUMDAY-IND PIC S9(4) BINARY.
77 BGN-IND    PIC S9(4) BINARY.
77 ENDCLS-IND PIC S9(4) BINARY.
EXEC SQL END DECLARE SECTION END-EXEC.
```

Related reference

References to variables

Coding SQL statements in PL/I applications

There are some unique application and coding requirements for embedding SQL statements in a PL/I program. In this topic, requirements for host structures and host variables are defined.

Note: By using the code examples, you agree to the terms of the “Code license and disclaimer information” on page 176.

Related concepts

“Writing applications that use SQL” on page 2
You can create database applications in host languages that use DB2 UDB for iSeries SQL statements and functions.

"Error and warning messages during a compile of application programs that use SQL" on page 132
These conditions might produce an error or warning message during an attempted compile process.

Related reference

"Example programs: Using DB2 UDB for iSeries statements" on page 136
Here is a sample application that shows how to code SQL statements in each of the languages that DB2 UDB for iSeries supports.

Defining the SQL communications area in PL/I applications that use SQL

A PL/I program that contains SQL statements must include one or both of these fields.

- An SQLCODE variable declared as FIXED BINARY(31)
- An SQLSTATE variable declared as CHAR(5)

Or,

- An SQLCA (which contains an SQLCODE and SQLSTATE variable).

The SQLCODE and SQLSTATE values are set by the database manager after each SQL statement is run. An application can check the SQLCODE or SQLSTATE value to determine whether the last SQL statement was successful.

The SQLCA can be coded in a PL/I program either directly or by using the SQL INCLUDE statement. Using the SQL INCLUDE statement requests the inclusion of a standard SQLCA declaration:

```
EXEC SQL INCLUDE SQLCA ;
```

The scope of the SQLCODE, SQLSTATE, and SQLCA variables must include the scope of all SQL statements in the program.

The included PL/I source statements for the SQLCA are:

```
DCL 1 SQLCA,
      2 SQLCAID       CHAR(8),
      2 SQLCABC       FIXED(31) BINARY,
      2 SQLCODE       FIXED(31) BINARY,
      2 SQLERRM       CHAR(70) VAR,
      2 SQLERRP       CHAR(8),
      2 SQLERRD(6)    FIXED(31) BINARY,
      2 SQLWARN,
        3 SQLWARN0    CHAR(1),
        3 SQLWARN1    CHAR(1),
        3 SQLWARN2    CHAR(1),
        3 SQLWARN3    CHAR(1),
        3 SQLWARN4    CHAR(1),
        3 SQLWARN5    CHAR(1),
        3 SQLWARN6    CHAR(1),
        3 SQLWARN7    CHAR(1),
        3 SQLWARN8    CHAR(1),
        3 SQLWARN9    CHAR(1),
        3 SQLWARNA    CHAR(1),
      2 SQLSTATE      CHAR(5);
```

SQLCODE is replaced with SQLCADE when a declare for SQLCODE is found in the program and the SQLCA is provided by the precompiler. SQLSTATE is replaced with SQLSTOTE when a declare for SQLSTATE is found in the program and the SQLCA is provided by the precompiler.

Related reference

SQL communication area

Defining SQL descriptor areas in PL/I applications that use SQL

There are two types of SQL descriptor areas. One is defined with the ALLOCATE DESCRIPTOR statement. The other is defined using the SQLDA structure. In this topic, only the SQLDA form is discussed.

The following statements can use an SQLDA:

- EXECUTE...USING DESCRIPTOR *descriptor-name*
- FETCH...USING DESCRIPTOR *descriptor-name*
- OPEN...USING DESCRIPTOR *descriptor-name*
- CALL...USING DESCRIPTOR *descriptor-name*
- DESCRIBE *statement-name* INTO *descriptor-name*
- DESCRIBE INPUT *statement-name* INTO *descriptor-name*
- DESCRIBE TABLE *host-variable* INTO *descriptor-name*
- PREPARE *statement-name* INTO *descriptor-name*

Unlike the SQLCA, there can be more than one SQLDA in a program, and an SQLDA can have any valid name. An SQLDA can be coded in a PL/I program either program directly or by using the SQL INCLUDE statement. Using the SQL INCLUDE statement requests the inclusion of a standard SQLDA declaration:

```
EXEC SQL INCLUDE SQLDA ;
```

The included PL/I source statements for the SQLDA are:

```
DCL 1 SQLDA BASED(SQLDAPTR),
      2 SQLDAID       CHAR(8),
      2 SQLDABC       FIXED(31) BINARY,
      2 SQLN          FIXED(15) BINARY,
      2 SQLD          FIXED(15) BINARY,
      2 SQLVAR(99),
        3 SQLTYPE     FIXED(15) BINARY,
        3 SQLLEN      FIXED(15) BINARY,
        3 SQLRES      CHAR(12),
        3 SQLDATA     PTR,
        3 SQLIND      PTR,
        3 SQLNAME     CHAR(30) VAR;
DCL SQLDAPTR PTR;
```

Dynamic SQL is an advanced programming technique. With dynamic SQL, your program can develop and then run SQL statements while the program is running. A SELECT statement with a variable SELECT list (that is, a list of the data to be returned as part of the query) that runs dynamically requires an SQL descriptor area (SQLDA). This is because you cannot know in advance how many or what type of variables to allocate in order to receive the results of the SELECT.

Related concepts

Dynamic SQL applications

Related reference

SQL descriptor area

Embedding SQL statements in PL/I applications that use SQL

The first statement of the PL/I program must be a PROCEDURE statement. SQL statements can be coded in a PL/I program wherever executable statements can appear.

Each SQL statement in a PL/I program must begin with EXEC SQL and end with a semicolon (;). The key words EXEC SQL must appear all on one line, but the remainder of the statement can appear on the next and subsequent lines.

Example: Embedding SQL statements in PL/I applications that use SQL

You can code an UPDATE statement in a PL/I program as in this example.

```
EXEC SQL  UPDATE DEPARTMENT
            SET MGRNO = :MGR_NUM
            WHERE DEPTNO = :INT_DEPT ;
```

Comments in PL/I applications that use SQL

In addition to SQL comments (--), you can include PL/I comments (/*...*/) in embedded SQL statements wherever a blank is allowed, except between the keywords EXEC and SQL.

Continuation for SQL statements in PL/I applications that use SQL

The line continuation rules for SQL statements are the same as those for other PL/I statements, except that EXEC SQL must be specified within one line.

Constants containing DBCS data can be continued across multiple lines by placing the shift-in and shift-out characters outside of the margins. This example assumes margins of 2 and 72. This SQL statement has a valid graphic constant of G’<AABBCCDDEEFFGGHHIIJJKK>’.

```
*(..+....1....+....2....+....3....+....4....+....5....+....6....+....7.)..
 EXEC SQL SELECT * FROM GRAPHTAB          WHERE GRAPHCOL =  G'<AABBCCDD>
<EEFFGGHHIIJJKK>';
```

Including code in PL/I applications that use SQL

SQL statements or PL/I host variable declaration statements can be included by placing the following SQL statement at the point in the source code where the statements are to be embedded.

```
EXEC SQL INCLUDE member-name ;
```

No PL/I preprocessor directives are permitted within SQL statements. PL/I %INCLUDE statements cannot be used to include SQL statements or declarations of PL/I host variables that are referenced in SQL statements.

Margins in PL/I applications that use SQL

You must code SQL statements within the margins specified by the MARGINS parameter on the CRTSQLPLI command. If EXEC SQL does not start within the specified margins, the SQL precompiler will not recognize the SQL statement.

Related concepts

“CL command descriptions for host language precompilers” on page 174
The DB2 UDB for iSeries database provides commands for precompiling programs coded in these programming languages.

Names in PL/I applications that use SQL

Any valid PL/I variable name can be used for a host variable and is subject to these restrictions.

Do not use host variable names or external entry names that begin with ’SQL’, ’RDI’, or ’DSN’. These names are reserved for the database manager.

Statement labels in PL/I applications that use SQL

All executable SQL statements, like PL/I statements, can have a label prefix.

WHENEVER statement in PL/I applications that use SQL

The target for the GOTO clause in an SQL WHENEVER statement must be a label in the PL/I source code and must be within the scope of any SQL statements affected by the WHENEVER statement.

Using host variables in PL/I applications that use SQL

All host variables used in SQL statements must be explicitly declared.

The PL/I statements that are used to define the host variables should be preceded by a BEGIN DECLARE SECTION statement and followed by an END DECLARE SECTION statement. If a BEGIN DECLARE SECTION and END DECLARE SECTION are specified, all host variable declarations used in SQL statements must be between the BEGIN DECLARE SECTION and the END DECLARE SECTION statements.

All host variables within an SQL statement must be preceded by a colon (:).

The names of host variables must be unique within the program, even if the host variables are in different blocks or procedures.

An SQL statement that uses a host variable must be within the scope of the statement in which the variable was declared.

Host variables must be scalar variables. They cannot be elements of an array.

Declaring host variables in PL/I applications that use SQL

The PL/I precompiler only recognizes a subset of valid PL/I declarations as valid host variable declarations.

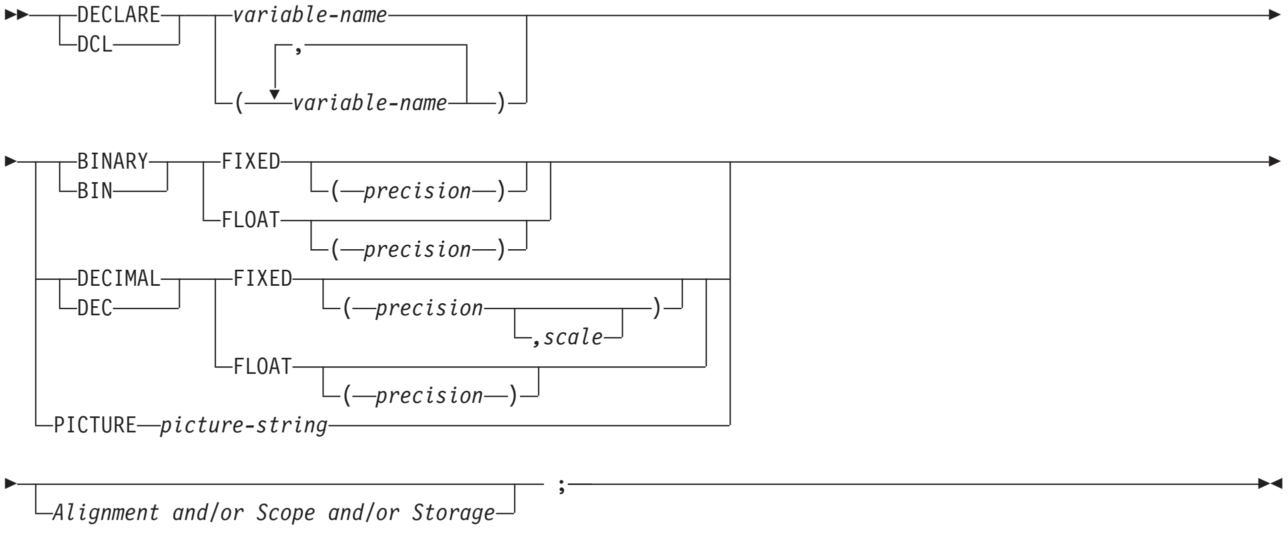
Only the names and data attributes of the variables are used by the precompilers; the alignment, scope, and storage attributes are ignored. Even though alignment, scope, and storage are ignored, there are some restrictions on their use that, if ignored, may result in problems when compiling PL/I source code that is created by the precompiler. These restrictions are:

- A declaration with the EXTERNAL scope attribute and the STATIC storage attribute must also have the INITIAL storage attribute.
- If the BASED storage attribute is coded, it must be followed by a PL/I element-locator-expression.

Numeric-host variables in PL/I applications that use SQL:

This figure shows the syntax for valid scalar numeric-host variable declarations.

Numeric



Notes:

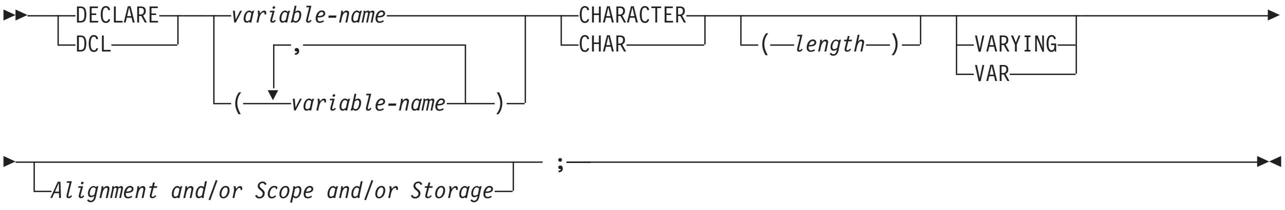
1. (BINARY, BIN, DECIMAL, or DEC) and (FIXED or FLOAT) and (precision, scale) can be specified in any order.

2. A picture-string in the form ’9...9V9...R’ indicates a numeric host variable. The R is required. The optional V indicates the implied decimal point.
3. A picture-string in the form ’S9...9V9...9’ indicates a sign leading separate host variable. The S is required. The optional V indicates the implied decimal point.

Character-host variables in PL/I applications that use SQL:

This figure shows the syntax for valid scalar character-host variables.

Character



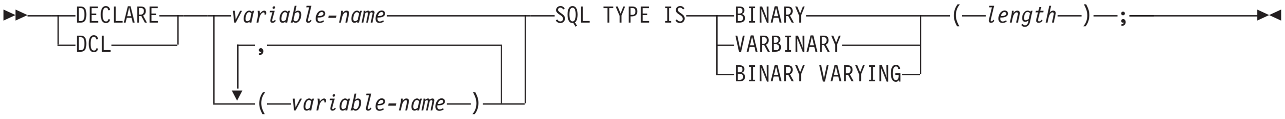
Notes:

1. *Length* must be an integer constant not greater than 32766 if VARYING or VAR is not specified.
2. If VARYING or VAR is specified, *length* must be a constant no greater than 32740.

Binary host variables in PL/I applications that use SQL:

PL/I does not have variables that correspond to the SQL binary data types. To create host variables that can be used with these data types, use the SQL TYPE IS clause. The SQL precompiler replaces this declaration with a PL/I language structure in the output source member.

BINARY and VARBINARY



Notes:

1. For BINARY host variables, the length must be in the range 1 to 32766.
2. For VARBINARY and BINARY VARYING host variables, the length must be in the range 1 to 32740.
3. SQL TYPE IS, BINARY, VARBINARY, BINARY VARYING can be in mixed case.

BINARY example

The following declaration:

```
 DCL MY_BINARY SQL TYPE IS BINARY(100);
```

Results in the generation of the following code:

```
DCL MY_BINARY CHARACTER(100);
```

VARBINARY example

The following declaration:

```
 DCL MY_VARBINARY SQL TYPE IS VARBINARY(250);
```

Results in the generation of the following code:

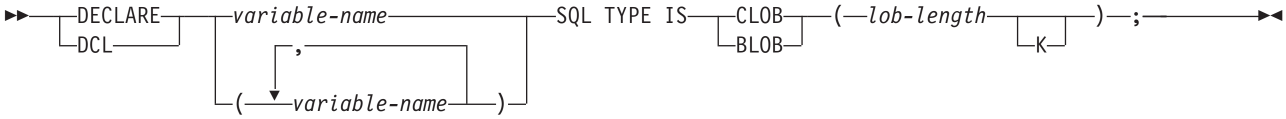
```
DCL MY_VARBINARY CHARACTER(250) VARYING;
```

LOB host variables in PL/I applications that use SQL:

PL/I does not have variables that correspond to the SQL data types for LOBs (large objects). To create host variables that can be used with these data types, use the SQL TYPE IS clause. The SQL precompiler replaces this declaration with a PL/I language structure in the output source member.

The following figure shows the syntax for valid LOB host variables.

LOB



Notes:

1. For BLOB and CLOB, 1 ≤ lob-length ≤ 32,766
2. SQL TYPE IS, BLOB, CLOB can be in mixed case.

CLOB example

The following declaration:

```
DCL MY_CLOB SQL TYPE IS CLOB(16384);
```

Results in the generation of the following structure:

```
DCL 1 MY_CLOB,
      3 MY_CLOB_LENGTH BINARY FIXED (31) UNALIGNED,
      3 MY_CLOB_DATA CHARACTER (16384);
```

BLOB example

The following declaration:

```
DCL MY_BLOB SQL TYPE IS BLOB(16384);
```

Results in the generation of the following structure:

```
DCL 1 MY_BLOB,
      3 MY_BLOB_LENGTH BINARY FIXED (31) UNALIGNED,
      3 MY_BLOB_DATA CHARACTER (16384);
```

The following figure shows the syntax for valid LOB locators.

LOB locator

```
►►──┬─DECLARE─┬──┬─variable-name──────────────┬──SQL TYPE IS──┬─CLOB_LOCATOR───┬──;────────────────────────►◄
    └─DCL─────┘  │    ┌─,─────────────┐       │               ├─DBCLOB_LOCATOR─┤
                 │    ▼               │       │               └─BLOB_LOCATOR───┘
                 └─(────variable-name─┴──)────┘
```

Note: SQL TYPE IS, BLOB_LOCATOR, CLOB_LOCATOR, DBCLOB_LOCATOR can be in mixed case.

CLOB locator example

The following declaration:

```
DCL MY_LOCATOR SQL TYPE IS CLOB_LOCATOR;
```

Results in the following generation:

```
DCL MY_LOCATOR BINARY FIXED(31) UNALIGNED;
```

BLOB and DBCLOB locators have similar syntax.

The following figure shows the syntax for valid LOB file reference variables.

LOB file reference variable

```
►►──┬─DECLARE─┬──┬─variable-name──────────────┬──SQL TYPE IS──┬─CLOB_FILE───┬──;──────────────────────────►◄
    └─DCL─────┘  │    ┌─,─────────────┐       │               ├─DBCLOB_FILE─┤
                 │    ▼               │       │               └─BLOB_FILE───┘
                 └─(────variable-name─┴──)────┘
```

Note: SQL TYPE IS, BLOB_FILE, CLOB_FILE, and DBCLOB_FILE can be in mixed case.

CLOB file reference example

The following declaration:

```
DCL MY_FILE SQL TYPE IS CLOB_FILE;
```

Results in the generation of the following structure:

```
DCL 1 MY_FILE,
      3 MY_FILE_NAME_LENGTH BINARY FIXED(31) UNALIGNED,
      3 MY_FILE_DATA_LENGTH BINARY FIXED(31) UNALIGNED,
      3 MY_FILE_FILE_OPTIONS BINARY FIXED(31) UNALIGNED,
      3 MY_FILE_NAME CHAR(255);
```

BLOB and DBCLOB file reference variables have similar syntax.

The pre-compiler will generate declarations for the following file option constants:

- SQL_FILE_READ (2)
- SQL_FILE_CREATE (8)
- SQL_FILE_OVERWRITE (16)
- SQL_FILE_APPEND (32)

Related reference

LOB file reference variables

ROWID host variables in PL/I applications that use SQL:

PL/I does not have a variable that corresponds to the SQL data type ROWID. To create host variables that can be used with this data type, use the SQL TYPE IS clause. The SQL precompiler replaces this declaration with a PL/I language structure in the output source member.

ROWID

```
►►──┬─DECLARE─┬──┬─variable-name──────────────┬──SQL TYPE IS ROWID───────────────────────────►◄
    └─DCL─────┘  │    ┌─,─────────────┐       │
                 │    ▼               │       │
                 └─(──────variable-name─┴───)─┘
```

Note: SQL TYPE IS ROWID can be in mixed case.

ROWID example

The following declaration:

```
DCL MY_ROWID SQL TYPE IS ROWID;
```

Results in the following generation:

```
DCL MY_ROWID CHARACTER(40) VARYING;
```

Using host structures in PL/I applications that use SQL

In PL/I programs, you can define a host structure, which is a named set of elementary PL/I variables. A host structure name can be a group name whose subordinate levels name elementary PL/I variables.

For example:

```
DCL 1 A,
      2 B,
        3 C1 CHAR(...),
        3 C2 CHAR(...);
```

In this example, B is the name of a host structure consisting of the elementary items C1 and C2.

You can use the structure name as shorthand notation for a list of scalars. You can qualify a host variable with a structure name (for example, STRUCTURE.FIELD). Host structures are limited to two levels. (For example, in the above host structure example, the A cannot be referred to in SQL.) A structure cannot contain an intermediate level structure. In the previous example, A could not be used as a host variable or referred to in an SQL statement. However, B is the first level structure. B can be referred to in an SQL statement. A host structure for SQL data is two levels deep and can be thought of as a named set of host variables. After the host structure is defined, you can refer to it in an SQL statement instead of listing the several host variables (that is, the names of the host variables that make up the host structure).

For example, you can retrieve all column values from selected rows of the table CORPDATA.EMPLOYEE with:

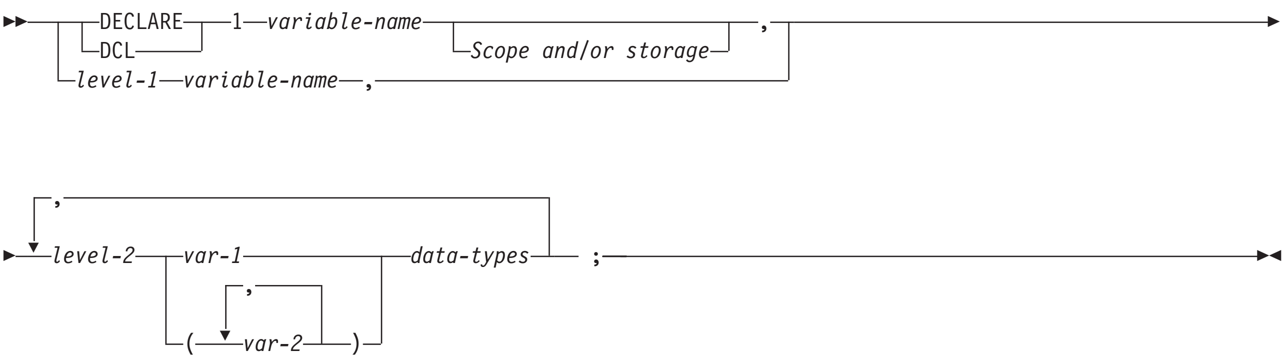
```
DCL 1 PEMPL,
      5 EMPNO    CHAR(6),
      5 FIRSTNME CHAR(12) VAR,
      5 MIDINIT  CHAR(1),
      5 LASTNAME CHAR(15) VAR,
      5 WORKDEPT CHAR(3);
 ...
EMPID = '000220';
 ...
   EXEC SQL
```

```
SELECT *
INTO :PEMPL
FROM CORPDATA.EMPLOYEE
WHERE EMPNO = :EMPID;
```

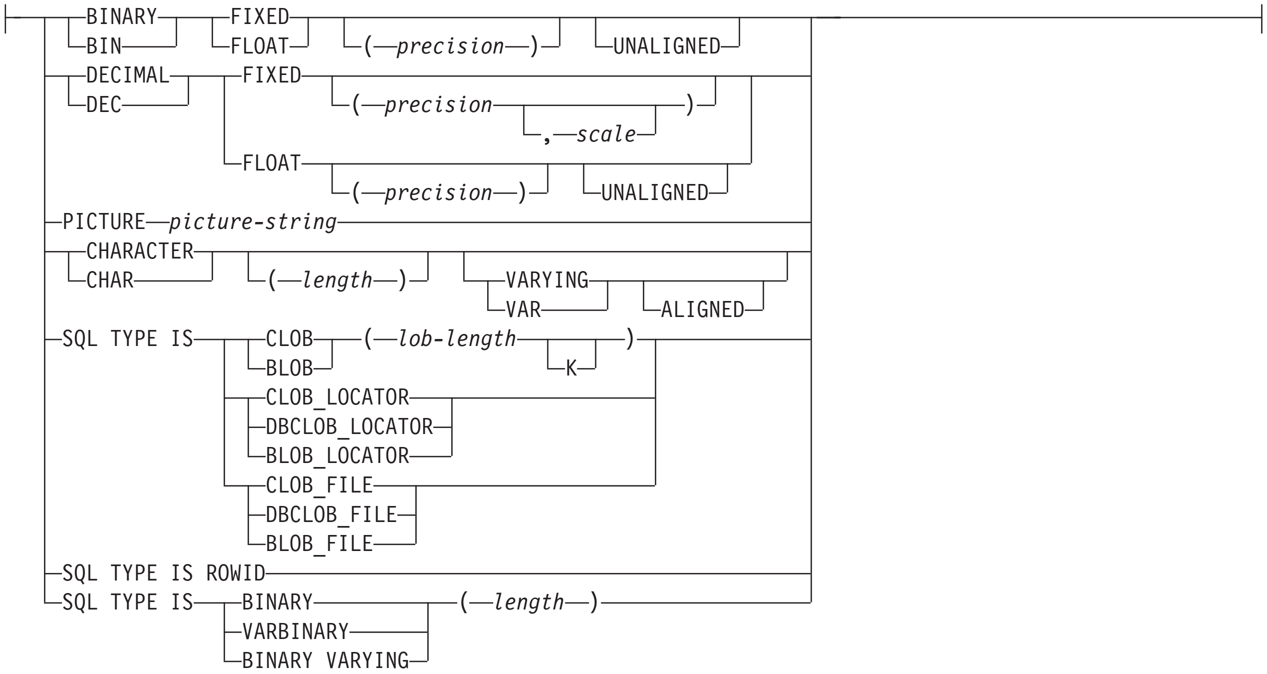
Host structures in PL/I applications that use SQL

This figure shows the syntax for valid host structure declarations.

Host structures



data-types:



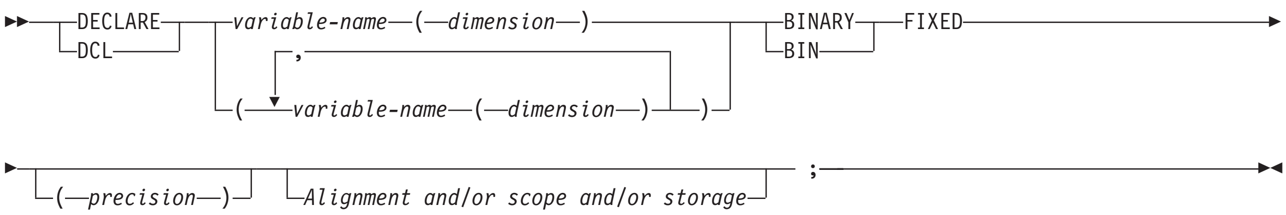
Notes:

1. Level-1 indicates that there is an intermediate level structure.
2. Level-1 must be an integer constant between 1 and 254.
3. Level-2 must be an integer constant between 2 and 255.
4. For details on declaring numeric, character, LOB, ROWID, and binary host variables, see the notes under numeric-host variables, character-host variables, LOB host variables, ROWID host variables, and binary host variables.

Host structure indicator arrays in PL/I applications that use SQL

This figure shows the syntax for valid host structure indicator array declarations.

Host structure indicator array


DECLARE
DCL
variable-name—(—dimension—)
,
(—variable-name—(—dimension—)—)
BINARY
BIN
FIXED
(—precision—)
Alignment and/or scope and/or storage
;

Note: Dimension must be an integer constant between 1 and 32766.

Using host structure arrays in PL/I applications that use SQL

In PL/I programs, you can define a host structure array.

In these examples, the following are true:

- B_ARRAY is the name of a host structure array that contains the items C1_VAR and C2_VAR.
- B_ARRAY cannot be qualified.
- B_ARRAY can only be used with the blocked forms of the FETCH and INSERT statements.
- All items in B_ARRAY must be valid host variables.
- C1_VAR and C2_VAR are not valid host variables in any SQL statement. A structure cannot contain an intermediate level structure. A_STRUCT cannot contain the dimension attribute.

```
DCL 1 A_STRUCT,
      2 B_ARRAY(10),
        3 C1_VAR CHAR(20),
        3 C2_FIXED BIN(15) UNALIGNED;
```

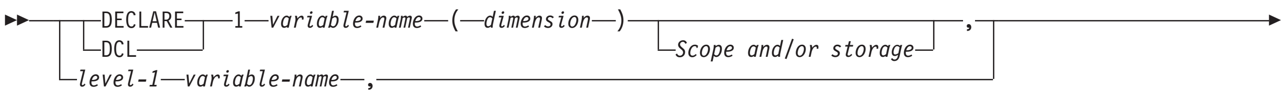
To retrieve 10 rows from the CORPDATA.DEPARTMENT table, do the following:

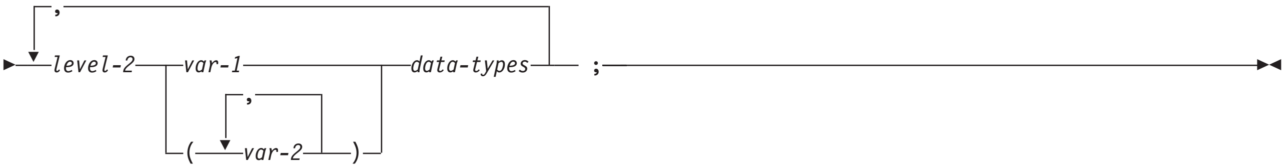
```
DCL 1 DEPT(10),
      5 DEPTPNO CHAR(3),
      5 DEPTNAME CHAR(29) VAR,
      5 MGRNO CHAR(6),
      5 ADMRDEPT CHAR (3);
DCL 1 IND_ARRAY(10),
      5 INDS(4) FIXED BIN(15);
EXEC SQL
  DECLARE C1 CURSOR FOR
     SELECT *
       FROM CORPDATA.DEPARTMENT;

EXEC SQL
  FETCH C1 FOR 10 ROWS INTO :DEPT :IND_ARRAY;
```

Host structure array in PL/I applications that use SQL

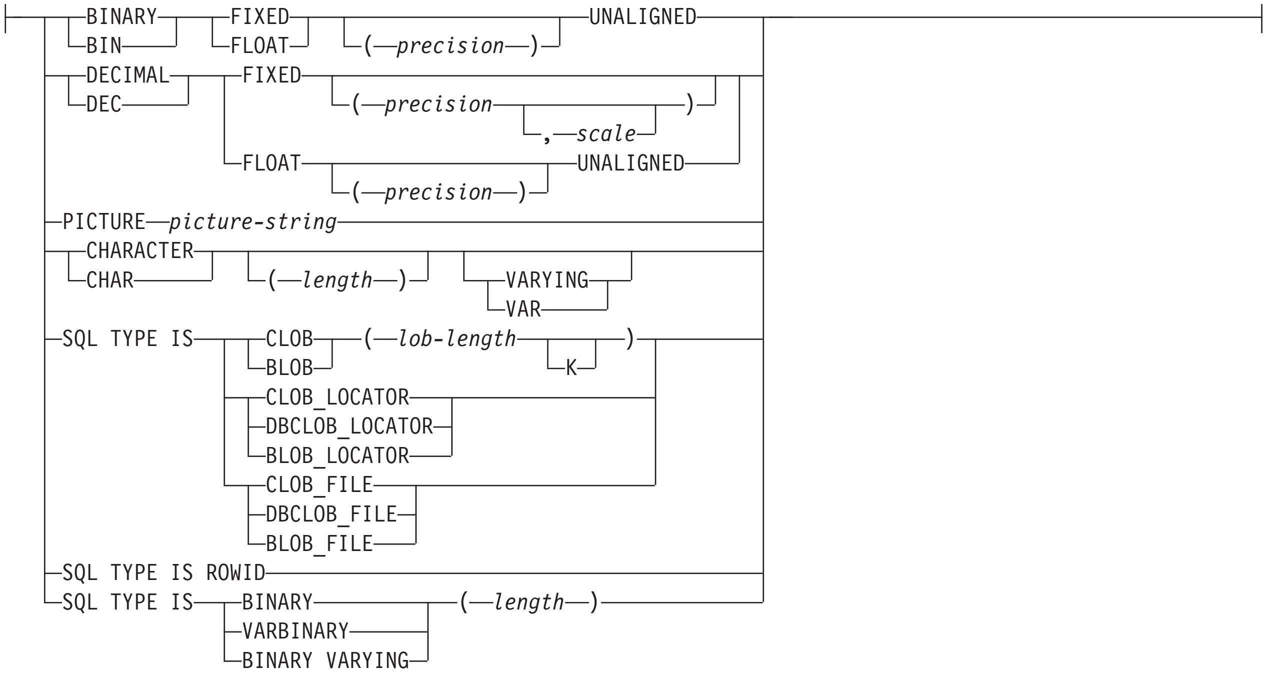
This syntax diagram shows the syntax for valid host structure array declarations.

Host structure array





data-types:

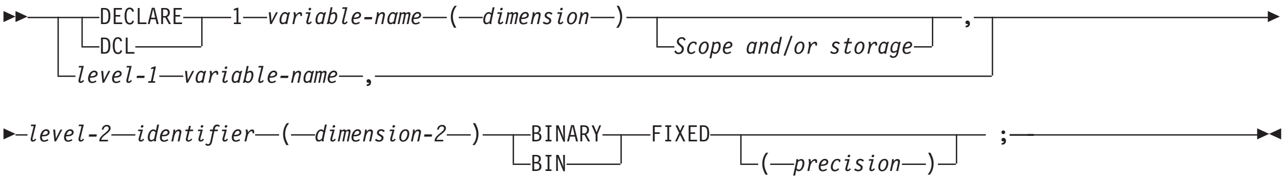


Notes:

1. Level-1 indicates that there is an intermediate level structure.
2. Level-1 must be an integer constant between 1 and 254.
3. Level-2 must be an integer constant between 2 and 255.
4. For details on declaring numeric, character, LOB, ROWID, and binary host variables, see the notes under numeric-host variables, character-host variables, LOB host variables, ROWID, and binary host variables.
5. Dimension must be an integer constant between 1 and 32767.

Host structure array indicator in PL/I applications that use SQL:

This figure shows the syntax diagram for the declaration of a valid host structure array indicator.



Notes:

1. Level-1 indicates that there is an intermediate level structure.
2. Level-1 must be an integer constant between 1 and 254.
3. Level-2 must be an integer constant between 2 and 255.
4. Dimension-1 and dimension-2 must be integer constants between 1 and 32767.

Using external file descriptions in PL/I applications that use SQL

You can use the PL/I %INCLUDE directive to include the definitions of externally described files in a source program.

When used with SQL, only a particular format of the %INCLUDE directive is recognized by the SQL precompiler. That directive format must have the following three elements or parameter values, otherwise the precompiler ignores the directive. The required elements are *file name*, *format name*, and *element type*. There are two optional elements supported by the SQL precompiler: prefix name and COMMA.

The structure is ended normally by the last data element of the record or key structure. However, if in the %INCLUDE directive the COMMA element is specified, then the structure is not ended.

To include the definition of the sample table DEPARTMENT described in DB2 UDB for iSeries sample tables in the DB2 UDB for iSeries SQL Programming topic collection, you can code:

```
DCL 1 TDEPT_STRUCTURE,
%INCLUDE DEPARTMENT(DEPARTMENT,RECORD);
```

In the above example, a host structure named TDEPT_STRUCTURE would be defined having four fields. The fields would be DEPTNO, DEPTNAME, MGRNO, and ADMRDEPT.

For device files, if INDARA is not specified and the file contains indicators, the declaration cannot be used as a host structure array. The indicator area is included in the generated structure and causes the storage to not be contiguous.

```
DCL  1 DEPT_REC(10),
     %INCLUDE DEPARTMENT(DEPARTMENT,RECORD);

EXEC SQL DECLARE C1 CURSOR FOR
    SELECT * FROM CORPDATA.DEPARTMENT;

EXEC SQL OPEN C1;

EXEC SQL FETCH C1 FOR 10 ROWS INTO :DEPT_REC;
```

Note: DATE, TIME, and TIMESTAMP columns will generate host variable definitions that are treated by SQL with the same comparison and assignment rules as a DATE, TIME, and TIMESTAMP column. For example, a date host variable can only be compared with a DATE column or a character string that is a valid representation of a date.

Although decimal and zoned fields with precision greater than 15 and binary with nonzero scale fields are mapped to character field variables in PL/I, SQL considers these fields to be numeric.

Although GRAPHIC and VARGRAPHIC are mapped to character variables in PL/I, SQL considers these to be GRAPHIC and VARGRAPHIC host variables. If the GRAPHIC or VARGRAPHIC column has a UCS-2 CCSID, the generated host variable will have the UCS-2 CCSID assigned to it. If the GRAPHIC or VARGRAPHIC column has a UTF-16 CCSID, the generated host variable will have the UTF-16 CCSID assigned to it.

Determining equivalent SQL and PL/I data types

The precompiler determines the base SQLTYPE and SQLLEN of host variables based on this table.

If a host variable appears with an indicator variable, the SQLTYPE is the base SQLTYPE plus one.

Table 5. PL/I declarations mapped to typical SQL data types

PL/I data type	SQLTYPE of host variable	SQLLEN of host variable	SQL data type
BIN FIXED(p) where p is in the range 1 to 15	500	2	SMALLINT
BIN FIXED(p) where p is in the range 16 to 31	496	4	INTEGER
DEC FIXED(p,s)	484	p in byte 1, s in byte 2	DECIMAL(p,s)
BIN FLOAT(p) p is in the range 1 to 24	480	4	FLOAT (single precision)
BIN FLOAT(p) p is in the range 25 to 53	480	8	FLOAT (double precision)
DEC FLOAT(m) m is in the range 1 to 7	480	4	FLOAT (single precision)
DEC FLOAT(m) m is in the range 8 to 16	480	8	FLOAT (double precision)
PICTURE picture string (numeric)	488	p in byte 1, s in byte 2	NUMERIC (p,s)
PICTURE picture string (sign leading separate)	504	p in byte 1, s in byte 2	No exact equivalent, use NUMERIC(p,s).
CHAR(n)	452	n	CHAR(n)
CHAR(n) VARYING	448	n	VARCHAR(n)

The following table can be used to determine the PL/I data type that is equivalent to a given SQL data type.

Table 6. SQL data types mapped to typical PL/I declarations

SQL data type	PL/I equivalent	Notes
SMALLINT	BIN FIXED(p)	p is a positive integer from 1 to 15.
INTEGER	BIN FIXED(p)	p is a positive integer from 16 to 31.
BIGINT	No exact equivalent	Use DEC FIXED(18).
DECIMAL(p,s) or NUMERIC(p,s)	DEC FIXED(p) or DEC FIXED(p,s) or PICTURE picture-string	*s* (the scale factor) and *p* (the precision) are positive integers. *p* is a positive integer from 1 to 31. *s* is a positive integer from 0 to *p*.
FLOAT (single precision)	BIN FLOAT(p) or DEC FLOAT(m)	*p* is a positive integer from 1 to 24. *m* is a positive integer from 1 to 7.
FLOAT (double precision)	BIN FLOAT(p) or DEC FLOAT(m)	*p* is a positive integer from 25 to 53. *m* is a positive integer from 8 to 16.
CHAR(n)	CHAR(n)	*n* is a positive integer from 1 to 32766.
VARCHAR(n)	CHAR(n) VARYING	*n* is a positive integer from 1 to 32740.
CLOB	None	Use SQL TYPE IS to declare a CLOB.
GRAPHIC(n)	Not supported	Not supported.
VARGRAPHIC(n)	Not supported	Not supported.
DBCLOB	Not supported	Not supported

Table 6. SQL data types mapped to typical PL/I declarations (continued)

SQL data type	PL/I equivalent	Notes
BINARY	None	Use SQL TYPE IS to declare a BINARY.
VARBINARY	None	Use SQL TYPE IS to declare a VARBINARY.
BLOB	None	Use SQL TYPE IS to declare a BLOB.
DATE	CHAR(n)	If the format is *USA, *JIS, *EUR, or *ISO, *n* must be at least 10 characters. If the format is *YMD, *DMY, or *MDY, *n* must be at least 8 characters. If the format is *JUL, *n* must be at least 6 characters.
TIME	CHAR(n)	*n* must be at least 6; to include seconds, *n* must be at least 8.
TIMESTAMP	CHAR(n)	*n* must be at least 19. To include microseconds at full precision, *n* must be 26; if *n* is less than 26, truncation occurs on the microseconds part.
DATALINK	Not supported	Not supported
ROWID	None	Use SQL TYPE IS to declare a ROWID.

Using indicator variables in PL/I applications that use SQL

An *indicator variable* is a two-byte integer (BIN FIXED(p), where p is 1 to 15).

You can also specify an indicator structure (defined as an array of halfword integer variables) to support a host structure. On retrieval, an indicator variable is used to show whether its associated host variable has been assigned a null value. On assignment to a column, a negative indicator variable is used to indicate that a null value should be assigned.

Indicator variables are declared in the same way as host variables and the declarations of the two can be mixed in any way that seems appropriate to the programmer.

Example

Given the statement:

```
EXEC SQL FETCH CLS_CURSOR INTO :CLS_CD,
                               :DAY :DAY_IND,
                               :BGN :BGN_IND,
                               :END :END_IND;
```

Variables can be declared as follows:

```
  EXEC SQL BEGIN DECLARE SECTION;
  DCL CLS_CD    CHAR(7);
  DCL DAY       BIN FIXED(15);
  DCL BGN       CHAR(8);
  DCL END       CHAR(8);
  DCL (DAY_IND, BGN_IND, END_IND)   BIN FIXED(15);
  EXEC SQL END DECLARE SECTION;
```

Related reference

References to variables

Differences in PL/I because of structure parameter passing techniques

The PL/I precompiler attempts to use the structure parameter passing technique, if possible. This structure parameter passing technique provides better performance for most PL/I programs using SQL.

The precompiler generates code where each host variable is a separate parameter when the following conditions are true:

- A PL/I %INCLUDE compiler directive is found that copies external text into the source program.
- The data length of the host variables referred to in the statement is greater than 32703. Because SQL uses 64 bytes of the structure, 32703 + 64 = 32767, the maximum length of a data structure.
- The PL/I precompiler estimates that it could possibly exceed the PL/I limit for user-defined names.
- A sign leading separate host variable is found in the host variable list for the SQL statement.

Related concepts

Application design tips for database performance

Coding SQL statements in RPG/400 applications

The RPG/400 licensed program supports both RPG II and RPG III programs.

SQL statements can only be used in RPG III programs. RPG II and AutoReport are NOT supported. All referrals to RPG in this guide apply to RPG III or ILE RPG only.

This topic describes the unique application and coding requirements for embedding SQL statements in a RPG/400 program. Requirements for host variables are defined.

Note: By using the code examples, you agree to the terms of the “Code license and disclaimer information” on page 176.

For more information about programming using RPG, see the RPG/400 User’s Guide and RPG/400 Reference manuals.

Related concepts

“Writing applications that use SQL” on page 2
You can create database applications in host languages that use DB2 UDB for iSeries SQL statements and functions.

“Error and warning messages during a compile of application programs that use SQL” on page 132
These conditions might produce an error or warning message during an attempted compile process.

Related reference

“Example programs: Using DB2 UDB for iSeries statements” on page 136
Here is a sample application that shows how to code SQL statements in each of the languages that DB2 UDB for iSeries supports.

Defining the SQL communications area in RPG/400 applications that use SQL

The SQL precompiler automatically places the SQLCA in the input specifications of the RPG/400 program prior to the first calculation specification.

INCLUDE SQLCA should not be coded in the source program. If the source program specifies INCLUDE SQLCA, the statement will be accepted, but it is redundant. The SQLCA, as defined for RPG/400:

```
ISQLCA       DS                                                      SQL
I*      SQL Communications area                                      SQL
I                                        1   8 SQLAID                SQL
I                                    B   9  120SQLABC                SQL
I                                    B  13  160SQLCOD                SQL
```

```
I                                    B  17  180SQLERL                SQL
I                                         19  88 SQLERM                SQL
I                                         89  96 SQLERP                SQL
I                                         97 120 SQLERR                SQL
I                                    B  97 1000SQLER1                SQL
I                                    B 101 1040SQLER2                SQL
I                                    B 105 1080SQLER3                SQL
I                                    B 109 1120SQLER4                SQL
I                                    B 113 1160SQLER5                SQL
I                                    B 117 1200SQLER6                SQL
I                                      121 131 SQLWRN                SQL
I                                      121 121 SQLWN0                SQL
I                                      122 122 SQLWN1                SQL
I                                      123 123 SQLWN2                SQL
I                                      124 124 SQLWN3                SQL
I                                      125 125 SQLWN4                SQL
I                                      126 126 SQLWN5                SQL
I                                      127 127 SQLWN6                SQL
I                                      128 128 SQLWN7                SQL
I                                      129 129 SQLWN8                SQL
I                                      130 130 SQLWN9                SQL
I                                      131 131 SQLWNA                SQL
I                                      132 136 SQLSTT                SQL
I*  End of SQLCA                                                     SQL
```

Note: Variable names in RPG/400 are limited to 6 characters. The standard SQLCA names have been changed to a length of 6. RPG/400 does not have a way of defining arrays in a data structure without also defining them in the extension specification. SQLERR is defined as character with SQLER1 through 6 used as the names of the elements.

Related reference

SQL communication area

Defining SQL descriptor areas in RPG/400 applications that use SQL

There are two types of SQL descriptor areas. One is defined with the ALLOCATE DESCRIPTOR statement. The other is defined using the SQLDA structure. In this topic, only the SQLDA form is discussed.

The following statements can use an SQLDA:

- EXECUTE...USING DESCRIPTOR *descriptor-name*
- FETCH...USING DESCRIPTOR *descriptor-name*
- OPEN...USING DESCRIPTOR *descriptor-name*
- CALL...USING DESCRIPTOR *descriptor-name*
- DESCRIBE *statement-name* INTO *descriptor-name*
- DESCRIBE INPUT *statement-name* INTO *descriptor-name*
- DESCRIBE TABLE *host-variable* INTO *descriptor-name*
- PREPARE *statement-name* INTO *descriptor-name*

Unlike the SQLCA, there can be more than one SQLDA in a program and an SQLDA can have any valid name.

Dynamic SQL is an advanced programming technique. With dynamic SQL, your program can develop and then run SQL statements while the program is running. A SELECT statement with a variable SELECT list (that is, a list of the data to be returned as part of the query) that runs dynamically requires an SQL descriptor area (SQLDA). This is because you cannot know in advance how many or what type of variables to allocate in order to receive the results of the SELECT.

Because the SQLDA uses pointer variables that are not supported by RPG/400, an INCLUDE SQLDA statement cannot be specified in an RPG/400 program. An SQLDA must be set up by a C, C++, COBOL, PL/I, or ILE RPG program and passed to the RPG program in order to use it.

Related concepts

Dynamic SQL applications

Related reference

SQL descriptor area

Embedding SQL statements in RPG/400 applications that use SQL

SQL statements coded in an RPG/400 program must be placed in the calculation section. This requires that a C be placed in position 6.

SQL statements can be placed in detail calculations, in total calculations, or in an RPG/400 subroutine. The SQL statements are run based on the logic of the RPG/400 statements.

The keywords EXEC SQL indicate the beginning of an SQL statement. EXEC SQL must occupy positions 8 through 16 of the source statement, preceded by a / in position 7. The SQL statement may start in position 17 and continue through position 74.

The keyword END-EXEC ends the SQL statement. END-EXEC must occupy positions 8 through 16 of the source statement, preceded by a slash (/) in position 7. Positions 17 through 74 must be blank.

Both uppercase and lowercase letters are acceptable in SQL statements.

Example: Embedding SQL statements in RPG/400 applications that use SQL

An UPDATE statement coded in an RPG/400 program might be coded as this example shows.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...*
C/EXEC SQL UPDATE DEPARTMENT
C+           SET MANAGER = :MGRNUM
C+           WHERE DEPTNO = :INTDEP
C/END-EXEC
```

Comments in RPG/400 applications that use SQL

In addition to SQL comments (--), RPG/400 comments can be included within SQL statements wherever a blank is allowed, except between the keywords EXEC and SQL.

To embed an RPG/400 comment within the SQL statement, place an asterisk (*) in position 7.

Continuation for SQL statements in RPG/400 applications that use SQL

When additional records are needed to contain the SQL statement, positions 9 through 74 can be used. Position 7 must be a + (plus sign), and position 8 must be blank.

Constants containing DBCS data can be continued across multiple lines by placing the shift-in character in position 75 of the continued line and placing the shift-out character in position 8 of the continuation line. This SQL statement has a valid graphic constant of G'<AABBCCDDEEFFGGHHIIJJKK>'.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
C/EXEC SQL SELECT * FROM GRAPHTAB           WHERE GRAPHCOL =  G'<AABB>
C+<CCDDEEFFGGHHIIJJKK>'
C/END-EXEC
```

Including code in RPG/400 applications that use SQL

SQL statements and RPG/400 calculation specifications can be included by embedding the SQL statement.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
C/EXEC SQL INCLUDE member-name
C/END-EXEC
```

The /COPY statement can be used to include SQL statements or RPG/400 specifications.

Sequence numbers in RPG/400 applications that use SQL

The sequence numbers of the source statements generated by the SQL precompiler are based on the *NOSEQSRC/*SEQSRC keywords of the OPTION parameter on the CRTSQLRPG command.

When *NOSEQSRC is specified, the sequence number from the input source member is used. For *SEQSRC, the sequence numbers start at 000001 and are incremented by 1.

Names in RPG/400 applications that use SQL

Any valid RPG variable name can be used for a host variable and is subject to these restrictions.

Do not use host variable names or external entry names that begin with 'SQ', 'SQL', 'RDI', or 'DSN'. These names are reserved for the database manager.

Statement labels in RPG/400 applications that use SQL

A TAG statement can precede any SQL statement. Code the TAG statement on the line preceding EXEC SQL.

WHENEVER statement in RPG/400 applications that use SQL

The target for the GOTO clause must be the label of the TAG statement. The scope rules for the GOTO/TAG must be observed.

Using host variables in RPG/400 applications that use SQL

All host variables used in SQL statements must be explicitly declared. LOB, ROWID, and binary host variables are not supported in RPG/400.

SQL embedded in RPG/400 does not use the SQL BEGIN DECLARE SECTION and END DECLARE SECTION statements to identify host variables. Do not put these statements in the source program.

All host variables within an SQL statement must be preceded by a colon (:).

The names of host variables must be unique within the program.

Declaring host variables in RPG/400 applications that use SQL

The SQL RPG/400 precompiler only recognizes a subset of RPG/400 declarations as valid host variable declarations.

Most variables defined in RPG/400 can be used in SQL statements. A partial listing of variables that are not supported includes the following:

- Indicator field names (*INxx)
- Tables
- UDATE
- UDAY
- UMONTH
- UYEAR
- Look-ahead fields
- Named constants

Fields used as host variables are passed to SQL, using the CALL/PARM functions of RPG/400. If a field cannot be used in the result field of the PARM, it cannot be used as a host variable.

Using host structures in RPG/400 applications that use SQL

The RPG/400 data structure name can be used as a host structure name if subfields exist in the data structure. The use of the data structure name in an SQL statement implies that it is the list of subfield names that make up the data structure.

When subfields are not present for the data structure, then the data structure name is a host variable of character type. This allows character variables larger than 256, because data structures can be up to 9999.

In the following example, BIGCHR is an RPG/400 data structure without subfields. SQL treats any referrals to BIGCHR as a character string with a length of 642.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...*
IBIGCHR      DS                            642
```

In the next example, PEMPL is the name of the host structure consisting of the subfields EMPNO, FIRSTN, MIDINT, LASTNAME, and DEPTNO. The referral to PEMPL uses the subfields. For example, the first column of EMPLOYEE is placed in *EMPNO*, the second column is placed in *FIRSTN*, and so on.

```
    *...1....+....2....+....3....+....4....+....5....+....6....+....7. ..*
    IPEMPL       DS
    I                                       01  06 EMPNO
    I                                       07  18 FIRSTN
    I                                       19  19 MIDINT
    I                                       20  34 LASTNA
    I                                       35  37 DEPTNO
...
    C                     MOVE  '000220'  EMPNO
...
    C/EXEC SQL
    C+ SELECT *  INTO :PEMPL
    C+ FROM  CORPDATA.EMPLOYEE
    C+ WHERE  EMPNO = :EMPNO
    C/END-EXEC
```

When writing an SQL statement, referrals to subfields can be qualified. Use the name of the data structure, followed by a period and the name of the subfield. For example, PEMPL.MIDINT is the same as specifying only MIDINT.

Using host structure arrays in RPG/400 applications that use SQL

A host structure array is defined as an occurrence data structure. An occurrence data structure can be used on the SQL FETCH statement when fetching multiple rows.

In these examples, the following are true:

- All items in BARRAY must be valid host variables.
- All items in BARRAY must be contiguous. The first FROM position must be 1 and there cannot be overlaps in the TO and FROM positions.
- For all statements other than the multiple-row FETCH and blocked INSERT, if an occurrence data structure is used, the current occurrence is used. For the multiple-row FETCH and blocked INSERT, the occurrence is set to 1.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7. ..*
IBARRAY      DS                         10
I                                       01 20 C1VAR
I                                    B  21 220C2VAR
```

The following example uses a host structure array called DEPT and a multiple-row FETCH statement to retrieve 10 rows from the DEPARTMENT table.

```
    *...1....+....2....+....3....+....4....+....5....+....6....+....7...*
    E                             INDS         4  4 0
    IDEPT        DS                         10
    I                                       01  03 DEPTNO
    I                                       04  32 DEPTNM
    I                                       33  38 MGRNO
    I                                       39  41 ADMRD
    IINDARR      DS                         10
    I                                     B   1   80INDS
...
    C/EXEC SQL
    C+ DECLARE C1 CURSOR FOR
    C+    SELECT *
    C+       FROM  CORPDATA.DEPARTMENT
    C/END-EXEC
    C/EXEC SQL
    C+ OPEN C1
    C/END-EXEC
    C/EXEC SQL
    C+   FETCH C1 FOR 10 ROWS INTO :DEPT:INDARR
    C/END-EXEC
```

Using external file descriptions in RPG/400 applications that use SQL

The SQL precompiler processes the RPG/400 source in much the same manner as the ILE RPG compiler. This means that the precompiler processes the /COPY statement for definitions of host variables.

Field definitions for externally described files are obtained and renamed, if different names are specified. The external definition form of the data structure can be used to obtain a copy of the column names to be used as host variables.

In the following example, the sample table DEPARTMENT is used as a file in an RPG/400 program. The SQL precompiler retrieves the field (column) definitions for DEPARTMENT for use as host variables.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....*
FTDEPT   IP  E                    DISK
F            TDEPT                          KRENAMEDEPTREC
IDEPTREC
I              DEPTNAME                     DEPTN
I              ADMRDEPT                     ADMRD
```

Note: Code an F-spec for a file in your RPG program only if you use RPG/400 statements to do I/O operations to the file. If you use only SQL statements to do I/O operations to the file, you can include the external definition by using an external data structure.

In the following example, the sample table is specified as an external data structure. The SQL precompiler retrieves the field (column) definitions as subfields of the data structure. Subfield names can be used as host variable names, and the data structure name TDEPT can be used as a host structure name. The field names must be changed because they are greater than six characters.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....*
ITDEPT     E DSDEPARTMENT
I              DEPTNAME                     DEPTN
I              ADMRDEPT                     ADMRD
```

Note: DATE, TIME, and TIMESTAMP columns will generate host variable definitions that are treated by SQL with the same comparison and assignment rules as a DATE, TIME, and TIMESTAMP column. For example, a date host variable can only be compared against a DATE column or a character string that is a valid representation of a date.

Although varying-length columns generate fixed-length character-host variable definitions, to SQL they are varying-length character variables.

Although GRAPHIC and VARGRAPHIC columns are mapped to character variables in RPG/400, SQL considers these GRAPHIC and VARGRAPHIC variables. If the GRAPHIC or VARGRAPHIC column has a UCS-2 CCSID, the generated host variable will have the UCS-2 CCSID assigned to it. If the GRAPHIC or VARGRAPHIC column has a UTF-16 CCSID, the generated host variable will have the UTF-16 CCSID assigned to it.

External file description considerations for host structure arrays in RPG/400 applications that use SQL

Field definitions for externally described files, including renaming of fields, are recognized by the SQL precompiler.

The external definition form of the data structure can be used to obtain a copy of the column names to be used as host variables.

In the following example, the DEPARTMENT table is included in the RPG/400 program and is used to declare a host structure array. A multiple-row FETCH statement is then used to retrieve 10 rows into the host structure array.

```
*...1....+....2....+....3....+....4....+....5....+....6....*
ITDEPT     E DSDEPARTMENT               10
I              DEPARTMENT                      DEPTN
I              ADMRDEPT                        ADMRD

...

C/EXEC SQL
C+   DECLARE C1 CURSOR FOR
C+      SELECT *
C+        FROM CORPDATA.DEPARTMENT
C/END-EXEC

...

C/EXEC SQL
C+ FETCH C1 FOR 10 ROWS INTO :TDEPT
C/END-EXEC
```

Determining equivalent SQL and RPG/400 data types

The precompiler determines the base SQLTYPE and SQLLEN of host variables based on the table. If a host variable appears with an indicator variable, the SQLTYPE is the base SQLTYPE plus one.

Table 7. RPG/400 declarations mapped to typical SQL data types

RPG/400 data type	Col 43	Col 52	Other RPG/400 coding	SQLTYPE of host variable	SQLLEN of host variable	SQL data type
Data Structure subfield	blank	blank	Length = n where n ≤ 256	452	n	CHAR(n)
Data structure (without subfields)	n/a	n/a	Length = n where n ≤ 9999	452	n	CHAR(n)
Input field	blank	blank	Length = n where n ≤ 256	452	n	CHAR(n)
Calculation result field	n/a	blank	Length = n where n ≤ 256	452	n	CHAR(n)
Data Structure subfield	B	0	Length = 2	500	2	SMALLINT
Data Structure subfield	B	0	Length = 4	496	4	INTEGER

Table 7. RPG/400 declarations mapped to typical SQL data types (continued)

RPG/400 data type	Col 43	Col 52	Other RPG/400 coding	SQLTYPE of host variable	SQLLEN of host variable	SQL data type
Data Structure subfield	B	1-4	Length = 2	500	2	DECIMAL(4,s) where s=column 52
Data Structure subfield	B	1-9	Length = 4	496	4	DECIMAL(9,s) where s=column 52
Data Structure subfield	P	0 to 9	Length = n where n is 1 to 16	484	p in byte 1, s in byte 2	DECIMAL(p,s) where p = n*2-1 and s = column 52
Input field	P	0 to 9	Length = n where n is 1 to 16	484	p in byte 1, s in byte 2	DECIMAL(p,s) where p = n*2-1 and s = column 52
Input field	blank	0 to 9	Length = n where n is 1 to 30	484	p in byte 1, s in byte 2	DECIMAL(p,s) where p = n and s = column 52
Input field	B	0 to 4 if n = 2; 0 to 9 if n = 4	Length = 2 or 4	484	p in byte 1, s in byte 2	DECIMAL(p,s) where p=4 if n=2 or 9 if n=4 and s = column 52
Calculation result field	n/a	0 to 9	Length = n where n is 1 to 30	484	p in byte 1, s in byte 2	DECIMAL(p,s) where p = n and s = column 52
Data Structure subfield	blank	0 to 9	Length = n where n is 1 to 30	488	p in byte 1, s in byte 2	NUMERIC(p,s) where p = n and s = column 52

Use the information in the following table to determine the RPG/400 data type that is equivalent to a given SQL data type.

Table 8. SQL data types mapped to typical RPG/400 declarations

SQL data type	RPG/400 data type	Notes
SMALLINT	Subfield of a data structure. B in position 43, length must be 2 and 0 in position 52 of the subfield specification.	
INTEGER	Subfield of a data structure. B in position 43, length must be 4 and 0 in position 52 of the subfield specification.	
BIGINT	No exact equivalent	Use P in position 43 and 0 in position 52 of the subfield specification.

Table 8. SQL data types mapped to typical RPG/400 declarations (continued)

SQL data type	RPG/400 data type	Notes
DECIMAL	Subfield of a data structure. P in position 43 and 0 through 9 in position 52 of the subfield specification. OR Defined as numeric and not a subfield of a data structure.	Maximum length of 16 (precision 30) and maximum scale of 9.
NUMERIC	Subfield of the data structure. Blank in position 43 and 0 through 9 in position 52 of the subfield	Maximum length of 30 (precision 30) and maximum scale of 9.
FLOAT (single precision)	No exact equivalent	Use one of the alternative numeric data types described above.
FLOAT (double precision)	No exact equivalent	Use one of the alternative numeric data types described above.
CHAR(n)	Subfield of a data structure or input field. Blank in positions 43 and 52 of the specification. OR Calculation result field defined without decimal places.	n can be from 1 to 256.
CHAR(n)	Data structure name with no subfields in the data structure.	*n* can be from 1 to 9999.
VARCHAR(n)	No exact equivalent	Use a character host variable large enough to contain the largest expected VARCHAR value.
CLOB	Not supported	Not supported
GRAPHIC(n)	Not supported	Not supported
VARGRAPHIC(n)	Not supported	Not supported
DBCLOB	Not supported	Not supported
BINARY	Not supported	Not supported
VARBINARY	Not supported	Not supported
BLOB	Not supported	Not supported
DATE	Subfield of a data structure. Blank in position 52 of the subfield specification. OR Field defined without decimal places.	If the format is *USA, *JIS, *EUR, or *ISO, the length must be at least 10. If the format is *YMD, *DMY, or *MDY, the length must be at least 8. If the format is *JUL, the length must be at least 6.
TIME	Subfield of a data structure. Blank in position 52 of the subfield specification. OR Field defined without decimal places.	Length must be at least 6; to include seconds, length must be at least 8.

Table 8. SQL data types mapped to typical RPG/400 declarations (continued)

SQL data type	RPG/400 data type	Notes
TIMESTAMP	Subfield of a data structure. Blank in position 52 of the subfield specification. OR Field defined without decimal places.	Length must be at least 19. To include microseconds at full precision, length must be 26. If length is less than 26, truncation occurs on the microseconds part.
DATALINK	Not supported	Not supported
ROWID	Not supported	Not supported

Assignment rules in RPG/400 applications that use SQL

RPG/400 associates precision and scale with all numeric types.

RPG/400 defines numeric operations, assuming the data is in packed format. This means that operations involving binary variables include an implicit conversion to packed format before the operation is performed (and back to binary, if necessary). Data is aligned to the implied decimal point when SQL operations are performed.

Using indicator variables in RPG/400 applications that use SQL

An indicator variable is a two-byte integer.

See the entry for the SMALLINT SQL data type in Table 7 on page 87.

An indicator structure can be defined by declaring the variable as an array with an element length of 4,0 and declaring the array name as a subfield of a data structure with B in position 43. On retrieval, an indicator variable is used to show whether its associated host variable has been assigned a null value. On assignment to a column, a negative indicator variable is used to indicate that a null value should be assigned.

Indicator variables are declared in the same way as host variables and the declarations of the two can be mixed in any way that seems appropriate to the programmer.

Related reference

References to variables

Example: Using indicator variables in RPG/400 applications that use SQL

This example shows declaring indicator variables in RPG.

Given the statement:

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...*
C/EXEC SQL FETCH CLS_CURSOR INTO :CLSCD,
C+                           :DAY :DAYIND,
C+                           :BGN :BGNIND,
C+                           :END :ENDIND
C/END-EXEC
```

variables can be declared as follows:

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...*
I            DS
I                                        1   7 CLSCD
I                                    B   8   90DAY
I                                    B  10  110DAYIND
```

```
I                                           12  19 BGN
I                                         B 20  210BGNIND
I                                           22  29 END
I                                         B 30  310ENDIND
```

Differences in RPG/400 because of structure parameter passing techniques

The SQL RPG/400 precompiler attempts to use the structure parameter passing technique, if possible.

The precompiler generates code where each host variable is a separate parameter when the following conditions are true:

- The data length of the host variables, referred to in the statement, is greater than 9935. Because SQL uses 64 bytes of the structure, 9935 + 64 = 9999, the maximum length of a data structure.
- An indicator is specified on the statement where the length of the indexed indicator name plus the required index value is greater than six characters. The precompiler must generate an assignment statement for the indicator with the indicator name in the result field that is limited to six characters (″INDIC,1″ requires seven characters).
- The length of a host variable is greater than 256. This can happen when a data structure without subfields is used as a host variable, and its length exceeds 256. Subfields cannot be defined with a length greater than 256.

Related concepts

Application design tips for database performance

Correctly ending a called RPG/400 program that uses SQL

SQL run time builds and maintains data areas (internal SQLDAs) for each SQL statement that contains host variables.

These internal SQLDAs are built the first time the statement is run and then reused on subsequent executions of the statement to increase performance. The internal SQLDAs can be reused as long as there is at least one SQL program active. The SQL precompiler allocates static storage used by SQL run time to manage the internal SQLDAs properly.

If an RPG/400 program containing SQL is called from another program that also contains SQL, the RPG/400 program should not set the Last Record (LR) indicator on. Setting the LR indicator on causes the static storage to be re-initialized the next time the RPG/400 program is run. Re-initializing the static storage causes the internal SQLDAs to be rebuilt, thus causing a performance degradation.

An RPG/400 program containing SQL statements that is called by a program that also contains SQL statements, should be ended one of two ways:

- By the RETRN statement
- By setting the RT indicator on.

This allows the internal SQLDAs to be used again and reduces the total run time.

Coding SQL statements in ILE RPG applications

You need to be aware of the unique application and coding requirements for embedding SQL statements in an ILE RPG program. In this topic, the coding requirements for host variables are defined.

Note: By using the code examples, you agree to the terms of the “Code license and disclaimer information” on page 176.

For more information about programming using ILE RPG, see the ILE RPG Programmer’s Guide topic and the ILE RPG Language Reference topic.

Related concepts

“Writing applications that use SQL” on page 2
You can create database applications in host languages that use DB2 UDB for iSeries SQL statements and functions.

“Error and warning messages during a compile of application programs that use SQL” on page 132
These conditions might produce an error or warning message during an attempted compile process.

Related reference

“Example: SQL statements in ILE RPG programs” on page 163
This example program is written in the ILE RPG programming language.

Defining the SQL communication area in ILE RPG applications that use SQL

The SQL precompiler automatically places the SQL communication area (SQLCA) in the definition specifications of the ILE RPG program prior to the first calculation specification, unless a SET OPTION SQLCA = *NO statement is found.

INCLUDE SQLCA should not be coded in the source program. If the source program specifies INCLUDE SQLCA, the statement will be accepted, but it is redundant. The SQLCA source statements for ILE RPG are:

```
D*      SQL Communication area
D SQLCA           DS
D  SQLCAID                        8A   INZ(X'0000000000000000')
D  SQLAID                         8A   OVERLAY(SQLCAID)
D  SQLCABC                       10I 0
D  SQLABC                         9B 0 OVERLAY(SQLCABC)
D  SQLCODE                       10I 0
D  SQLCOD                         9B 0 OVERLAY(SQLCODE)
D  SQLERRML                       5I 0
D  SQLERL                         4B 0 OVERLAY(SQLERRML)
D  SQLERRMC                      70A
D  SQLERM                        70A   OVERLAY(SQLERRMC)
D  SQLERRP                        8A
D  SQLERP                         8A   OVERLAY(SQLERRP)
D  SQLERR                        24A
D   SQLER1                        9B 0 OVERLAY(SQLERR:*NEXT)
D   SQLER2                        9B 0 OVERLAY(SQLERR:*NEXT)
D   SQLER3                        9B 0 OVERLAY(SQLERR:*NEXT)
D   SQLER4                        9B 0 OVERLAY(SQLERR:*NEXT)
D   SQLER5                        9B 0 OVERLAY(SQLERR:*NEXT)
D   SQLER6                        9B 0 OVERLAY(SQLERR:*NEXT)
D   SQLERRD                      10I 0 DIM(6)  OVERLAY(SQLERR)
D  SQLWRN                        11A
D   SQLWN0                        1A   OVERLAY(SQLWRN:*NEXT)
D   SQLWN1                        1A   OVERLAY(SQLWRN:*NEXT)
D   SQLWN2                        1A   OVERLAY(SQLWRN:*NEXT)
D   SQLWN3                        1A   OVERLAY(SQLWRN:*NEXT)
D   SQLWN4                        1A   OVERLAY(SQLWRN:*NEXT)
D   SQLWN5                        1A   OVERLAY(SQLWRN:*NEXT)
D   SQLWN6                        1A   OVERLAY(SQLWRN:*NEXT)
D   SQLWN7                        1A   OVERLAY(SQLWRN:*NEXT)
D   SQLWN8                        1A   OVERLAY(SQLWRN:*NEXT)
D   SQLWN9                        1A   OVERLAY(SQLWRN:*NEXT)
D   SQLWNA                        1A   OVERLAY(SQLWRN:*NEXT)
```

```
 D  SQLWARN                      1A   DIM(11) OVERLAY(SQLWRN)
 D  SQLSTATE                     5A
 D  SQLSTT                       5A   OVERLAY(SQLSTATE)
 D*  End of SQLCA
```

If a SET OPTION SQLCA = *NO statement is found, the SQL precompiler automatically places SQLCODE and SQLSTATE variables in the definition specification. They are defined as follows when the SQLCA is not included:

```
D SQLCODE          S           10I 0
D SQLSTATE         S            5A
```

Related reference

SQL communication area

Defining SQL descriptor areas in ILE RPG applications that use SQL

There are two types of SQL descriptor areas (SQLDAs). One is defined with the ALLOCATE DESCRIPTOR statement. The other is defined using the SQLDA structure. In this topic, only the SQLDA form is discussed.

The following statements can use an SQLDA:

- EXECUTE...USING DESCRIPTOR *descriptor-name*
- FETCH...USING DESCRIPTOR *descriptor-name*
- OPEN...USING DESCRIPTOR *descriptor-name*
- CALL...USING DESCRIPTOR *descriptor-name*
- DESCRIBE *statement-name* INTO *descriptor-name*
- DESCRIBE INPUT *statement-name* INTO *descriptor-name*
- DESCRIBE TABLE *host-variable* INTO *descriptor-name*
- PREPARE *statement-name* INTO *descriptor-name*

Unlike the SQLCA, there can be more than one SQLDA in a program and an SQLDA can have any valid name.

Dynamic SQL is a programming technique.With dynamic SQL, your program can develop and then run SQL statements while the program is running. A SELECT statement with a variable SELECT list (that is, a list of columns to be returned as part of the query) that runs dynamically requires an SQL descriptor area (SQLDA). This is because you cannot know in advance how many or what type of variables to allocate in order to receive the results of the SELECT.

You can specify an INCLUDE SQLDA statement in an ILE RPG program; however, it is not allowed in free format. The format of the statement is:

```
C/EXEC SQL INCLUDE SQLDA
C/END-EXEC
```

The INCLUDE SQLDA generates the following data structure.

```
D*      SQL Descriptor area
D SQLDA           DS
D  SQLDAID               1      8A
D  SQLDABC               9     12B 0
D  SQLN                 13     14B 0
D  SQLD                 15     16B 0
D  SQL_VAR                     80A   DIM(SQL_NUM)
D                       17     18B 0
D                       19     20B 0
D                       21     32A
D                       33     48*
```

```
D                        49     64*
D                        65     66B 0
D                        67     96A
D*
D SQLVAR          DS
D  SQLTYPE                1      2B 0
D  SQLLEN                 3      4B 0
D  SQLRES                 5     16A
D  SQLDATA               17     32*
D  SQLIND                33     48*
D  SQLNAMELEN            49     50B 0
D  SQLNAME               51     80A
D*  End of SQLDA
```

The user is responsible for the definition of SQL_NUM. SQL_NUM must be defined as a numeric constant with the dimension required for SQL_VAR.

The INCLUDE SQLDA generates two data structures. The second data structure is used to setup and reference the part of the SQLDA that contains the field descriptions.

To set the field descriptions of the SQLDA the program sets up the field description in the subfields of SQLVAR and then assigns SQLVAR to SQL_VAR(n), where n is the number of the field in the SQLDA. This is repeated until all the field descriptions are set.

When the SQLDA field descriptions are to be referenced the user assigns SQLVAR(n) to SQL_VAR where n is the number of the field description to be processed.

Related concepts

Dynamic SQL applications

Related reference

SQL descriptor area

Embedding SQL statements in ILE RPG applications that use SQL

SQL statements coded in an ILE RPG program must be placed in the calculation section. This requires that a C be placed in position 6.

SQL statements can be placed in detail calculations, in total calculations, or in RPG subroutines. The SQL statements are run based on the logic of the RPG statements.

Both uppercase and lowercase letters are acceptable in SQL statements.

Fixed-form RPG

The keywords EXEC SQL indicate the beginning of an SQL statement. EXEC SQL must occupy positions 8 through 16 of the source statement, preceded by a / in position 7. The SQL statement may start in position 17 and continue through position 80.

The keyword END-EXEC ends the SQL statement. END-EXEC must occupy positions 8 through 16 of the source statement, preceded by a slash (/) in position 7. Positions 17 through 80 must be blank.

An UPDATE statement coded in an ILE RPG program might be coded as follows:

```
C/EXEC SQL UPDATE DEPARTMENT
C+           SET MANAGER = :MGRNUM
C+           WHERE DEPTNO = :INTDEP
C/END-EXEC
```

Free-form RPG

Each SQL statement must begin with EXEC SQL and end with a semicolon (;). The EXEC SQL keywords must be on one line. The remaining part of the SQL statement can be on more than one line.

Example: An UPDATE statement coded in free form might be coded in the following way:

```
EXEC SQL UPDATE DEPARTMENT
  SET MGRNO = :MGR_NUM
  WHERE DEPTNO = :INT_DEP;
```

Comments in ILE RPG applications that use SQL

In addition to SQL comments (--), ILE RPG comments can be included within SQL statements wherever SQL allows a blank character.

Fixed-form RPG

To embed an ILE RPG comment within the SQL statement, place an asterisk (*) in position 7.

Free-form RPG

Bracketed comments (/*...*/) are allowed within embedded SQL statements between positions 8 through 80 and whenever a blank is allowed, except between the keywords EXEC and SQL. Comments can span any number of lines. Single-line comments (//) can also be used.

Continuation for SQL statements in ILE RPG applications that use SQL

SQL statements can be continued across many records in ILE RPG.

Fixed-form RPG

When additional records are needed to contain the SQL statement, positions 9 through 80 can be used. Position 7 must be a plus sign (+), and position 8 must be blank. Position 80 of the continued line is concatenated with position 9 of the continuation line.

Constants containing DBCS data can be continued across multiple lines by placing the shift-in character in position 81 of the continued line and placing the shift-out character in position 8 of the continuation line.

In this example, the SQL statement has a valid graphic constant of G'<AABBCCDDEEFFGGHHIIJJKK>'.

```
C/EXEC SQL    SELECT  *  FROM  GRAPHTAB  WHERE  GRAPHCOL  =   G'<AABBCCDDEE>
C+<FFGGHHIIJJKK>'
C/END-EXEC
```

Free-form RPG

SQL statements can be contained on one or more lines. To continue an SQL statement across multiple lines, the SQL statement can be split wherever a blank is allowed. The plus sign (+) can be used to indicate a continuation of a string constant. The literal continues with the first nonblank character on the next line.

Including code in ILE RPG applications that use SQL

To include SQL statements and RPG specifications in ILE RPG applications, use the SQL INCLUDE statement.

```
C/EXEC SQL INCLUDE member-name
C/END-EXEC
```

RPG directives are handled by the SQL precompiler according to the value of the RPG preprocessor options parameter (RPGPPOPT).

Related reference

"Using directives in ILE RPG applications that use SQL"
RPG directives are handled by the SQL precompiler according to the value of the RPG preprocessor options parameter (RPGPPOPT). If the RPG preprocessor is used, the SQL precompile will run using the expanded preprocessed source.

Using directives in ILE RPG applications that use SQL

RPG directives are handled by the SQL precompiler according to the value of the RPG preprocessor options parameter (RPGPPOPT). If the RPG preprocessor is used, the SQL precompile will run using the expanded preprocessed source.

- When the value is *NONE, the RPG preprocessor is not called to preprocess the RPG source. The only directive handled by the SQL precompiler is /COPY. Nested /COPY statements will not be handled. All other directives will be ignored until the RPG compiler is called. This means that all RPG and SQL statements within conditional logic blocks will be processed unconditionally by the SQL precompiler.
- When the value is *LVL1, the RPG preprocessor will be called to preprocess the RPG source. All /COPY statements are expanded, even nested /COPY statements, and the conditional compilation directives will be handled.
- When the value is *LVL2, the RPG preprocessor will be called to preprocess the RPG source. All /COPY and /INCLUDE statements are expanded and the conditional compilation directives will be handled.
- When *LVL1 or *LVL2 is used, there is a possibility that the expanded source generated by the RPG preprocessor will become very large and reach a resource limit due to the expansion of the /COPY and /INCLUDE statements. If this happens you must either break up your source into smaller pieces, or not use the RPG preprocessor.

Related reference

"Including code in ILE RPG applications that use SQL" on page 95
To include SQL statements and RPG specifications in ILE RPG applications, use the SQL INCLUDE statement.

Sequence numbers in ILE RPG applications that use SQL

The sequence numbers of the source statements generated by the SQL precompiler are based on the *NOSEQSRC/*SEQSRC keywords of the OPTION parameter on the CRTSQLRPGI command.

When *NOSEQSRC is specified, the sequence number from the input source member is used. For *SEQSRC, the sequence numbers start at 000001 and are incremented by 1.

Names in ILE RPG applications that use SQL

Any valid ILE RPG variable name can be used for a host variable with these restrictions.

- Do not use host variable names or external entry names that begin with the characters SQ, SQL, RDI, or DSN. These names are reserved for the database manager.
- The length of host variable names is limited to 64.
- The names of host variables must be unique within the program. The one exception is that if a stand-alone field, parameter, or both, are defined exactly the same as another stand-alone field, parameter, or both, the duplicated name is accepted.
- If a host variable is a duplicated name and does not belong to the exceptional category mentioned in the previous item, but does have the same type, the precompiler issues SQL0314 as a severity 11 error instead of its normal severity of 35. If you want to ignore these severity 11 errors, change the GENLVL parameter value on the CRTSQLRPGI command to be 11 or higher.

Statement labels in ILE RPG applications that use SQL

A TAG statement can precede any SQL statement. Code the TAG statement on the line preceding EXEC SQL.

WHENEVER statement in ILE RPG applications that use SQL

The target for the GOTO clause must be the label of the TAG statement. The scope rules for the GOTO/TAG must be observed.

Using host variables in ILE RPG applications that use SQL

All host variables used in SQL statements must be explicitly declared.

SQL embedded in ILE RPG does not use the SQL BEGIN DECLARE SECTION and END DECLARE SECTION statements to identify host variables. Do not put these statements in the source program.

All host variables within an SQL statement must be preceded by a colon (:).

The names of host variables must be unique within the program, even if the host variables are in different procedures. However, if a data structure has the QUALIFIED keyword, then the subfields of that data structure can have the same name as a subfield in a different data structure or as a stand-alone variable. The subfield of a data structure with the QUALIFIED keyword must be referenced using the data structure name to qualify the subfield name.

An SQL statement that uses a host variable must be within the scope of the statement in which the variable was declared.

If an error stating that a host variable is not defined or not usable is issued, look at the cross-reference in the precompiler listing to see how the precompiler defined the variable. To generate a cross-reference in the listing, run the precompile command with *XREF specified on the OPTIONS parameter.

Declaring host variables in ILE RPG applications that use SQL

The SQL ILE RPG precompiler only recognizes a subset of valid ILE RPG declarations as valid host variable declarations.

Most variables defined in ILE RPG can be used in SQL statements. A partial listing of variables that are not supported includes the following:

- Unsigned integers
- Pointer
- Tables
- UDATE
- UDAY
- UMONTH
- UYEAR
- Look-ahead fields
- Named constants
- Multiple dimension arrays
- Definitions requiring the resolution of %SIZE or %ELEM
- Definitions requiring the resolution of constants unless the constant is used in OCCURS or DIM.

Fields used as host variables are passed to SQL using the CALL/PARM functions of ILE RPG. If a field cannot be used in the result field of the PARM, it cannot be used as a host variable.

Date and time host variables are always assigned to corresponding date and time subfields in the structures generated by the SQL precompiler. The generated date and time subfields are declared using the format and separator specified by the DATFMT, DATSEP, TIMFMT, and TIMSEP parameters on the CRTSQLRPGI command or with the SET OPTION statement. Conversion from the user declared host variable format to the precompile specified format occurs on assignment to and from the SQL generated structure. If the DATFMT parameter value is a system format (*MDY, *YMD, *DMY, or *JUL), then all input and output host variables must contain date values within the range 1940-2039. If any date value is outside of this range, then the DATFMT on the precompile must be specified as one of the IBM SQL formats of *ISO, *USA, *EUR, or *JIS.

Graphic host variables will use the RPG CCSID value if one is specified. An SQL DECLARE VARIABLE statement cannot be used to change the CCSID of a host variable whose CCSID has been defined in RPG, or a host variable that is defined as UCS-2 or UTF-16.

The precompiler will generate an RPG logical (indicator) variable as a character of length 1. This type can be used wherever SQL allows a character host variable. It cannot be used as an SQL indicator variable. It is up to the user to make sure that only values of 1 or 0 are assigned to it.

The precompiler supports EXTNAME(filename : fmtname), but does not support EXTNAME(filename : fmtname : fieldtype), where fieldtype is *ALL, *INPUT, *OUTPUT, or *KEY.

The precompiler supports LIKEREC(intrecname), but does not support the optional second parameter.

If there is an unnamed subfield, the precompiler will not allow the data structure containing the subfield to be used in the blocked fetch and blocked insert statements. For all other SQL statements where the data structure containing the subfield is used, only the subfields that are named will be used.

If the PREFIX keyword has a prefix that contains a period, the precompiler will not recognize the externally described file.

Declaring binary host variables in ILE RPG applications that use SQL:

ILE RPG does not have variables that correspond to the SQL binary data types.

To create host variables that can be used with these data types, use the SQLTYPE keyword. The SQL precompiler replaces this declaration with an ILE RPG language declaration in the output source member. Binary declarations can be either standalone or within a data structure.

BINARY example

The following declaration:

```
D MYBINARY   S   SQLTYPE(BINARY:50)
```

results in the generation of the following code:

```
D MYBINARY   S    50A
```

VARBINARY example

The following declaration:

```
D MYVARBINARY   S    SQLTYPE(VARBINARY:100)
```

results in the generation of the following code:

```
D MYVARBINARY   S    100A VARYING
```

Notes:

1. For BINARY host variables, the length must be in the range 1 to 32766.
2. For VARBINARY host variables, the length must be in the range 1 to 32740.
3. BINARY and VARBINARY host variables are allowed to be declared in host structures.
4. SQLTYPE, BINARY, and VARBINARY can be in mixed case.
5. SQLTYPE must be between positions 44 to 80.
6. When a BINARY or VARBINARY is declared as a standalone host variable, position 24 must contain the character **S** and position 25 must be blank.
7. The standalone field indicator **S** in position 24 should be omitted when a BINARY or VARBINARY host variable is declared in a host structure.

Declaring LOB host variables in ILE RPG applications that use SQL:

ILE RPG does not have variables that correspond to the SQL data types for LOBs (large objects).

To create host variables that can be used with these data types, use the SQLTYPE keyword. The SQL precompiler replaces this declaration with an ILE RPG language structure in the output source member. LOB declarations can be either standalone or within a data structure.

LOB host variables in ILE RPG applications that use SQL:

Here are some examples of LOB host variables (CLOB, DBCLOB, BLOB) in ILE RPG applications.

CLOB example

The following declaration:

```
D MYCLOB          S          SQLTYPE(CLOB:1000)
```

results in the generation of the following structure:

```
D MYCLOB          DS
D MYCLOB_LEN                 10U
D MYCLOB_DATA              1000A
```

DBCLOB example

The following declaration:

```
D MYDBCLOB          S          SQLTYPE(DBCLOB:400)
```

results in the generation of the following structure:

```
D MYDBCLOB          DS
D MYDBCLOB_LEN                 10U
D MYDBCLOB_DATA               400G
```

BLOB example

The following declaration:

```
D MYBLOB          S          SQLTYPE(BLOB:500)
```

results in the generation of the following structure:

```
D MYBLOB          DS
D MYBLOB_LEN                 10U
D MYBLOB_DATA               500A
```

Notes:

1. For BLOB and CLOB, 1 ≤ lob-length ≤ 65 531

2. For DBCLOB, 1≤ lob-length ≤ 16 383
3. LOB host variables are allowed to be declared in host structures.
4. LOB host variables are not allowed in host structure arrays. LOB locators should be used instead.
5. LOB host variables declared in structure arrays cannot be used as standalone host variables.
6. SQLTYPE, BLOB, CLOB, DBCLOB can be in mixed case.
7. SQLTYPE must be between positions 44 to 80.
8. When a LOB is declared as a stand-alone host variable, position 24 must contain the character ’S’ and position 25 must be blank.
9. The stand-alone field indicator S in position 24 should be omitted when a LOB is declared in a host structure.
10. LOB host variables cannot be initialized.

LOB locators in ILE RPG applications that use SQL:

BLOB, CLOB, and DBCLOB locators have similar syntax. Here is an example of a BLOB locator.

Example: BLOB locator

The following declaration:

```
D MYBLOB          S          SQLTYPE(BLOB_LOCATOR)
```

results in the following generation:

```
D MYBLOB          S          10U
```

Notes:

1. LOB locators are allowed to be declared in host structures.
2. SQLTYPE, BLOB_LOCATOR, CLOB_LOCATOR, DBCLOB_LOCATOR can be in mixed case.
3. SQLTYPE must be between positions 44 to 80.
4. When a LOB locator is declared as a standalone host variable, position 24 must contain the character ’S’ and position 25 must be blank.
5. The standalone field indicator **S** in position 24 should be omitted when a LOB locator is declared in a host structure.
6. LOB locators cannot be initialized.

LOB file reference variables in ILE RPG applications that use SQL:

Here is an example of a CLOB file reference variable in ILE RPG. BLOB and DBCLOB file reference variables have similar syntax.

CLOB file reference example

The following declaration:

```
D MY_FILE         S          SQLTYPE(CLOB_FILE)
```

results in the generation of the following structure:

```
D MY_FILE         DS
D MY_FILE_NL                     10U
D MY_FILE_DL                     10U
D MY_FILE_FO                     10U
D MY_FILE_NAME                  255A
```

BLOB and DBCLOB locators have similar syntax.

Notes:

1. LOB file reference variables are allowed to be declared in host structures.
2. SQLTYPE, BLOB_FILE, CLOB_FILE, DBCLOB_FILE can be in mixed case.
3. SQLTYPE must be between positions 44 to 80.
4. When a LOB file reference is declared as a standalone host variable, position 24 must contain the character 'S' and position 25 must be blank.
5. The standalone field indicator 'S' in position 24 should be omitted when a LOB file reference variable is declared in a host structure.
6. LOB file reference variables cannot be initialized.

The pre-compiler will generate declarations for the following file option constants. You can use these constants to set the xxx_FO variable when you use file reference host variables.

- SQFRD (2)
- SQFCRT (8)
- SQFOVR (16)
- SQFAPP (32)

Related reference

LOB file reference variables

Declaring ROWID variables in ILE RPG applications that use SQL:

ILE RPG does not have a variable that corresponds to the SQL data type ROWID.

To create host variables that can be used with this data type, use the SQLTYPE keyword. The SQL precompiler replaces this declaration with an ILE RPG language declaration in the output source member. ROWID declarations can be either standalone or within a data structure.

ROWID example

The following declaration:

```
D MY_ROWID        S         SQLTYPE(ROWID)
```

results in the following generation:

```
D MYROWID         S         40A  VARYING
```

Notes:

1. SQLTYPE, ROWID can be in mixed case.
2. ROWID host variables are allowed to be declared in host structures.
3. SQLTYPE must be between positions 44 and 80.
4. When a ROWID is declared as a standalone host variable, position 24 must contain the character 'S' and position 25 must be blank.
5. The standalone field indicator 'S' in position 24 should be omitted when a ROWID is declared in a host structure.
6. ROWID host variables cannot be initialized.

Using host structures in ILE RPG applications that use SQL

The ILE RPG data structure name can be used as a host structure name if subfields exist in the data structure. The use of the data structure name in an SQL statement implies the specification of the list of subfield names that make up the data structure.

When a data structure contains one or more unnamed subfields, the data structure name cannot be used as a host structure in an SQL statement. The named subfields can be used as host variables.

In the following example, BIGCHR is an ILE data structure without subfields. SQL treats any references to BIGCHR as a character string with a length of 642.

```
DBIGCHR           DS           642
```

In the next example, PEMPL is the name of the host structure consisting of the subfields EMPNO, FIRSTN, MIDINT, LASTNAME, and DEPTNO. A reference to PEMPL uses the subfields. For example, the first column of CORPDATA.EMPLOYEE is placed in *EMPNO*, the second column is placed in *FIRSTN*, and so on.

```
DPEMPL            DS
D EMPNO                  01     06A
D FIRSTN                 07     18A
D MIDINT                 19     19A
D LASTNA                 20     34A
D DEPTNO                 35     37A

...
C                   MOVE      '000220'      EMPNO

...
C/EXEC SQL
C+ SELECT *  INTO :PEMPL
C+ FROM  CORPDATA.EMPLOYEE
C+ WHERE  EMPNO = :EMPNO
C/END-EXEC
```

When writing an SQL statement, references to subfields that are not in a QUALIFIED data structure can be qualified. Use the name of the data structure, followed by a period and the name of the subfield. For example, PEMPL.MIDINT is the same as specifying only MIDINT. If the data structure has the QUALIFIED keyword, then the subfield must be referenced using the data structure name to qualify the subfield name.

In this example, there are two data structures, one QUALIFIED and one not QUALIFIED, that contain the same subfield names:

```
Dfststruct       DS
D sub1                          4B 0
D sub2                          9B 0
D sub3                         20I 0
D sub4                          9B 0

Dsecstruct       DS                      QUALIFIED
D sub1                          4A
D sub2                         12A
D sub3                         20I 0
D myvar                         5A
D sub5                         20A

D myvar          S             10I 0
```

Referencing *secstruct.sub1* as a host variable will be a character variable with a length of 4.

sub2 as a host variable will have an SQL data type of small integer. It picks up its attributes from the data structure that is not QUALIFIED.

A host variable reference to *myvar* will use the standalone declaration to pick up the data type of integer. If you use *secstruct.myvar*, the character variable in the QUALIFIED structure will be used.

You cannot refer to *sub5* without qualifying it with *secstruct* because it is in a QUALIFIED data structure.

The precompiler will recognize a host structure defined using the LIKEDS keyword. However, the SQL syntax for a host variable only allows using a single level of qualification in an SQL statement. This means that if a data structure DS has a subfield S1 which is defined like a data structure with a subfield S2, an SQL statement cannot refer to S2 using the fully qualified host variable name of DS.S1.S2. If you use S1.S2 as the host variable reference, the precompiler will recognize it as DS.S1.S2. The following additional restrictions apply:

- The top level structure, DS, cannot be an array.
- S1.S2 must be unique. That is, there must be no other valid names in the program ending with S1.S2, such as a structure S1 with a subfield S1.S2, or a structure DS3 with a subfield DS3.S0.S1.S2.

Example

```
D CustomerInfo     DS                         QUALIFIED
D    Name                               20A
D    Address                            50A

D ProductInfo      DS                         QUALIFIED
D    Number                              5A
D    Description                        20A
D    Cost                                9P 2

D SalesTransaction...
D                  DS                         QUALIFIED
D    Buyer                                    LIKEDS(CustomerInfo)
D    Seller                                   LIKEDS(CustomerInfo)
D    NumProducts                        10I 0
D    Product                                  LIKEDS(ProductInfo)
D                                             DIM(10)

C/EXEC SQL
C+ SELECT * INTO :CustomerInfo.Name, :Buyer.Name FROM MYTABLE
C/END-EXEC
```

CustomerInfo.Name will be recognized as a reference to the QUALIFIED structure's variable. *Buyer.Name* will be defined as *SalesTransaction.Buyer.Name*.

You cannot use *SalesTransaction.Buyer.Name* in an SQL statement because only one level of qualification is allowed in SQL syntax. You cannot use *Product.Cost* in an SQL statement because COST is in a dimensioned array.

If there is a *SalesTransaction2* defined like *SalesTransaction*, then the subfields that are structures cannot be used in SQL statements. Because only one level of qualification is supported by SQL, a reference to *Buyer.Name* is ambiguous.

Using host structure arrays in ILE RPG applications that use SQL

A host structure array is defined as an occurrence data structure or a data structure with the keyword DIM coded. Both types of data structures can be used on the SQL FETCH or INSERT statement when processing multiple rows.

The following list of items must be considered when using a data structure with multiple row blocking support.

- All subfields must be valid host variables.
- All subfields must be contiguous. The first FROM position must be 1 and there cannot be overlaps in the TO and FROM positions.
- If the date and time format and separator of date and time subfields within the host structure are not the same as the DATFMT, DATSEP, TIMFMT, and TIMSEP parameters on the CRTSQLRPGI command (or in the SET OPTION statement), then the host structure array is not usable.

For all statements, other than the blocked FETCH and blocked INSERT, if an occurrence data structure is used, the current occurrence is used. For the blocked FETCH and blocked INSERT, the occurrence is set to 1.

The following example uses a host structure array called DEPARTMENT and a blocked FETCH statement to retrieve 10 rows from the DEPARTMENT table.

```
DDEPARTMENT               DS                 OCCURS(10)
D DEPTNO                 01     03A
D DEPTNM                 04     32A
D MGRNO                  33     38A
D ADMRD                  39     41A

DIND_ARRAY        DS                 OCCURS(10)
D INDS                          4B 0 DIM(4)
...
C/EXEC SQL
C+ DECLARE C1 CURSOR FOR
C+    SELECT *
C+       FROM  CORPDATA.DEPARTMENT
C/END-EXEC
...

C/EXEC SQL
C+   FETCH C1 FOR 10 ROWS
C+     INTO :DEPARTMENT:IND_ARRAY
C/END-EXEC
```

Blocked FETCH and blocked INSERT are the only SQL statements that allow a data structure with the DIM keyword. A host variable reference with a subscript like *MyStructure(index).Mysubfield* is not supported by SQL.

Example

```
Dfststruct        DS                     DIM(10)   QUALIFIED
D sub1                          4B 0
D sub2                          9B 0
D sub3                         20I 0
D sub4                          9B 0

C/EXEC SQL
C+   FETCH C1 FOR 10 ROWS INTO :fststruct
C/END-EXEC
```

Using external file descriptions in ILE RPG applications that use SQL

Field definitions for externally described files, including renaming of fields, are recognized by the SQL precompiler. The external definition form of the data structure can be used to obtain a copy of the column names to be used as host variables.

How date and time field definition are retrieved and processed by the SQL precompiler depends on whether *NOCVTDT or *CVTDT is specified on the OPTION parameter of the CRTSQLRPGI command. If *NOCVTDT is specified, then date and time field definitions are retrieved including the format and separator. If *CVTDT is specified, then the format and separator is ignored when date and time field definitions are retrieved, and the precompiler assumes that the variable declarations are date/time host variables in character format. *CVTDT is a compatibility option for the ILE RPG precompiler.

If the GRAPHIC or VARGRAPHIC column has a UCS-2 CCSID, the generated host variable will have the UCS-2 CCSID assigned to it. If the GRAPHIC or VARGRAPHIC column has a UTF-16 CCSID, the generated host variable will have the UTF-16 CCSID assigned to it.

In the following example, the sample table DEPARTMENT is used as a file in an ILE RPG program. The SQL precompiler retrieves the field (column) definitions for DEPARTMENT for use as host variables.

```
FDEPARTMENTIP   E             DISK    RENAME(ORIGREC:DEPTREC)
```

Note: Code an F-spec for a file in your ILE RPG program only if you use ILE RPG statements to do I/O operations to the file. If you use only SQL statements to do I/O operations to the file, you can include the external definition of the file (table) by using an external data structure.

In the following example, the sample table is specified as an external data structure. The SQL precompiler retrieves the field (column) definitions as subfields of the data structure. Subfield names can be used as host variable names, and the data structure name TDEPT can be used as a host structure name. The example shows that the field names can be renamed if required by the program.

```
DTDEPT          E DS                EXTNAME(DEPARTMENT)
D DEPTN         E                   EXTFLD(DEPTNAME)
D ADMRD         E                   EXTFLD(ADMRDEPT)
```

External file description considerations for host structure arrays in ILE RPG applications that use SQL

For device files, if INDARA was not specified and the file contains indicators, the declaration is not used as a host structure array. The indicator area is included in the structure that is generated and would cause the storage to be separated.

If OPTION(*NOCVTDT) is specified and the date and time format and separator of date and time field definitions within the file are not the same as the DATFMT, DATSEP, TIMFMT, and TIMSEP parameters on the CRTSQLRPGI command, then the host structure array is not usable.

In the following example, the DEPARTMENT table is included in the ILE RPG program and used to declare a host structure array. A blocked FETCH statement is then used to retrieve 10 rows into the host structure array.

```
DDEPARTMENT     E DS                OCCURS(10)


C/EXEC SQL
C+   DECLARE C1 CURSOR FOR
C+      SELECT *
C+        FROM CORPDATA.DEPARTMENT
C/END-EXEC

...

C/EXEC SQL
C+        FETCH C1 FOR 10 ROWS
C+           INTO :DEPARTMENT
C/END-EXEC
```

Determining equivalent SQL and ILE RPG data types

The precompiler determines the base SQLTYPE and SQLLEN of host variables according to this table. If a host variable appears with an indicator variable, the SQLTYPE is the base SQLTYPE plus one.

Table 9. ILE RPG declarations mapped to typical SQL data types

RPG data type	RPG coding	SQLTYPE of host variable	SQLLEN of host variable	SQL data type
Data structure (without subfields)	Length = n where n ≤ 32766.	452	n	CHAR(n)

Table 9. ILE RPG declarations mapped to typical SQL data types (continued)

RPG data type	RPG coding	SQLTYPE of host variable	SQLLEN of host variable	SQL data type
Zoned data	• Defined on Definition specification as subfield with data type S or blank. • Defined on Definition specification with data type S. • Defined on Input specification with data type S or blank.	488	p in byte 1, s in byte 2	NUMERIC(p, s) where p is the number of digits and s is the number of decimal places
Packed data	• Defined on Definition specification with decimal positions (pos 69-70) not blank. • Defined on Definition specification subfield with data type P. • Defined on Definition specification with data type P or blank. • Defined on Input specification with data type P.	484	p in byte 1, s in byte 2	DECIMAL(p, s) where p is the number of digits and s is the number of decimal places
2-byte binary with zero decimal positions	• Defined on Definition specification as subfield with from and to positions and data type B and byte length 2. • Defined on Definition specification with data type B and digits from 1 to 4. • Defined on Input specification with data type B and byte length 2	500	2	SMALLINT
4-byte binary with zero decimal positions	• Defined on Definition specification as subfield with from and to positions and data type B and byte length 4. • Defined on Definition specification with data type B and digits from 5 to 9. • Defined on Input specification with data type B and byte length 4.	496	4	INTEGER

Table 9. ILE RPG declarations mapped to typical SQL data types (continued)

RPG data type	RPG coding	SQLTYPE of host variable	SQLLEN of host variable	SQL data type
2-byte integer	• Defined on Definition specification as subfield with from and to positions and data type I and byte length 2. • Defined on Definition specification with data type I and digits 5. • Defined on Input specification with data type I and byte length 2.	500	2	SMALLINT
4-byte integer	• Defined on Definition specification as subfield with from and to positions and data type I and byte length 4. • Defined on Definition specification with data type I and digits 10. • Defined on Input specification with data type I and byte length 4.	496	4	INTEGER
8-byte integer	• Defined on Definition specification as subfield with from and to positions and data type I and byte length 8. • Defined on Definition specification with data type I and digits 20. • Defined on Input specification with data type I and byte length 8.	492	8	BIGINT
short float	Data type = F, length = 4.	480	4	FLOAT (single precision)
long float	Data type = F, length = 8.	480	8	FLOAT (double precision)
Character	Data type = A or blank, decimal positions blank, length between 1 and 32766.	452	n	CHAR (n) where n is the length
Character varying length greater than 254	Data type = A or blank, decimal positions blank, VARYING keyword on Definition specification or format *VAR on Input specification.	448	n	VARCHAR (n) where n is the length
Character varying length between 1 and 254	Data type = A or blank, decimal positions blank, VARYING keyword on Definition specification or format *VAR on Input specification.	456	n	VARCHAR (n) where n is the length

Table 9. ILE RPG declarations mapped to typical SQL data types (continued)

RPG data type	RPG coding	SQLTYPE of host variable	SQLLEN of host variable	SQL data type
graphic	• Defined on Definition specification as subfield with from and to positions and data type G and byte-length b. • Defined on Definition specification with data type G and length n. • Defined on Input specification with data type G and byte-length b	468	m	GRAPHIC(m) where m = n or m = b/2
varying graphic	• Defined on Definition specification as subfield with from and to positions and data type G and byte-length b and VARYING keyword. • Defined on Definition specification with data type G and length n and VARYING keyword. • Defined on Input specification with data type G and byte-length b and format *VAR.	464	m	VARGRAPHIC(m) where m = n or m = (b-2)/2
UCS-2	• Defined on Definition specification as subfield with from and to positions and data type C and byte-length b. • Defined on Definition specification with data type C and length n. • Defined on Input specification with data type C and byte-length b.	468	m	GRAPHIC(m) with CCSID 13488 where m = n or m = b/2
varying UCS-2	• Defined on Definition specification as subfield with from and to positions and data type C and byte-length b and VARYING keyword. • Defined on Definition specification with data type C and length n and VARYING keyword. • Defined on Input specification with data type C and byte-length b and format *VAR.	464	m	VARGRAPHIC(m) with CCSID 13488 where m = n or m = b/2

Table 9. ILE RPG declarations mapped to typical SQL data types (continued)

RPG data type	RPG coding	SQLTYPE of host variable	SQLLEN of host variable	SQL data type
Date	• Defined on Definition specification with data type D, format f and separator s from DATFMT keyword. • Defined on Input specification with data type D and format in pos 31-34, separator in pos 35.	384	n	DATE DATFMT(f) DATSEP(s)[1]
Time	• Defined on Definition specification with data type T, format f and separator s from TIMFMT keyword. • Defined on Input specification with data type T and format in pos 31-34, separator in pos 35.	388	n	TIME TIMFMT(f) TIMSEP(s)[1]
Timestamp	Data type Z.	392	n	TIMESTAMP
[1]SQL creates the date/time subfield using the DATE/TIME format specified on the CRTSQLRPGI command. The conversion to the host variable DATE/TIME format occurs when the mapping is done between the host variables and the SQL-generated subfields.				

The following table can be used to determine the RPG data type that is equivalent to a given SQL data type.

Table 10. SQL data types mapped to typical RPG declarations

SQL data type	RPG data type	Notes
SMALLINT	Definition specification. I in position 40, length must be 5 and 0 in position 42. OR Definition specification. B in position 40, length must be ≤ 4 and 0 in position 42.	
INTEGER	Definition specification. I in position 40, length must be 10 and 0 in position 42. OR Definition specification. B in position 40, length must be ≤ 9 and ≥ 5 and 0 in position 42.	
BIGINT	Definition specification. I in position 40, length must be 20 and 0 in position 42.	

Table 10. SQL data types mapped to typical RPG declarations (continued)

SQL data type	RPG data type	Notes
DECIMAL	Definition specification. P in position 40 or blank in position 40 for a non-subfield, 0 through 30 in position 41,42. OR Defined as numeric on non-definition specification.	Maximum length of 16 (precision 30) and maximum scale of 30.
NUMERIC	Definition specification. S in position 40 or blank in position 40 for a subfield, 0 through 30 in position 41,42.	Maximum length of 30 (precision 30) and maximum scale of 30.
FLOAT (single precision)	Definition specification. F in position 40, length must be 4.	
FLOAT (double precision)	Definition specification. F in position 40, length must be 8.	
CHAR(n)	Definition specification. A or blank in positions 40 and blanks in position 41,42. OR Input field defined without decimal places. OR Calculation result field defined without decimal places.	n can be from 1 to 32766.
CHAR(n)	Data structure name with no subfields in the data structure.	n can be from 1 to 32766.
VARCHAR(n)	Definition specification. A or blank in position 40 and VARYING in positions 44-80.	n can be from 1 to 32740.
CLOB	Not supported	Use SQLTYPE keyword to declare a CLOB.
GRAPHIC(n)	Definition specification. G in position 40. OR Input field defined with G in position 36.	n can be 1 to 16383.
VARGRAPHIC(n)	Definition specification. G in position 40 and VARYING in positions 44-80.	n can be from 1 to 16370.
DBCLOB	Not supported	Use SQLTYPE keyword to declare a DBCLOB.
BINARY	Not supported	Use SQLTYPE keyword to declare a BINARY.
VARBINARY	Not supported	Use SQLTYPE keyword to declare a VARBINARY.

Table 10. SQL data types mapped to typical RPG declarations (continued)

SQL data type	RPG data type	Notes
BLOB	Not supported	Use SQLTYPE keyword to declare a BLOB.
DATE	A character field OR Definition specification with a D in position 40. OR Input field defined with D in position 36.	If the format is *USA, *JIS, *EUR, or *ISO, the length must be at least 10. If the format is *YMD, *DMY, or *MDY, the length must be at least 8. If the format is *JUL, the length must be at least 6.
TIME	A character field OR Definition specification with a T in position 40. OR Input field defined with T in position 36.	Length must be at least 6; to include seconds, length must be at least 8.
TIMESTAMP	A character field OR Definition specification with a Z in position 40. OR Input field defined with Z in position 36.	Length must be at least 19; to include microseconds, length must be at least 26. If length is less than 26, truncation occurs on the microsecond part.
DATALINK	Not supported	
ROWID	Not supported	Use SQLTYPE keyword to declare a ROWID.

Notes on ILE RPG variable declaration and usage

ILE RPG associates precision and scale with all numeric types.

ILE RPG defines numeric operations, assuming the data is in packed format. This means that operations involving binary variables include an implicit conversion to packed format before the operation is performed (and back to binary, if necessary). Data is aligned to the implied decimal point when SQL operations are performed.

Using indicator variables in ILE RPG applications that use SQL

An indicator variable is a binary field with length less than 5 (2 bytes).

An indicator array can be defined by declaring the variable element length of 4,0 and specifying the DIM on the definition specification.

On retrieval, an indicator variable is used to show if its associated host variable has been assigned a null value. On assignment to a column, a negative indicator variable is used to indicate that a null value should be assigned.

Indicator variables are declared in the same way as host variables and the declarations of the two can be mixed in any way that seems appropriate to the programmer.

Related reference

References to variables

Example: Using indicator variables in ILE RPG applications that use SQL

Here is an example of declaring indicator variables in ILE RPG.

Given the statement:

```
C/EXEC SQL FETCH CLS_CURSOR INTO :CLSCD,
C+                            :DAY :DAYIND,
C+                            :BGN :BGNIND,
C+                            :END :ENDIND
C/END-EXEC
```

variables can be declared as follows:

```
D CLSCD           S               7
D DAY             S               2B 0
D DAYIND          S               2B 0
D BGN             S               8A
D BGNIND          S               2B 0
D END             S               8
D ENDIND          S               2B 0
```

Example: SQLDA for a multiple row-area fetch in ILE RPG applications that use SQL

Here is an example of the SQL descriptor area (SQLDA) for a multiple row-area fetch in ILE RPG.

```
C/EXEC SQL INCLUDE SQLDA
C/END-EXEC
DDEPARTMENT       DS                  OCCURS(10)
D DEPTNO                 01      03A
D DEPTNM                 04      32A
D MGRNO                  33      38A
D ADMRD                  39      41A
...

DIND_ARRAY        DS                  OCCURS(10)
D INDS                            4B 0 DIM(4)
...
C* setup number of sqlda entries and length of the sqlda
C                   eval      sqld = 4
C                   eval      sqln = 4
C                   eval      sqldabc = 336
C*
C* setup the first entry in the sqlda
C*
C                   eval      sqltype = 453
C                   eval      sqllen  = 3
C                   eval      sql_var(1) = sqlvar
C*
C* setup the second entry in the sqlda
C*
C                   eval      sqltype = 453
C                   eval      sqllen  = 29
C                   eval      sql_var(2) = sqlvar
...
C*
```

```
C* setup the forth entry in the sqlda
C*
C                   eval      sqltype = 453
C                   eval      sqllen  = 3
C                   eval      sql_var(4) = sqlvar

...
C/EXEC SQL
C+ DECLARE C1 FOR
C+    SELECT *
C+       FROM  CORPDATA.DEPARTMENT
C/END-EXEC
...

C/EXEC SQL
C+   FETCH C1 FOR 10 ROWS
C+     USING DESCRIPTOR :SQLDA
C+     INTO :DEPARTMENT:IND_ARRAY
C/END-EXEC
```

Example: Dynamic SQL in an ILE RPG application that uses SQL

Here is an example of using dynamic SQL in ILE RPG.

```
D**************************************************
D* Declare program variables.                     *
D* STMT initialized to the                        *
D* listed SQL statement.                          *
D**************************************************
D EMPNUM          S              6A
D NAME            S             15A
D STMT            S            500A   INZ('SELECT LASTNAME          -
D                                     FROM CORPDATA.EMPLOYEE WHERE -
D                                     EMPNO = ?')

...

C*************************************************************
C* Prepare STMT as initialized in declare section            *
C*************************************************************
C/EXEC SQL
C+ PREPARE S1 FROM :STMT
C/END-EXEC
C*
C*************************************
C* Declare Cursor for STMT           *
C*************************************
C/EXEC SQL
C+ DECLARE C1 CURSOR FOR S1
C/END-EXEC
C*
C*****************************************************
C* Assign employee number to use in select statement *
C*****************************************************
C                   eval      EMPNUM = '000110'

C**********************
C* Open Cursor        *
C**********************
C/EXEC SQL
C+ OPEN C1 USING :EMPNUM
C/END-EXEC
C*
C***********************************************
C* Fetch record and put value of               *
C* LASTNAME into NAME                          *
C***********************************************
C/EXEC SQL
```

```
C+  FETCH C1 INTO :NAME
C/END-EXEC
...


C********************************
C* Program processes NAME here  *
C********************************
...
C******************
C* Close cursor   *
C******************
C/EXEC SQL
C+  CLOSE C1
C/END-EXEC
```

Coding SQL statements in REXX applications

REXX procedures do not have to be preprocessed. At run time, the REXX interpreter passes statements that it does not understand to the current active command environment for processing.

The command environment can be changed to *EXECSQL to send all unknown statements to the database manager in two ways:

1. CMDENV parameter on the STRREXPRC CL command
2. address positional parameter on the ADDRESS REXX command

For more information about the STRREXPRC CL command or the ADDRESS REXX command, see the REXX/400 Programmer's Guide topic and the REXX/400 Reference topic.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 176.

Related concepts

"Writing applications that use SQL" on page 2
You can create database applications in host languages that use DB2 UDB for iSeries SQL statements and functions.

Related reference

"Handling exception conditions with the WHENEVER statement" on page 11
The WHENEVER statement causes SQL to check the SQLSTATE and SQLCODE and continue processing your program, or branch to another area in your program if an error, exception, or warning exists as a result of running an SQL statement.

"Example: SQL statements in REXX programs" on page 169
This example program is written in the REXX programming language.

Using the SQL communication area in REXX applications

The fields that make up the SQL communication area (SQLCA) are automatically included by the SQL/REXX interface.

An INCLUDE SQLCA statement is not required and is not allowed. The SQLCODE and SQLSTATE fields of the SQLCA contain SQL return codes. These values are set by the database manager after each SQL statement is run. An application can check the SQLCODE or SQLSTATE value to determine whether the last SQL statement was successful.

The SQL/REXX interface uses the SQLCA in a manner consistent with the typical SQL usage. However, the SQL/REXX interface maintains the fields of the SQLCA in separate variables rather than in a contiguous data area. The variables that the SQL/REXX interface maintains for the SQLCA are defined as follows:

SQLCODE
The primary SQL return code.

SQLERRMC
Error and warning message tokens.

SQLERRP
Product code and, if there is an error, the name of the module that returned the error.

SQLERRD.*n*
Six variables (*n* is a number between 1 and 6) containing diagnostic information.

SQLWARN.*n*
Eleven variables (*n* is a number between 0 and 10) containing warning flags.

SQLSTATE
The alternate SQL return code.

Related reference

SQL communication area

Using SQL descriptor areas in REXX applications

There are two types of SQL descriptor areas. One is defined with the ALLOCATE DESCRIPTOR statement. The other is defined using the SQL descriptor area (SQLDA) structure. Only the SQLDA form is discussed here. Allocated descriptors are not supported in REXX.

The following statements can use an SQLDA:

- EXECUTE...USING DESCRIPTOR *descriptor-name*
- FETCH...USING DESCRIPTOR *descriptor-name*
- OPEN...USING DESCRIPTOR *descriptor-name*
- CALL...USING DESCRIPTOR *descriptor-name*
- DESCRIBE *statement-name* INTO *descriptor-name*
- DESCRIBE TABLE *host-variable* INTO *descriptor-name*

Unlike the SQLCA, more than one SQLDA can be in a procedure, and an SQLDA can have any valid name.

Each SQLDA consists of a set of REXX variables with a common stem, where the name of the stem is the *descriptor-name* from the appropriate SQL statements. This must be a simple stem; that is, the stem itself must not contain any periods. The SQL/REXX interface automatically provides the fields of the SQLDA for each unique descriptor name. An INCLUDE SQLDA statement is not required and is not allowed.

The SQL/REXX interface uses the SQLDA in a manner consistent with the typical SQL usage. However, the SQL/REXX interface maintains the fields of the SQLDA in separate variables rather than in a contiguous data area.

The following variables are returned to the application after a DESCRIBE, a DESCRIBE TABLE, or a PREPARE INTO statement:

stem.n.SQLNAME
The name of the nth column in the result table.

The following variables must be provided by the application before an EXECUTE...USING DESCRIPTOR, an OPEN...USING DESCRIPTOR, a CALL...USING DESCRIPTOR, or a FETCH...USING DESCRIPTOR statement. They are returned to the application after a DESCRIBE, a DESCRIBE TABLE, or a PREPARE INTO statement:

stem.SQLD
Number of variable elements that the SQLDA actually contains.

stem.n.SQLTYPE
An integer representing the data type of the nth element (for example, the first element is in stem.1.SQLTYPE).

The following data types are not allowed:

400/401
NUL-terminated graphic string

404/405
BLOB host variable

408/409
CLOB host variable

412/413
DBCLOB host variable

460/461
NUL-terminated character string

476/477
PASCAL L-string

496/497
Large integer (where scale is greater than 0)

500/501
Small integer (where scale is greater than 0)

504/505
DISPLAY SIGN LEADING SEPARATE

904/905
ROWID

908/909
VARBINARY host variable

912/913
BINARY host variable

916/917
BLOB file reference variable

920/921
CLOB file reference variable

924/925
DBCLOB file reference variable

960/961
BLOB locator

964/965
CLOB locator

968/969
DBCLOB locator

stem.n.SQLLEN
If SQLTYPE does not indicate a DECIMAL or NUMERIC data type, the maximum length of the data contained in stem.n.SQLDATA.

stem.n.SQLLEN.SQLPRECISION
If the data type is DECIMAL or NUMERIC, this contains the precision of the number.

stem.n.SQLLEN.SQLSCALE
If the type is DECIMAL or NUMERIC, this contains the scale of the number.

stem.n.SQLCCSID
The CCSID of the nth column of the data.

The following variables must be provided by the application before an EXECUTE...USING DESCRIPTOR or an OPEN...USING DESCRIPTOR statement, and they are returned to the application after a FETCH...USING DESCRIPTOR statement. They are not used after a DESCRIBE, a DESCRIBE TABLE, or a PREPARE INTO statement:

stem.n.SQLDATA
This contains the input value supplied by the application, or the output value fetched by SQL.

This value is converted to the attributes specified in SQLTYPE, SQLLEN, SQLPRECISION, and SQLSCALE.

stem.n.SQLIND
If the input or output value is null, this is a negative number.

Related reference

SQL descriptor area

Embedding SQL statements in REXX applications

An SQL statement can be placed anywhere a REXX command can be placed.

Each SQL statement in a REXX procedure must begin with EXECSQL (in any combination of uppercase and lowercase letters), followed by either:

- The SQL statement enclosed in single or double quotation marks, or
- A REXX variable containing the statement. Note that a colon must not precede a REXX variable when it contains an SQL statement.

For example:

```
EXECSQL "COMMIT"
```

is equivalent to:

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

The command follows normal REXX rules. For example, it can optionally be followed by a semicolon (;) to allow a single line to contain more than one REXX statement. REXX also permits command names to be included within single quotation marks, for example:

```
'EXECSQL COMMIT'
```

The SQL/REXX interface supports the following SQL statements:

ALTER SEQUENCE
ALTER TABLE
CALL [2]
CLOSE
COMMENT ON
COMMIT
CREATE ALIAS
CREATE DISTINCT TYPE
CREATE FUNCTION
CREATE INDEX
CREATE PROCEDURE
CREATE SCHEMA
CREATE SEQUENCE
CREATE TABLE
CREATE TRIGGER
CREATE VIEW
DECLARE CURSOR [2]
DECLARE GLOBAL TEMPORARY TABLE
DELETE [2]
DESCRIBE
DESCRIBE TABLE
DROP
EXECUTE
EXECUTE IMMEDIATE
FETCH [1]
GRANT
INSERT [1]
LABEL ON
LOCK TABLE
OPEN
PREPARE
REFRESH
RELEASE SAVEPOINT
RENAME
REVOKE
ROLLBACK
SAVEPOINT
SET ENCRYPTION PASSWORD
SET OPTION [3]
SET PATH
SET SCHEMA
SET TRANSACTION
SET variable [2]
UPDATE [2]
VALUES INTO [2]

The following SQL statements are not supported by the SQL/REXX interface:

ALLOCATE DESCRIPTOR
BEGIN DECLARE SECTION
CONNECT
DEALLOCATE DESCRIPTOR
DECLARE PROCEDURE
DECLARE STATEMENT
DECLARE VARIABLE
DESCRIBE INPUT
DISCONNECT
END DECLARE SECTION
FREE LOCATOR
GET DESCRIPTOR
GET DIAGNOSTICS
HOLD LOCATOR
INCLUDE
RELEASE
SELECT INTO
SET CONNECTION
SET CURRENT DEGREE
SET DESCRIPTOR
SET RESULT SETS
SET SESSION AUTHORIZATION
SIGNAL
WHENEVER[4]

1. The blocked form of this statement is not supported.
2. These statements cannot be run directly if they contain host variables; they must be the object of a PREPARE and then an EXECUTE.
3. The SET OPTION statement can be used in a REXX procedure to change some of the processing options used for running SQL statements. These options include the commitment control level and date format. See the SQL reference topic for more information about the SET OPTION statement.
4. See “Handling errors and warnings in REXX applications that use SQL” on page 119 for more information.

Comments in REXX applications that use SQL

Neither SQL comments (--) nor REXX comments are allowed in strings representing SQL statements.

Continuation of SQL statements in REXX applications that use SQL

The string containing an SQL statement can be split into several strings on several lines, separated by commas or concatenation operators, according to standard REXX usage.

Including code in REXX applications that use SQL

Unlike the other host languages, support is not provided for including externally defined statements.

Margins in REXX applications that use SQL

There are no special margin rules for the SQL/REXX interface.

Names in REXX applications that use SQL

Any valid REXX name not ending in a period (.) can be used for a host variable. The name must be 64 characters or less.

Variable names should not begin with the characters ’SQL’, ’RDI’, ’DSN’, ’RXSQL’, or ’QRW’.

Nulls in REXX applications that use SQL

Although the term *null* is used in both REXX and SQL, the term has different meanings in the two languages.

REXX has a null string (a string of length zero) and a null clause (a clause consisting only of blanks and comments). The SQL null value is a special value that is distinct from all non-null values and denotes the absence of a (non-null) value.

Statement labels in REXX applications that use SQL

REXX command statements can be labeled as usual.

Handling errors and warnings in REXX applications that use SQL

The WHENEVER statement is not supported by the SQL/REXX interface. You can use one of several substitutes, however.

Any of the following may be used instead:

- A test of the REXX SQLCODE or SQLSTATE variables after each SQL statement to detect error and warning conditions issued by the database manager, but not for those issued by the SQL/REXX interface.
- A test of the REXX RC variable after each SQL statement to detect error and warning conditions. Each use of the EXECSQL command sets the RC variable to:

  0 Statement completed successfully.

  +10 A SQL warning occurred.

  -10 An SQL error occurred

  -100 An SQL/REXX interface error occurred.

  This can be used to detect errors and warnings issued by either the database manager or by the SQL/REXX interface.
- The SIGNAL ON ERROR and SIGNAL ON FAILURE facilities can be used to detect errors (negative RC values), but not warnings.

Using host variables in REXX applications that use SQL

REXX does not provide for variable declarations.

LOB, ROWID, and binary host variables are not supported in REXX. New variables are recognized by their appearance in assignment statements. Therefore, there is no declare section, and the BEGIN DECLARE SECTION and END DECLARE SECTION statements are not supported.

All host variables within an SQL statement must be preceded by a colon (:).

The SQL/REXX interface performs substitution in compound variables before passing statements to the database manager. For example:

```
a = 1
b = 2
EXECSQL 'OPEN c1 USING :x.a.b'
```

causes the contents of x.1.2 to be passed to SQL.

Determining data types of input host variables in REXX applications that use SQL

All data in REXX is in the form of strings.

The data type of input host variables (that is, host variables used in a ’USING host variable’ clause in an EXECUTE or OPEN statement) is inferred by the database manager at run time from the contents of the variable according to the table below.

These rules define either numeric, character, or graphic values. A numeric value can be used as input to a numeric column of any type. A character value can be used as input to a character column of any type, or to a date, time, or timestamp column. A graphic value can be used as input to a graphic column of any type.

Table 11. Determining data types of host variables in REXX

Host variable contents	Assumed data type	SQL type code	SQL type description
A number with neither decimal point nor exponent. It can have a leading plus or minus sign.	Signed integers	496/497	INTEGER
A number that includes a decimal point, but no exponent, or a number that does not include a decimal point or an exponent and is greater than 2147483647 or smaller than -2147483647. It can have a leading plus or minus sign. *m* is the total number of digits in the number. *n* is the number of digits to the left of the decimal point (if any).	Packed decimal	484/485	DECIMAL(m,n)
A number that is in scientific or engineering notation (that is, followed immediately by an ’E’ or ’e’, an optional plus or minus sign, and a series of digits). It can have a leading plus or minus sign.	Floating point	480/481	DOUBLE PRECISION
A string with leading and trailing single quotation marks (′) or quotation marks (″), which has length n after removing the two delimiters, or a string with a leading X or x followed by a single quotation mark (′) or quotation mark (″), and a trailing single quotation mark (′) or quotation mark (″). The string has a length of 2n after removing the X or x and the two delimiters. Each remaining pair of characters is the hexadecimal representation of a single character. or a string of length n, which cannot be recognized as character, numeric, or graphic through other rules in this table	Varying-length character string	448/449	VARCHAR(n)

Table 11. Determining data types of host variables in REXX (continued)

Host variable contents	Assumed data type	SQL type code	SQL type description
A string with a leading and trailing single quotation mark (′) or quotation marks (″) preceded by: [1] • A string that starts with a G, g, N, or n. This is followed by a single quotation mark or quotation mark and a shift-out (x′0E′). This is followed by n graphic characters, each 2 characters long. The string must end with a shift-in (X′0F′) and a quotation mark or quotation mark (whichever the string started with). • A string with a leading GX, Gx, gX, or gx, followed by a quotation mark or quotation mark and a shift-out (x′0E′). This is followed by n graphic characters, each 2 characters long. The string must end with a shift-in (X′0F′) and a quotation mark or quotation mark (whichever the string started with). The string has a length of 4n after removing the GX and the delimiters. Each remaining group of 4 characters is the hexadecimal representation of a single graphic character.	Varying-length graphic string	464/465	VARGRAPHIC(n)
Undefined Variable	Variable for which a value has not been assigned	None	Data that is not valid was detected.

Note: The byte immediately following the leading single quotation mark is a X′0E′ shift-out, and the byte immediately preceding the trailing single quotation mark is a X′0F′ shift-in.

The format of output host variables in REXX applications that use SQL

It is not necessary to determine the data type of an *output host variable* (that is, a host variable used in an ′INTO host variable′ clause in a FETCH statement).

Output values are assigned to host variables as follows:

- Character values are assigned without leading and trailing apostrophes.
- Graphic values are assigned without a leading G or apostrophe, without a trailing apostrophe, and without shift-out and shift-in characters.
- Numeric values are translated into strings.
- Integer values do not retain any leading zeros. Negative values have a leading minus sign.
- Decimal values retain leading and trailing zeros according to their precision and scale. Negative values have a leading minus sign. Positive values do not have a leading plus sign.
- Floating-point values are in scientific notation, with one digit to the left of the decimal place. The ′E′ is in uppercase.

Avoiding REXX conversion in REXX applications that use SQL

To guarantee that a string is not converted to a number or assumed to be of graphic type, strings should be enclosed in "′". Enclosing the string in single quotation marks does not work.

For example:

```
stringvar = '100'
```

causes REXX to set the variable *stringvar* to the string of characters 100 (without the single quotation marks). This is evaluated by the SQL/REXX interface as the number 100, and it is passed to SQL as such.

On the other hand,

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

causes REXX to set the variable *stringvar* to the string of characters ’100’ (with the single quotation marks). This is evaluated by the SQL/REXX interface as the string 100, and it is passed to SQL as such.

Using indicator variables in REXX applications that use SQL

An indicator variable is an integer.

On retrieval, an indicator variable is used to show if its associated host variable was assigned a null value. On assignment to a column, a negative indicator variable is used to indicate that a null value should be assigned.

Unlike other languages, a valid value must be specified in the host variable even if its associated indicator variable contains a negative value.

Related reference

References to variables

Preparing and running a program with SQL statements

This topic describes some of the tasks for preparing and running an application program.

Related concepts

“Writing applications that use SQL” on page 2
You can create database applications in host languages that use DB2 UDB for iSeries SQL statements and functions.

Basic processes of the SQL precompiler

You must precompile and compile an application program containing embedded SQL statements before you can run it.

Note: SQL statements in a REXX procedure are not precompiled and compiled.

Precompiling of such programs is done by the SQL precompiler. The SQL precompiler scans each statement of the application program source and does the following:

- **Looks for SQL statements and for the definition of host variable names.** The variable names and definitions are used to verify the SQL statements. You can examine the listing after the SQL precompiler completes processing to see if any errors occurred.
- **Verifies that each SQL statement is valid and free of syntax errors.** The validation procedure supplies error messages in the output listing that help you correct any errors that occur.
- **Validates the SQL statements using the description in the database.** During the precompile, the SQL statements are checked for valid table, view, and column names. If a specified table or view does not exist, or you are not authorized to the table or view at the time of the precompile or compile, the validation is done at run time. If the table or view does not exist at run time, an error occurs.

Notes:

1. Overrides are processed when retrieving external definitions.
2. You need some authority (at least *OBJOPR) to any tables or views referred to in the SQL statements in order to validate the SQL statements. The actual authority required to process any SQL statement is checked at run time.

3. When the RDB parameter is specified on the CRTSQLxxx commands, the precompiler accesses the specified relational database to obtain the table and view descriptions.

- **Prepares each SQL statement for compilation in the host language.** For most SQL statements, the SQL precompiler inserts a comment and a CALL statement to one of the SQL interface modules. For some SQL statements (for example, DECLARE statements), the SQL precompiler produces no host language statement except a comment.
- **Produces information about each precompiled SQL statement.** The information is stored internally in a temporary source file member, where it is available for use during the bind process.

To get complete diagnostic information when you precompile, specify either of the following:

- OPTION(*SOURCE *XREF) for CRTSQLxxx (where xxx=CBL, PLI, or RPG)
- OPTION(*XREF) OUTPUT(*PRINT) for CRTSQLxxx (where xxx=CI, CPPI, CBLI, or RPGI)

Related concepts

Database programming

Database file management

SQL reference

Input to the SQL precompiler

Application programming statements and embedded SQL statements are the primary input to the SQL precompiler.

In PL/I, C, and C++ programs, the SQL statements must use the margins that are specified in the MARGINS parameter of the CRTSQLPLI, CRTSQLCI, and CRTSQLCPPI commands.

The SQL precompiler assumes that the host language statements are syntactically correct. If the host language statements are not syntactically correct, the precompiler may not correctly identify SQL statements and host variable declarations. There are limits on the forms of source statements that can be passed through the precompiler. Literals and comments that are not accepted by the application language compiler, can interfere with the precompiler source scanning process and cause errors.

You can use the SQL INCLUDE statement to get secondary input from the file that is specified by the INCFILE parameter of the CRTSQLxxx. The xxx in this command refers to the host language indicators: CBL for the OPM COBOL language, CBLI for the ILE COBOL language, PLI for the PL/I PRPQ language, CI for the ILE C language, RPG for the RPG/400 language, RPGI for the ILE RPG language, and CPPI for the ILE C++ language. The SQL INCLUDE statement causes input to be read from the specified member until it reaches the end of the member. The included member cannot contain other precompiler INCLUDE statements, but can contain both application program and SQL statements.

If mixed DBCS constants are specified in the application program source, the source file must be a mixed CCSID.

You can specify many of the precompiler options in the input source member by using the SQL SET OPTION statement.

The RPG preprocessor options (RPGPPORT) parameter of the CRTSQLRPGI command has two options to call the RPG preprocessor. If *LVL1 or *LVL2 is specified, the RPG compiler will be called to preprocess the source member before the SQL precompile is run. Preprocessing the SQL source member will allow many compiler directives to be handled before the SQL precompile. The preprocessed source will be placed in file QSQLPRE in QTEMP. This source will be used as the input for the SQL precompile. The CCSID used by the SQL precompile is the CCSID of QSQLPRE.

Related reference

SET OPTION

Create SQL ILE RPG Object (CRTSQLRPGI) command

Source file CCSIDs in the SQL precompiler

The SQL precompiler reads the source records by using the CCSID of the source file.

When processing SQL INCLUDE statements, the include source is converted to the CCSID of the original source file if necessary. If the include source cannot be converted to the CCSID of the original source file, an error occurs.

The SQL precompiler processes SQL statements using the source CCSID. This affects variant characters the most. For example, the not sign (¬) is located at ’BA’X in CCSID 500. This means that if the CCSID of your source file is 500, SQL expects the not sign (¬) to be located at ’BA’X.

If the source file CCSID is 65535, SQL processes variant characters as if they had a CCSID of 37. This means that SQL looks for the not sign (¬) at ’5F’X.

Output from the SQL precompiler

The SQL precompiler generates two pieces of output: a listing and a source file number.

Listing:

The output listing is sent to the printer file that is specified by the PRTFILE parameter of the CRTSQLxxx command.

The following items are written to the printer file:

- Precompiler options

  Options specified in the CRTSQLxxx command.
- Precompiler source

  This output supplies precompiler source statements with the record numbers that are assigned by the precompiler, if the listing option is in effect.
- Precompiler cross-reference

  If *XREF was specified in the OPTION parameter, this output supplies a cross-reference listing. The listing shows the precompiler record numbers of SQL statements that contain the referred to host names and column names.
- Precompiler diagnostics

  This output supplies diagnostic messages, showing the precompiler record numbers of statements in error.

  The output to the printer file will use a CCSID value of 65535. The data will not be converted when it is written to the printer file.

Temporary source file members created by the SQL precompiler:

Source statements processed by the precompiler are written to an output source file.

In the precompiler-changed source code, SQL statements have been converted to comments and calls to the SQL run time code. Include files that are processed by SQL are expanded.

The output source file is specified on the CRTSQLxxx command in the TOSRCFILE parameter. For languages other than C and C++, the default file is QSQLTEMP (QSQLTEMP1 for ILE RPG) in the QTEMP library. For C and C++ when *CALC is specified as the output source file, QSQLTEMP will be used if the source file’s record length is 92 or less. For a C or C++ source file where the record length is greater than 92, the output source file name will be generated as QSQLTxxxxx, where xxxxx is the record length. The name of the output source file member is the same as the name specified in the PGM or OBJ parameter of the CRTSQLxxx command. This member cannot be changed before being used as input to the compiler. When SQL creates the output source file, it uses the CCSID value of the source file as the CCSID value for the new file.

If the precompile generates output in a source file in QTEMP, the file can be moved to a permanent library after the precompile if you want to compile at a later time. You cannot change the records of the source member, or the attempted compile fails.

The source member that is generated by SQL as the result of the precompile should never be edited and reused as an input member to another precompile step. The additional SQL information that is saved with the source member during the first precompile will cause the second precompile to work incorrectly. Once this information is attached to a source member, it stays with the member until the member is deleted.

The SQL precompiler uses the CRTSRCPF command to create the output source file. If the defaults for this command have changed, then the results may be unpredictable. If the source file is created by the user, not the SQL precompiler, the file's attributes may be different as well. It is recommended that the user allow SQL to create the output source file. Once it has been created by SQL, it can be reused on later precompiles.

Sample SQL precompiler output:

The precompiler output can provide information about your program source.

To generate the listing:

- For non-ILE precompilers, specify the *SOURCE (*SRC) and *XREF options on the OPTION parameter of the CRTSQLxxx command.
- For ILE precompilers, specify OPTION(*XREF) and OUTPUT(*PRINT) on the CRTSQLxxx command.

The format of the precompiler output is:

```
5722ST1 V5R4M0 060210         Create SQL COBOL Program       CBLTEST1          08/06/02 11:14:21   Page   1
Source type...............COBOL
Program name..............CORPDATA/CBLTEST1
Source file...............CORPDATA/SRC
Member....................CBLTEST1
To source file............QTEMP/QSQLTEMP
(1)Options...................*SRC      *XREF     *SQL
Target release............V5R4M0
INCLUDE file..............*SRCFILE
Commit....................*CHG
Allow copy of data........*YES
Close SQL cursor..........*ENDPGM
Allow blocking............*READ
Delay PREPARE.............*NO
Generation level..........10
Printer file..............*LIBL/QSYSPRT
Date format...............*JOB
Date separator............*JOB
Time format...............*HMS
Time separator ...........*JOB
Replace...................*YES
Relational database.......*LOCAL
User .....................*CURRENT
RDB connect method........*DUW
Default Collection........*NONE
Dynamic default
   collection..............*NO
Package name..............*PGMLIB/*PGM
Path......................*NAMING
SQL rules.................*DB2
User profile..............*NAMING
Dynamic User Profile......*USER
Sort Sequence.............*JOB
Language ID...............*JOB
IBM SQL flagging..........*NOFLAG
ANS flagging..............*NONE
Text......................*SRCMBRTXT
Source file CCSID.........65535
Job CCSID.................65535
Decimal result options:
  Maximum precision.......31
  Maximum scale...........31
  Minimum divide scale....0
Compiler options..........*NONE
(2) Source member changed on 06/06/00  10:16:44
```

1 A list of the options you specified when the SQL precompiler was called.

2 The date the source member was last changed.

Figure 2. Sample COBOL precompiler output format

```
5722ST1 V5R4M0 060210       Create SQL COBOL Program  CBLTEST1                   08/06/02 11:14:21   Page   2
(1)Record  *...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8   (2)SEQNBR  (3)Last Change

    1          IDENTIFICATION DIVISION.                                                       100
    2          PROGRAM-ID.  CBLTEST1.                                                         200
    3          ENVIRONMENT DIVISION.                                                          300
    4          CONFIGURATION SECTION.                                                         400
    5          SOURCE-COMPUTER. IBM-AS400.                                                    500
    6          OBJECT-COMPUTER. IBM-AS400.                                                    600
    7          INPUT-OUTPUT SECTION.                                                          700
    8          FILE-CONTROL.                                                                  800
    9              SELECT OUTFILE, ASSIGN TO PRINTER-QPRINT,                                  900
   10                 FILE STATUS IS FSTAT.                                                  1000
   11          DATA DIVISION.                                                                1100
   12          FILE SECTION.                                                                 1200
   13          FD  OUTFILE                                                                   1300
   14              DATA RECORD IS REC-1,                                                     1400
   15              LABEL RECORDS ARE OMITTED.                                                1500
   16          01  REC-1.                                                                    1600
   17              05  CC                     PIC X.                                         1700
   18              05  DEPT-NO                PIC X(3).                                      1800
   19              05  FILLER                 PIC X(5).                                      1900
   20              05  AVERAGE-EDUCATION-LEVEL PIC ZZZ.                                      2000
   21              05  FILLER                  PIC X(5).                                     2100
   22              05  AVERAGE-SALARY          PIC ZZZZ9.99.                                 2200
   23          01  ERROR-RECORD.                                                             2300
   24              05  CC                      PIC X.                                        2400
   25              05  ERROR-CODE              PIC S9(5).                                    2500
   26              05  ERROR-MESSAGE           PIC X(70).                                    2600
   27          WORKING-STORAGE SECTION.                                                      2700
   28              EXEC SQL                                                                  2800
   29                INCLUDE SQLCA                                                           2900
   30              END-EXEC.                                                                 3000
   31          77  FSTAT                       PIC XX.                                       3100
   32          01  AVG-RECORD.                                                               3200
   33              05  WORKDEPT                PIC X(3).                                     3300
   34              05  AVG-EDUC                PIC S9(4) USAGE COMP-4.                       3400
   35              05  AVG-SALARY              PIC S9(6)V99 COMP-3.                          3500
   36          PROCEDURE DIVISION.                                                           3600
   37         ***************************************************************                3700
   38         *  This program will get the average education level and the  *                3800
   39         *  average salary by department.                              *                3900
   40         ***************************************************************                4000
   41          A000-MAIN-PROCEDURE.                                                          4100
   42              OPEN OUTPUT OUTFILE.                                                      4200
   43         ***************************************************************                4300
   44         * Set-up WHENEVER statement to handle SQL errors.             *                4400
   45         ***************************************************************                4500
   46              EXEC SQL                                                                  4600
   47                WHENEVER SQLERROR GO TO B000-SQL-ERROR                                  4700
   48              END-EXEC.                                                                 4800
```

1 Record number assigned by the precompiler when it reads the source record. Record numbers are used to identify the source record in error messages and SQL run-time processing.

2 Sequence number taken from the source record. The sequence number is the number seen when you use the source entry utility (SEU) to edit the source member.

3 Date when the source record was last changed. If Last Change is blank, it indicates that the record has not been changed since it was created.

```
5722ST1 V5R4M0 060210      Create SQL COBOL Program          CBLTEST1            08/06/02 11:14:21    Page    3
Record  *...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8   SEQNBR  Last change
   49         ***************************************************************                4900
   50         * Declare cursor                                              *                5000
   51         ***************************************************************                5100
   52              EXEC SQL                                                                  5200
   53                DECLARE CURS CURSOR FOR                                                 5300
   54                  SELECT WORKDEPT, AVG(EDLEVEL), AVG(SALARY)                            5400
   55                    FROM CORPDATA.EMPLOYEE                                              5500
   56                    GROUP BY WORKDEPT                                                   5600
   57              END-EXEC.                                                                 5700
   58         ***************************************************************                5800
   59         *  Open cursor                                                *                5900
   60         ***************************************************************                6000
   61              EXEC SQL                                                                  6100
   62                OPEN CURS                                                               6200
   63              END-EXEC.                                                                 6300
   64         ***************************************************************                6400
   65         *  Fetch all result rows                                      *                6500
   66         ***************************************************************                6600
   67              PERFORM A010-FETCH-PROCEDURE THROUGH A010-FETCH-EXIT                      6700
   68                UNTIL SQLCODE IS = 100.                                                 6800
   69         ***************************************************************                6900
   70         *  Close cursor                                               *                7000
   71         ***************************************************************                7100
   72              EXEC SQL                                                                  7200
   73                CLOSE CURS                                                              7300
   74              END-EXEC.                                                                 7400
   75              CLOSE OUTFILE.                                                            7500
   76              STOP RUN.                                                                 7600
   77         ***************************************************************                7700
   78         *  Fetch a row and move the information to the output record. *                7800
   79         ***************************************************************                7900
   80          A010-FETCH-PROCEDURE.                                                         8000
   81              MOVE SPACES TO REC-1.                                                     8100
   82              EXEC SQL                                                                  8200
   83                FETCH CURS INTO :AVG-RECORD                                             8300
   84              END-EXEC.                                                                 8400
   85              IF SQLCODE IS = 0                                                         8500
   86                MOVE WORKDEPT TO DEPT-NO                                                8600
   87                MOVE AVG-SALARY TO AVERAGE-SALARY                                       8700
   88                MOVE AVG-EDUC TO AVERAGE-EDUCATION-LEVEL                                8800
   89                WRITE REC-1 AFTER ADVANCING 1 LINE.                                     8900
   90          A010-FETCH-EXIT.                                                              9000
   91              EXIT.                                                                     9100
   92         ***************************************************************                9200
   93         *  An SQL error occurred.  Move the error number to the error *                9300
   94         *  record and stop running.                                   *                9400
   95         ***************************************************************                9500
   96          B000-SQL-ERROR.                                                               9600
   97              MOVE SPACES TO ERROR-RECORD.                                              9700
   98              MOVE SQLCODE TO ERROR-CODE.                                               9800
   99              MOVE "AN SQL ERROR HAS OCCURRED" TO ERROR-MESSAGE.                        9900
  100              WRITE ERROR-RECORD AFTER ADVANCING 1 LINE.                               10000
  101              CLOSE OUTFILE.                                                           10100
  102              STOP RUN.                                                                10200
* * * * *  E N D  O F  S O U R C E  * * * * *
```

```
5722ST1 V5R4M0 060210          Create SQL COBOL Program      CBLTEST1              08/06/02 11:14:21   Page     4
CROSS REFERENCE
1                             2       3
Data Names                    Define    Reference
AVERAGE-EDUCATION-LEVEL          20       IN REC-1
AVERAGE-SALARY                   22       IN REC-1
AVG-EDUC                         34       SMALL INTEGER PRECISION(4,0) IN AVG-RECORD
AVG-RECORD                       32       STRUCTURE
                                          83
AVG-SALARY                       35       DECIMAL(8,2) IN AVG-RECORD
BIRTHDATE                        55       DATE(10) COLUMN IN CORPDATA.EMPLOYEE
BONUS                            55       DECIMAL(9,2) COLUMN IN CORPDATA.EMPLOYEE
B000-SQL-ERROR                   ****     LABEL
                                          47
CC                               17       CHARACTER(1) IN REC-1
CC                               24       CHARACTER(1) IN ERROR-RECORD
COMM                             55       DECIMAL(9,2) COLUMN IN CORPDATA.EMPLOYEE
CORPDATA                         ****   (4) COLLECTION
                                        (5) 55
CURS                             53       CURSOR
                                          62 73 83
DEPT-NO                          18       CHARACTER(3) IN REC-1
EDLEVEL                          ****     COLUMN
                                          54
                                           (6)
EDLEVEL                          55       SMALL INTEGER PRECISION(4,0) COLUMN (NOT NULL) IN CORPDATA.EMPLOYEE
EMPLOYEE                         ****     TABLE IN CORPDATA                                      (7)
                                          55
EMPNO                            55       CHARACTER(6) COLUMN (NOT NULL) IN CORPDATA.EMPLOYEE
ERROR-CODE                       25       NUMERIC(5,0) IN ERROR-RECORD
ERROR-MESSAGE                    26       CHARACTER(70) IN ERROR-RECORD
ERROR-RECORD                     23       STRUCTURE
FIRSTNME                         55       VARCHAR(12) COLUMN (NOT NULL) IN CORPDATA.EMPLOYEE
FSTAT                            31       CHARACTER(2)
HIREDATE                         55       DATE(10) COLUMN IN CORPDATA.EMPLOYEE
JOB                              55       CHARACTER(8) COLUMN IN CORPDATA.EMPLOYEE
LASTNAME                         55       VARCHAR(15) COLUMN (NOT NULL) IN CORPDATA.EMPLOYEE
MIDINIT                          55       CHARACTER(1) COLUMN (NOT NULL) IN CORPDATA.EMPLOYEE
PHONENO                          55       CHARACTER(4) COLUMN IN CORPDATA.EMPLOYEE
REC-1                            16
SALARY                           ****     COLUMN
                                          54
SALARY                           55       DECIMAL(9,2) COLUMN IN CORPDATA.EMPLOYEE
SEX                              55       CHARACTER(1) COLUMN IN CORPDATA.EMPLOYEE
WORKDEPT                         33       CHARACTER(3) IN AVG-RECORD
WORKDEPT                         ****     COLUMN
                                          54 56
WORKDEPT                         55       CHARACTER(3) COLUMN IN CORPDATA.EMPLOYEE
No errors found in source
102 Source records processed
* * * * *  E N D  O F  L I S T I N G  * * * * *
```

1 Data names are the symbolic names used in source statements.

2 The define column specifies the line number at which the name is defined. The line number is generated by the SQL precompiler. **** means that the object was not defined or the precompiler did not recognize the declarations.

3 The reference column contains two types of information:

- What the symbolic name is defined as 4
- The line numbers where the symbolic name occurs 5

If the symbolic name refers to a valid host variable, the data-type 6 or data-structure 7 is also noted.

Non-ILE SQL precompiler commands

The DB2 UDB Query Manager and SQL Development Kit licensed program includes non-ILE precompiler commands for the following host languages: CRTSQLCBL (for OPM COBOL), CRTSQLPLI (for PL/I PRPQ), and CRTSQLRPG (for RPG III, which is part of RPG/400).

Some options only apply to certain languages. For example, the options *APOST and *QUOTE are unique to COBOL. They are not included in the commands for the other languages.

Related concepts

"CL command descriptions for host language precompilers" on page 174
The DB2 UDB for iSeries database provides commands for precompiling programs coded in these programming languages.

Compiling a non-ILE application program that uses SQL

The SQL precompiler automatically calls the host language compiler after the successful completion of a precompile, unless *NOGEN is specified.

The CRTxxxPGM command is run specifying the program name, source file name, precompiler created source member name, text, and USRPRF.

Within these languages, the following parameters are passed:

- For COBOL, the *QUOTE or *APOST is passed on the CRTCBLPGM command.
- For RPG and COBOL, SAAFLAG (*FLAG) is passed on the CRTxxxPGM command.
- For RPG and COBOL, the SRTSEQ and LANGID parameter from the CRTSQLxxx command is specified on the CRTxxxPGM command.
- For RPG and COBOL, the CVTOPT (*DATETIME *VARCHAR) is always specified on the CRTxxxPGM command.
- For COBOL and RPG, the TGTRLS parameter value from the CRTSQLxxx command is specified on the CRTxxxPGM command. TGTRLS is not specified on the CRTPLIPGM command. The program can be saved or restored to the level specified on the TGTRLS parameter of the CRTSQLPLI command.
- For PL/I, the MARGINS are set in the temporary source file.
- For all languages, the REPLACE parameter from the CRTSQLxxx command is specified on the CRTxxxPGM command.

  If a package is created as part of the precompile process, the REPLACE parameter value from the CRTSQLxxx command is specified on the CRTSQLPKG command.
- For all languages, if USRPRF(*USER) or system naming (*SYS) with USRPRF(*NAMING) is specified, then USRPRF(*USER) is specified on the CRTxxxPGM command. If USRPRF(*OWNER) or SQL naming (*SQL) with USRPRF(*NAMING) is specified, then USRPRF(*OWNER) is specified on the CRTxxxPGM command.

Defaults are used for all other parameters with CRTxxxPGM commands.

You can interrupt the call to the host language compiler by specifying *NOGEN on the OPTION parameter of the precompiler command. *NOGEN specifies that the host language compiler will not be called. Using the object name in the CRTSQLxxx command as the member name, the precompiler created the source member in the output source file (specified as the TOSRCFILE parameter on the CRTSQLxxx command). You now can explicitly call the host language compilers, specify the source member in the output source file, and change the defaults. If the precompile and compile were done as separate steps, the CRTSQLPKG command can be used to create the SQL package for a distributed program.

Note: You must not change the source member in QTEMP/QSQLTEMP prior to issuing the CRTxxxPGM command or the compile will fail.

ILE SQL precompiler commands

In the DB2 UDB Query Manager and SQL Development Kit licensed program, these ILE precompiler commands exist: CRTSQLCI, CRTSQLCPPI, CRTSQLCBLI, and CRTSQLRPGI.

A precompiler command exists for each of the host languages: ILE C, ILE C++, ILE COBOL, and ILE RPG. Separate commands, by language, let you specify the required parameters and take the default for the remaining parameters. The defaults are applicable only to the language you are using. For example, the options *APOST and *QUOTE are unique to COBOL. They are not included in the commands for the other languages.

Related concepts

"CL command descriptions for host language precompilers" on page 174
The DB2 UDB for iSeries database provides commands for precompiling programs coded in these programming languages.

Compiling an ILE application program that uses SQL

The SQL precompiler automatically calls the host language compiler after the successful completion of a precompile for the CRTSQLxxx commands, unless *NOGEN is specified.

If the *MODULE option is specified, the SQL precompiler issues the CRTxxxMOD command to create the module. If the *PGM option is specified, the SQL precompiler issues the CRTBNDxxx command to create the program. If the *SRVPGM option is specified, the SQL precompiler issues the CRTxxxMOD command to create the module, followed by the Create Service Program (CRTSRVPGM) command to create the service program. The CRTSQLCPPI command only creates *MODULE objects.

Within these languages, the following parameters are passed:

- If DBGVIEW(*SOURCE) is specified on the CRTSQLxxx command, then DBGVIEW(*ALL) is specified on both the CRTxxxMOD and CRTBNDxxx commands.
- If OUTPUT(*PRINT) is specified on the CRTSQLxxx command, it is passed on both the CRTxxxMOD and CRTBNDxxx commands.

  If OUTPUT(*NONE) is specified on the CRTSQLxxx command, it is not specified on either the CRTxxxMOD command or the CRTBNDxxx command.
- The TGTRLS parameter value from the CRTSQLxxx command is specified on the CRTxxxMOD, CRTBNDxxx, and Create Service Program (CRTSRVPGM) commands.
- The REPLACE parameter value from the CRTSQLxxx command is specified on the CRTxxxMOD, CRTBNDxxx, and CRTSRVPGM commands.

  If a package is created as part of the precompile process, the REPLACE parameter value from the CRTSQLxxx command is specified on the CRTSQLPKG command.
- If OBJTYPE is either *PGM or *SRVPGM, and USRPRF(*USER) or system naming (*SYS) with USRPRF(*NAMING) is specified, USRPRF(*USER) is specified on the CRTBNDxxx or the CRTSRVPGM commands.

  If OBJTYPE is either *PGM or *SRVPGM, and USRPRF(*OWNER) or SQL naming (*SQL) with USRPRF(*NAMING) is specified, USRPRF(*OWNER) is specified on the CRTBNDxxx or the CRTSRVPGM commands.
- For C and C++, the MARGINS are set in the temporary source file.

  If the precompiler calculates that the total length of the LOB host variables is close to 15M, the TERASPACE(*YES *TSIFC) option is specified on the CRTCMOD, CRTBNDC, or CRTCPPMOD commands.
- For COBOL, the *QUOTE or *APOST is passed on the CRTBNDCBL or the CRTCBLMOD commands.
- FOR RPG and COBOL, the SRTSEQ and LANGID parameter from the CRTSQLxxx command is specified on the CRTxxxMOD and CRTBNDxxx commands.
- For COBOL, CVTOPT(*VARCHAR *DATETIME *PICGGRAPHIC *FLOAT) is always specified on the CRTCBLMOD and CRTBNDCBL commands. If OPTION(*NOCVTDT) is specified (the shipped command default), the additional options *DATE *TIME *TIMESTAMP are also specified for the CVTOPT.
- For RPG, if OPTION(*CVTDT) is specified, then CVTOPT(*DATETIME) is specified on the CRTRPGMOD and CRTBNDRPG commands.

You can interrupt the call to the host language compiler by specifying *NOGEN on the OPTION parameter of the precompiler command. *NOGEN specifies that the host language compiler is not called. Using the specified program name in the CRTSQLxxx command as the member name, the precompiler creates the source member in the output source file (TOSRCFILE parameter). You can now explicitly call the host language compiler, specify the source member in the output source file, and change the defaults. If the precompile and compile were done as separate steps, the CRTSQLPKG command can be used to create the SQL package for a distributed program.

If the program or service program is created later, the USRPRF parameter may not be set correctly on the CRTBNDxxx, Create Program (CRTPGM), or Create Service Program (CRTSRVPGM) command. The SQL program runs predictably only after the USRPRF parameter is corrected. If system naming is used, then the USRPRF parameter must be set to *USER. If SQL naming is used, then the USRPRF parameter must be set to *OWNER.

Setting compiler options using the precompiler commands

The COMPILEOPT string is available on the precompiler command and on the SET OPTION statement to allow additional parameters to be used on the compiler command.

The COMPILEOPT string is added to the compiler command built by the precompiler. This allows specifying compiler parameters without requiring a two step process of precompiling and then compiling. Do not specify parameters in the COMPILEOPT string that the SQL precompiler passes. Doing so will cause the compiler command to fail with a duplicate parameter error. It is possible that the SQL precompiler will pass additional parameters to the compiler in the future. This could lead to a duplicate parameter error, requiring your COMPILEOPT string to be changed at that time.

If ″INCDIR(″ is anywhere in the COMPILEOPT string, the precompiler will call the compiler using the SRCSTMF parameter.

```
EXEC SQL SET OPTION COMPILEOPT ='OPTION(*SHOWINC *EXPMAC)
     INCDIR(''/QSYS.LIB/MYLIB.LIB/MYFILE.MBR '')';
```

Interpreting compile errors in applications that use SQL

Sometimes you will encounter compile errors. Use the following information to interpret these errors.

Attention: If you separate precompile and compile steps, and the source program refers to externally described files, the referred to files must not be changed between precompile and compile. Otherwise, results that are not predictable may occur because the changes to the field definitions are not changed in the temporary source member.

Examples of externally described files are:
- COPY DDS in COBOL
- %INCLUDE in PL/I
- #pragma mapinc and #include in C or C++
- Externally-described files and externally-described data structures in RPG

When the SQL precompiler does not recognize host variables, try compiling the source. The compiler will not recognize the EXEC SQL statements, ignore these errors. Verify that the compiler interprets the host variable declaration as defined by the SQL precompiler for that language.

Error and warning messages during a compile of application programs that use SQL

These conditions might produce an error or warning message during an attempted compile process.

Related concepts

"Coding SQL statements in C and C++ applications" on page 13
To embed SQL statements in an ILE C or C++ program, you need to be aware of some unique application and coding requirements. This topic also defines the requirements for host structures and host variables.

"Coding SQL statements in COBOL applications" on page 41
There are unique application and coding requirements for embedding SQL statements in a COBOL program. In this topic, requirements for host structures and host variables are defined.

"Coding SQL statements in PL/I applications" on page 66
There are some unique application and coding requirements for embedding SQL statements in a PL/I program. In this topic, requirements for host structures and host variables are defined.

"Coding SQL statements in RPG/400 applications" on page 81
The RPG/400 licensed program supports both RPG II and RPG III programs.

"Coding SQL statements in ILE RPG applications" on page 91
You need to be aware of the unique application and coding requirements for embedding SQL statements in an ILE RPG program. In this topic, the coding requirements for host variables are defined.

Error and warning messages during a PL/I, C, or C++ compile:

If EXEC SQL starts before the left margin (as specified with the MARGINS parameter, the default), the SQL precompiler will not recognize the statement as an SQL statement. Consequently, it will be passed as is to the compiler.

Error and warning messages during a COBOL compile:

If EXEC SQL starts before column 12, the SQL precompiler will not recognize the statement as an SQL statement. Consequently, it will be passed as is to the compiler.

Error and warning messages during a RPG compile:

If EXEC SQL is not coded in positions 8 through 16, and preceded with the '/' character in position 7, the SQL precompiler will not recognize the statement as an SQL statement. Consequently, it will be passed as is to the compiler.

For more information, see the specific programming examples in the language sections.

Binding an application that uses SQL

Before you can run your application program, a relationship between the program and any specified tables and views must be established. This process is called *binding*. The result of binding is an *access plan*.

The access plan is a control structure that describes the actions necessary to satisfy each SQL request. An access plan contains information about the program and about the data the program intends to use.

For a nondistributed SQL program, the access plan is stored in the program. For a distributed SQL program (where the RDB parameter is specified on the CRTSQLxxx command), the access plan is stored in the SQL package at the specified relational database.

SQL automatically attempts to bind and create access plans when the program object is created. For non-ILE compiles, this occurs as the result of a successful CRTxxxPGM. For ILE compiles, this occurs as the result of a successful CRTBNDxxx, CRTPGM, or CRTSRVPGM command. If DB2 UDB for iSeries detects at run time that an access plan is not valid (for example, the referenced tables are in a different library) or detects that changes have occurred to the database that may improve performance (for example, the addition of indexes), a new access plan is automatically created. Binding does three things:

1. **It revalidates the SQL statements using the description in the database.** During the bind process, the SQL statements are checked for valid table, view, and column names. If a specified table or view does not exist at the time of the precompile or compile, the validation is done at run time. If the table or view does not exist at run time, a negative SQLCODE is returned.
2. **It selects the index needed to access the data your program wants to process.** In selecting an index, table sizes, and other factors are considered, when it builds an access plan. It considers all indexes available to access the data and decides which ones (if any) to use when selecting a path to the data.
3. **It attempts to build access plans.** If all the SQL statements are valid, the bind process then builds and stores access plans in the program.

If the characteristics of a table or view your program accesses have changed, the access plan may no longer be valid. When you attempt to run a program that contains an access plan that is not valid, the system automatically attempts to rebuild the access plan. If the access plan cannot be rebuilt, a negative SQLCODE is returned. In this case, you might have to change the program's SQL statements and reissue the CRTSQLxxx command to correct the situation.

Assume that a program contains an SQL statement that refers to COLUMNA in TABLEA and the user deletes and re-creates TABLEA so that COLUMNA no longer exists. When you call the program, the automatic rebind will be unsuccessful because COLUMNA no longer exists. In this case you must change the program source and reissue the CRTSQLxxx command.

Program references in applications that use SQL

All schemas, tables, views, SQL packages, and indexes referenced in SQL statements in an SQL program are placed in the object information repository (OIR) of the library when the program is created.

You can use the CL command Display Program References (DSPPGMREF) to display all object references in the program. If the SQL naming convention is used, the library name is stored in the OIR in one of three ways:

1. If the SQL name is fully qualified, the collection name is stored as the name qualifier.
2. If the SQL name is not fully qualified and the DFTRDBCOL parameter is not specified, the authorization ID of the statement is stored as the name qualifier.
3. If the SQL name is not fully qualified and the DFTRDBCOL parameter is specified, the schema name specified on the DFTRDBCOL parameter is stored as the name qualifier.

If the system naming convention is used, the library name is stored in the OIR in one of three ways:

1. If the object name is fully qualified, the library name is stored as the name qualifier.
2. If the object is not fully qualified and the DFTRDBCOL parameter is not specified, *LIBL is stored.
3. If the SQL name is not fully qualified and the DFTRDBCOL parameter is specified, the schema name specified on the DFTRDBCOL parameter is stored as the name qualifier.

Displaying SQL precompiler options

When the SQL application program is successfully compiled, the Display Module (DSPMOD), the Display Program (DSPPGM), or the Display Service Program (DSPSRVPGM) command can be used to determine some of the options that were specified on the SQL precompile.

This information may be needed when the source of the program has to be changed. These same SQL precompiler options can then be specified on the CRTSQLxxx command when the program is compiled again.

The Print SQL Information (PRTSQLINF) command can also be used to determine some of the options that were specified on the SQL precompile.

Running a program with embedded SQL

Running a host language program with embedded SQL statements, after the precompile and compile have been successfully done, is the same as running any host program.

Enter the following CALL statement:

```
CALL pgm-name
```

on the system command line.

Note: After installing a new release, users may encounter message CPF2218 in QHST using any Structured Query Language (SQL) program if the user does not have *CHANGE authority to the program. Once a user with *CHANGE authority calls the program, the access plan is updated and the message will be issued.

Related concepts

Control language

Running a program with embedded SQL: i5/OS DDM considerations

SQL does not support remote file access through distributed data management (DDM) files. SQL does support remote access through Distributed Relational Database Architecture™ (DRDA®).

Running a program with embedded SQL: Override considerations

You can use overrides (specified by the OVRDBF command) to direct a reference to a different table or view or to change certain operational characteristics of the program or SQL Package.

The following parameters are processed if an override is specified:

- TOFILE
- MBR
- SEQONLY
- INHWRT
- WAITRCD

All other override parameters are ignored. Overrides of statements in SQL packages are accomplished by doing both of the following:

1. Specifying the OVRSCOPE(*JOB) parameter on the OVRDBF command
2. Sending the command to the application server by using the Submit Remote Command (SBMRMTCMD) command

To override tables and views that are created with long names, you can create an override using the system name that is associated with the table or view. When the long name is specified in an SQL statement, the override is found using the corresponding system name.

An alias is actually created as a DDM file. You can create an override that refers to an alias name (DDM file). In this case, an SQL statement that refers to the file that has the override actually uses the file to which the alias refers.

Related concepts

Database programming

Database file management

Running a program with embedded SQL: SQL return codes

An SQL return code is sent by the database manager after the completion of each SQL statement.

Related concepts

SQL messages and codes

Example programs: Using DB2 UDB for iSeries statements

Here is a sample application that shows how to code SQL statements in each of the languages that DB2 UDB for iSeries supports.

The sample application gives raises based on commission.

Each sample program produces the same report, which is shown at the end of this topic. The first part of the report shows, by project, all employees working on the project who received a raise. The second part of the report shows the new salary expense for each project.

Notes about the sample programs

The following notes apply to all the sample programs:

SQL statements can be entered in uppercase or lowercase.

1 This host language statement retrieves the external definitions for the SQL table PROJECT. These definitions can be used as host variables or as a host structure.

Notes:

1. In RPG/400, field names in an externally described structure that are longer than 6 characters must be renamed.
2. REXX does not support the retrieval of external definitions.

2 The SQL INCLUDE SQLCA statement is used to include the SQLCA for PL/I, C, and COBOL programs. For RPG programs, the SQL precompiler automatically places the SQLCA data structure into the source at the end of the Input specification section. For REXX, the SQLCA fields are maintained in separate variables rather than in a contiguous data area mapped by the SQLCA.

3 This SQL WHENEVER statement defines the host language label to which control is passed if an SQLERROR (SQLCODE < 0) occurs in an SQL statement. This WHENEVER SQLERROR statement applies to all the following SQL statements until the next WHENEVER SQLERROR statement is encountered. REXX does not support the WHENEVER statement. Instead, REXX uses the SIGNAL ON ERROR facility.

4 This SQL UPDATE statement updates the *SALARY* column, which contains the employee salary by the percentage in the host variable PERCENTAGE (PERCNT for RPG). The updated rows are those that have employee commissions greater than 2000. For REXX, this is PREPARE and EXECUTE since UPDATE cannot be run directly if there is a host variable.

5 This SQL COMMIT statement commits the changes made by the SQL UPDATE statement. Record locks on all changed rows are released.

Note: The program was precompiled using COMMIT(*CHG). (For REXX, *CHG is the default.)

6 This SQL DECLARE CURSOR statement defines cursor C1, which joins two tables, EMPLOYEE and EMPPROJACT, and returns rows for employees who received a raise (commission > 2000). Rows are returned in ascending order by project number and employee number (PROJNO and EMPNO columns). For REXX, this is a PREPARE and DECLARE CURSOR since the DECLARE CURSOR statement cannot be specified directly with a statement string if it has host variables.

7 This SQL OPEN statement opens cursor C1 so that the rows can be fetched.

8 This SQL WHENEVER statement defines the host language label to which control is passed when all rows are fetched (SQLCODE = 100). For REXX, the SQLCODE must be explicitly checked.

9 This SQL FETCH statement returns all columns for cursor C1 and places the returned values into the corresponding elements of the host structure.

10 After all rows are fetched, control is passed to this label. The SQL CLOSE statement closes cursor C1.

11 This SQL DECLARE CURSOR statement defines cursor C2, which joins the three tables, EMPPROJACT, PROJECT, and EMPLOYEE. The results are grouped by columns PROJNO and PROJNAME. The COUNT function returns the number of rows in each group. The SUM function calculates the new salary cost for each project. The ORDER BY 1 clause specifies that rows are retrieved based on the contents of the final results column (EMPPROJACT.PROJNO). For REXX, this is a PREPARE and DECLARE CURSOR since the DECLARE CURSOR statement cannot be specified directly with a statement string if it has host variables.

12 This SQL FETCH statement returns the results columns for cursor C2 and places the returned values into the corresponding elements of the host structure described by the program.

13 This SQL WHENEVER statement with the CONTINUE option causes processing to continue to the next statement regardless if an error occurs on the SQL ROLLBACK statement. Errors are not expected on the SQL ROLLBACK statement; however, this prevents the program from going into a loop if an error does occur. SQL statements until the next WHENEVER SQLERROR statement is encountered. REXX does not support the WHENEVER statement. Instead, REXX uses the SIGNAL OFF ERROR facility.

14 This SQL ROLLBACK statement restores the table to its original condition if an error occurred during the update.

Related concepts

"Coding SQL statements in C and C++ applications" on page 13
To embed SQL statements in an ILE C or C++ program, you need to be aware of some unique application and coding requirements. This topic also defines the requirements for host structures and host variables.

"Coding SQL statements in COBOL applications" on page 41
There are unique application and coding requirements for embedding SQL statements in a COBOL program. In this topic, requirements for host structures and host variables are defined.

"Coding SQL statements in PL/I applications" on page 66
There are some unique application and coding requirements for embedding SQL statements in a PL/I program. In this topic, requirements for host structures and host variables are defined.

"Coding SQL statements in RPG/400 applications" on page 81
The RPG/400 licensed program supports both RPG II and RPG III programs.

Example: SQL statements in ILE C and C++ programs

This example program is written in the C programming language.

The same program would work in C++ if the following conditions are true:

- An SQL BEGIN DECLARE SECTION statement was added before line 18
- An SQL END DECLARE SECTION statement was added after line 42

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 176.

```
5722ST1 V5R4M0 060210          Create SQL ILE C Object          CEX                 08/06/02 15:52:26   Page   1
Source type...............C
Object name...............CORPDATA/CEX
Source file...............CORPDATA/SRC
Member....................CEX
To source file............QTEMP/QSQLTEMP
Options...................*XREF
Listing option............*PRINT
Target release............v5r4m0
INCLUDE file..............*SRCFILE
Commit....................*CHG
Allow copy of data........*YES
Close SQL cursor..........*ENDACTGRP
Allow blocking............*READ
Delay PREPARE.............*NO
Generation level..........10
Margins...................*SRCFILE
Printer file..............*LIBL/QSYSPRT
Date format...............*JOB
Date separator............*JOB
Time format...............*HMS
Time separator ...........*JOB
Replace...................*YES
Relational database.......*LOCAL
User .....................*CURRENT
RDB connect method........*DUW
Default collection........*NONE
Dynamic default
  collection..............*NO
Package name..............*OBJLIB/*OBJ
Path......................*NAMING
SQL rules.................*DB2
Created object type.......*PGM
Debugging view............*NONE
User profile..............*NAMING
Dynamic user profile......*USER
Sort Sequence.............*JOB
Language ID...............*JOB
IBM SQL flagging..........*NOFLAG
ANS flagging..............*NONE
Text......................*SRCMBRTXT
Source file CCSID.........65535
Job CCSID.................65535
Decimal result options:
  Maximum precision.......31
  Maximum scale...........31
  Minimum divide scale....0
Compiler options..........*NONE
Source member changed on 06/06/00  17:15:17
```

Figure 3. Sample C program using SQL statements

```
5722ST1 V5R4M0 060210     Create SQL ILE C Object          CEX                    08/06/02 15:52:26   Page   2
Record  *...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8   SEQNBR  Last change
     1   #include "string.h"                                                                  100
     2   #include "stdlib.h"                                                                  200
     3   #include "stdio.h"                                                                   300
     4                                                                                        400
     5   main()                                                                               500
     6   {                                                                                    600
     7   /* A sample program which updates the salaries for those employees   */              700
     8   /* whose current commission total is greater than or equal to the    */              800
     9   /* value of 'commission'. The salaries of those who qualify are      */              900
    10   /* increased by the value of 'percentage' retroactive to 'raise_date'*/             1000
    11   /* A report is generated showing the projects which these employees  */             1100
    12   /* have contributed to ordered by project number and employee ID.    */             1200
    13   /* A second report shows each project having an end date occurring   */             1300
    14   /* after 'raise_date' (is potentially affected by the retroactive    */             1400
    15   /* raises) with its total salary expenses and a count of employees   */             1500
    16   /* who contributed to the project.                                   */             1600
    17                                                                                       1700
    18      short work_days  = 253;          /* work days during in one year */             1800
    19      float commission = 2000.00;      /* cutoff to qualify for raise  */             1900
    20      float percentage = 1.04;         /* raised salary as percentage  */             2000
    21      char raise_date??(12??) = "1982-06-01"; /*  effective raise date */             2100
    22                                                                                       2200
    23      /* File declaration for qprint */                                                2300
    24      FILE *qprint;                                                                    2400
    25                                                                                       2500
    26      /* Structure for report 1 */                                                     2600
    27     1 #pragma mapinc ("project","CORPDATA/PROJECT(PROJECT)","both","p z")             2700
    28      #include "project"                                                               2800
    29      struct {                                                                         2900
    30              CORPDATA_PROJECT_PROJECT_both_t Proj_struct;                             3000
    31              char  empno??(7??);                                                      3100
    32              char  name??(30??);                                                      3200
    33              float salary;                                                            3300
    34              } rpt1;                                                                  3400
    35                                                                                       3500
    36      /* Structure for report 2 */                                                     3600
    37      struct {                                                                         3700
    38              char projno??(7??);                                                      3800
    39              char project_name??(37??);                                               3900
    40              short employee_count;                                                    4000
    41              double total_proj_cost;                                                  4100
    42             } rpt2;                                                                   4200
    43                                                                                       4300
    44     2 exec sql include SQLCA;                                                         4400
    45                                                                                       4500
    46      qprint=fopen("QPRINT","w");                                                      4600
    47                                                                                       4700
    48      /* Update the selected projects by the new percentage. If an error */           4800
    49      /* occurs during the update, ROLLBACK the changes.                 */           4900
    50     3 EXEC SQL WHENEVER SQLERROR GO TO update_error;                                  5000
    51     4 EXEC SQL                                                                        5100
    52           UPDATE CORPDATA/EMPLOYEE                                                    5200
    53              SET SALARY = SALARY * :percentage                                        5300
    54              WHERE COMM >= :commission ;                                              5400
    55                                                                                       5500
    56      /* Commit changes */                                                             5600
    57     5 EXEC SQL                                                                        5700
    58           COMMIT;                                                                     5800
    59      EXEC SQL WHENEVER SQLERROR GO TO report_error;                                   5900
    60                                                                                       6000
```

```
5722ST1 V5R4M0 060210     Create SQL ILE C Object          CEX                    08/06/02 15:52:26   Page   3
Record  *...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8   SEQNBR  Last change
   61      /* Report the updated statistics for each employee assigned to the */             6100
   62      /* selected projects.                                              */             6200
   63                                                                                        6300
   64      /* Write out the header for Report 1 */                                           6400
   65      fprintf(qprint,"                     REPORT OF PROJECTS AFFECTED \                6500
   66   BY RAISES");                                                                         6600
   67      fprintf(qprint,"\n\nPROJECT  EMPID     EMPLOYEE NAME    ");                       6700
   68      fprintf(qprint,  "                  SALARY\n");                                   6800
   69                                                                                        6900
   70     6 exec sql                                                                         7000
   71           declare c1 cursor for                                                        7100
   72             select distinct projno, empprojact.empno,                                  7200
   73                    lastname||', '||firstnme, salary                                    7300
   74             from corpdata/empprojact, corpdata/employee                                7400
   75             where empprojact.empno = employee.empno and comm >= :commission            7500
   76             order by projno, empno;                                                    7600
   77     7 EXEC SQL                                                                         7700
   78           OPEN C1;                                                                     7800
   79                                                                                        7900
   80      /* Fetch and write the rows to QPRINT */                                          8000
   81     8 EXEC SQL WHENEVER NOT FOUND GO TO done1;                                         8100
   82                                                                                        8200
   83      do {                                                                              8300
   84     10 EXEC SQL                                                                        8400
   85             FETCH C1 INTO :Proj_struct.PROJNO, :rpt1.empno,                            8500
   86                           :rpt1.name,:rpt1.salary;                                     8600
   87        fprintf(qprint,"\n%6s   %6s    %-30s    %8.2f",                                 8700
   88                rpt1.Proj_struct.PROJNO,rpt1.empno,                                     8800
   89                rpt1.name,rpt1.salary);                                                 8900
   90        }                                                                               9000
   91      while (SQLCODE==0);                                                               9100
   92                                                                                        9200
   93    done1:                                                                              9300
   94      EXEC SQL                                                                          9400
   95           CLOSE C1;                                                                    9500
   96                                                                                        9600
   97      /* For all projects ending at a date later than the 'raise_date'   * /            9700
   98      /* (i.e. those projects potentially affected by the salary raises) */             9800
   99      /* generate a report containing the project number, project name   */             9900
  100      /* the count of employees participating in the project and the     */            10000
  101      /* total salary cost of the project.                               */            10100
  102                                                                                       10200
  103      /* Write out the header for Report 2 */                                          10300
  104      fprintf(qprint,"\n\n\n                     ACCUMULATED STATISTICS\               10400
  105    BY PROJECT");                                                                      10500
  106      fprintf(qprint,  "\n\nPROJECT                                    \               10600
  107       NUMBER OF       TOTAL");                                                        10700
  108      fprintf(qprint,    "\nNUMBER   PROJECT NAME                       \               10800
  109       EMPLOYEES       COST\n");                                                       10900
  110                                                                                       11000
  111   11 EXEC SQL                                                                         11100
  112           DECLARE C2 CURSOR FOR                                                       11200
  113             SELECT EMPPROJACT.PROJNO, PROJNAME, COUNT(*),                             11300
  114                SUM ( ( DAYS(EMENDATE) - DAYS(EMSTDATE) ) * EMPTIME *                  11400
  115                    (DECIMAL( SALARY / :work_days ,8,2)))                              11500
  116             FROM CORPDATA/EMPPROJACT, CORPDATA/PROJECT, CORPDATA/EMPLOYEE             11600
  117             WHERE EMPPROJACT.PROJNO=PROJECT.PROJNO  AND                               11700
  118                   EMPPROJACT.EMPNO =EMPLOYEE.EMPNO  AND                               11800
  119                   PRENDATE > :raise_date                                              11900
  120             GROUP BY EMPPROJACT.PROJNO, PROJNAME                                      12000
  121             ORDER BY 1;                                                               12100
  122      EXEC SQL                                                                         12200
  123           OPEN C2;                                                                    12300
```

```
5722ST1 V5R4M0 060210            Create SQL ILE C Object          CEX              08/06/02 15:52:26   Page   4
Record  *...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8   SEQNBR  Last change
  124                                                                                        12400
  125      /* Fetch and write the rows to QPRINT */                                          12500
  126      EXEC SQL WHENEVER NOT FOUND GO TO done2;                                          12600
  127                                                                                        12700
  128      do {                                                                              12800
  129     12 EXEC SQL                                                                        12900
  130             FETCH C2 INTO :rpt2;                                                       13000
  131        fprintf(qprint,"\n%6s   %-36s  %6d       %9.2f",                                  13100
  132                rpt2.projno,rpt2.project_name,rpt2.employee_count,                          13200
  133                rpt2.total_proj_cost);                                                      13300
  134      }                                                                                 13400
  135      while (SQLCODE==0);                                                               13500
  136                                                                                        13600
  137    done2:                                                                              13700
  138      EXEC SQL                                                                          13800
  139           CLOSE C2;                                                                    13900
  140      goto finished;                                                                    14000
  141                                                                                        14100
  142      /* Error occurred while updating table.  Inform user and rollback   */             14200
  143      /* changes.                                                        */             14300
  144    update_error:                                                                       14400
  145     13 EXEC SQL WHENEVER SQLERROR CONTINUE;                                            14500
  146      fprintf(qprint,"*** ERROR Occurred while updating table.  SQLCODE="               14600
  147              "%5d\n",SQLCODE);                                                         14700
  148     14 EXEC SQL                                                                        14800
  149           ROLLBACK;                                                                    14900
  150      goto finished;                                                                    15000
  151                                                                                        15100
  152      /* Error occurred while generating reports.  Inform user and exit.  */             15200
  153    report_error:                                                                       15300
  154      fprintf(qprint,"*** ERROR Occurred while generating reports.  "                   15400
  155              "SQLCODE=%5d\n",SQLCODE);                                                 15500
  156      goto finished;                                                                    15600
  157                                                                                        15700
  158      /* All done  */                                                                   15800
  159    finished:                                                                           15900
  160      fclose(qprint);                                                                   16000
  161      exit(0);                                                                          16100
  162                                                                                        16200
  163   }                                                                                    16300
* * * * *  E N D  O F  S O U R C E  * * * * *
```

```
5722ST1 V5R4M0 060210           Create SQL ILE C Object          CEX                  08/06/02 15:52:26   Page   5

CROSS REFERENCE
Data Names                    Define    Reference
commission                       19       FLOAT(24)
                                          54 75
done1                            ****     LABEL
                                          81
done2                            ****     LABEL
                                          126
employee_count                   40       SMALL INTEGER PRECISION(4,0) IN rpt2
empno                            31       VARCHAR(7) IN rpt1
                                          85
name                             32       VARCHAR(30) IN rpt1
                                          86
percentage                       20       FLOAT(24)
                                          53
project_name                     39       VARCHAR(37) IN rpt2
projno                           38       VARCHAR(7) IN rpt2
raise_date                       21       VARCHAR(12)
                                          119
report_error                     ****     LABEL
                                          59
rpt1                             34
rpt2                             42       STRUCTURE
                                          130
salary                           33       FLOAT(24) IN rpt1
                                          86
total_proj_cost                  41       FLOAT(53) IN rpt2
update_error                     ****     LABEL
                                          50
work_days                        18       SMALL INTEGER PRECISION(4,0)
                                          115
ACTNO                            74       SMALL INTEGER PRECISION(4,0) COLUMN (NOT NULL) IN CORPDATA.EMPPROJACT
BIRTHDATE                        74       DATE(10) COLUMN IN CORPDATA.EMPLOYEE
BONUS                            74       DECIMAL(9,2) COLUMN IN CORPDATA.EMPLOYEE
COMM                             ****     COLUMN
                                          54 75
COMM                             74       DECIMAL(9,2) COLUMN IN CORPDATA.EMPLOYEE
CORPDATA                         ****     COLLECTION
                                          52 74 74 116 116 116
C1                               71       CURSOR
                                          78 85 95
C2                               112      CURSOR
                                          123 130 139
DEPTNO                           27       VARCHAR(3) IN Proj_struct
DEPTNO                           116      CHARACTER(3) COLUMN (NOT NULL) IN CORPDATA.PROJECT
EDLEVEL                          74       SMALL INTEGER PRECISION(4,0) COLUMN (NOT NULL) IN CORPDATA.EMPLOYEE
EMENDATE                         74       DATE(10) COLUMN IN CORPDATA.EMPPROJACT
EMENDATE                         ****     COLUMN
                                          114
EMPLOYEE                         ****     TABLE IN CORPDATA
                                          52 74 116
EMPLOYEE                         ****     TABLE
                                          75 118
EMPNO                            ****     COLUMN IN EMPPROJACT
                                          72 75 76 118
EMPNO                            ****     COLUMN IN EMPLOYEE
                                          75 118
EMPNO                            74       CHARACTER(6) COLUMN (NOT NULL) IN CORPDATA.EMPPROJACT
EMPNO                            74       CHARACTER(6) COLUMN (NOT NULL) IN CORPDATA.EMPLOYEE
EMPPROJACT                       ****     TABLE
                                          72 75 113 117 118 120
EMPPROJACT                       ****     TABLE IN CORPDATA
                                          74 116
```

```
5722ST1 V5R4M0 060210         Create SQL ILE C Object         CEX                  08/06/02 15:52:26   Page   6
CROSS REFERENCE

EMPTIME                         74       DECIMAL(5,2) COLUMN IN CORPDATA.EMPPROJACT
EMPTIME                         ****     COLUMN
                                         114
EMSTDATE                        74       DATE(10) COLUMN IN CORPDATA.EMPPROJACT
EMSTDATE                        ****     COLUMN
                                         114
FIRSTNME                        ****     COLUMN
                                         73
FIRSTNME                        74       VARCHAR(12) COLUMN (NOT NULL) IN CORPDATA.EMPLOYEE
HIREDATE                        74       DATE(10) COLUMN IN CORPDATA.EMPLOYEE
JOB                             74       CHARACTER(8) COLUMN IN CORPDATA.EMPLOYEE
LASTNAME                        ****     COLUMN
                                73
LASTNAME                        74       VARCHAR(15) COLUMN (NOT NULL) IN CORPDATA.EMPLOYEE
MAJPROJ                         27       VARCHAR(6) IN Proj_struct
MAJPROJ                         116      CHARACTER(6) COLUMN IN CORPDATA.PROJECT
MIDINIT                         74       CHARACTER(1) COLUMN (NOT NULL) IN CORPDATA.EMPLOYEE
Proj_struct                     30       STRUCTURE IN rpt1
PHONENO                         74       CHARACTER(4) COLUMN IN CORPDATA.EMPLOYEE
PRENDATE                        27       DATE(10) IN Proj_struct
PRENDATE                        ****     COLUMN
                                         119
PRENDATE                        116      DATE(10) COLUMN IN CORPDATA.PROJECT
PROJECT                         ****     TABLE IN CORPDATA
                                         116
PROJECT                         ****     TABLE
                                         117
PROJNAME                        27       VARCHAR(24) IN Proj_struct
PROJNAME                        ****     COLUMN
                                         113 120
PROJNAME                        116      VARCHAR(24) COLUMN (NOT NULL) IN CORPDATA.PROJECT
PROJNO                          27       VARCHAR(6) IN Proj_struct
                                         85
PROJNO                          ****     COLUMN
                                         72 76
PROJNO                          74       CHARACTER(6) COLUMN (NOT NULL) IN CORPDATA.EMPPROJACT
PROJNO                          ****     COLUMN IN EMPPROJACT
                                         113 117 120
PROJNO                          ****     COLUMN IN PROJECT
                                         117
PROJNO                          116      CHARACTER(6) COLUMN (NOT NULL) IN CORPDATA.PROJECT
PRSTAFF                         27       DECIMAL(5,2) IN Proj_struct
PRSTAFF                         116      DECIMAL(5,2) COLUMN IN CORPDATA.PROJECT
PRSTDATE                        27       DATE(10) IN Proj_struct
PRSTDATE                        116      DATE(10) COLUMN IN CORPDATA.PROJECT
RESPEMP                         27       VARCHAR(6) IN Proj_struct
RESPEMP                         116      CHARACTER(6) COLUMN (NOT NULL) IN CORPDATA.PROJECT
SALARY                          ****     COLUMN
                                         53 53 73 115
SALARY                          74       DECIMAL(9,2) COLUMN IN CORPDATA.EMPLOYEE
SEX                             74       CHARACTER(1) COLUMN IN CORPDATA.EMPLOYEE
WORKDEPT                        74       CHARACTER(3) COLUMN IN CORPDATA.EMPLOYEE
No errors found in source
163 Source records processed
* * * * *  E N D  O F  L I S T I N G  * * * * *
```

Example: SQL statements in COBOL and ILE COBOL programs

This example program is written in the COBOL programming language.

Note: By using the code examples, you agree to the terms of the “Code license and disclaimer information” on page 176.

```
5722ST1 V5R4M0 060210      Create SQL COBOL Program          CBLEX             08/06/02 11:09:13   Page   1
Source type...............COBOL
Program name..............CORPDATA/CBLEX
Source file...............CORPDATA/SRC
Member....................CBLEX
To source file............QTEMP/QSQLTEMP
Options...................*SRC      *XREF
Target release............v5r4m0
INCLUDE file..............*SRCFILE
Commit....................*CHG
Allow copy of data........*YES
Close SQL cursor..........*ENDPGM
Allow blocking............*READ
Delay PREPARE.............*NO
Generation level..........10
Printer file..............*LIBL/QSYSPRT
Date format...............*JOB
Date separator............*JOB
Time format...............*HMS
Time separator ...........*JOB
Replace...................*YES
Relational database.......*LOCAL
User .....................*CURRENT
RDB connect method........*DUW
Default collection........*NONE
Dynamic default
  collection..............*NO
Package name..............*PGMLIB/*PGM
Path......................*NAMING
Created object type.......*PGM
SQL rules.................*DB2
User profile..............*NAMING
Dynamic user profile......*USER
Sort Sequence.............*JOB
Language ID...............*JOB
IBM SQL flagging..........*NOFLAG
ANS flagging..............*NONE
Text......................*SRCMBRTXT
Source file CCSID.........65535
Job CCSID.................65535
Decimal result options:
  Maximum precision.......31
  Maximum scale...........31
  Minimum divide scale....0
Compiler options..........*NONE
Source member changed on 07/01/96  09:44:58
```

Figure 4. Sample COBOL program using SQL statements

```
5722ST1 V5R4M0 060210     Create SQL COBOL Program          CBLEX           08/06/02 11:09:13         Page   2
Record  *...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8   SEQNBR  Last change
     1
     2        ****************************************************************
     3        * A sample program which updates the salaries for those        *
     4        * employees whose current commission total is greater than or  *
     5        * equal to the value of COMMISSION. The salaries of those who  *
     6        * qualify are increased by the value of PERCENTAGE retroactive *
     7        * to RAISE-DATE. A report is generated showing the projects    *
     8        * which these employees have contributed to ordered by the     *
     9        * project number and employee ID. A second report shows each   *
    10        * project having an end date occurring  after RAISE-DATE       *
    11        * (i.e. potentially affected by the retroactive raises ) with  *
    12        * its total salary expenses and a count of employees who       *
    13        * contributed to the project.                                  *
    14        ****************************************************************
    15
    16
    17         IDENTIFICATION DIVISION.
    18
    19         PROGRAM-ID.  CBLEX.
    20         ENVIRONMENT DIVISION.
    21         CONFIGURATION SECTION.
    22         SOURCE-COMPUTER. IBM-AS400.
    23         OBJECT-COMPUTER. IBM-AS400.
    24         INPUT-OUTPUT SECTION.
    25
    26         FILE-CONTROL.
    27             SELECT PRINTFILE ASSIGN TO PRINTER-QPRINT
    28                ORGANIZATION IS SEQUENTIAL.
    29
    30         DATA DIVISION.
    31
    32         FILE SECTION.
    33
    34         FD  PRINTFILE
    35             BLOCK CONTAINS 1 RECORDS
    36             LABEL RECORDS ARE OMITTED.
    37         01  PRINT-RECORD PIC X(132).
    38
    39         WORKING-STORAGE SECTION.
    40         77  WORK-DAYS PIC S9(4) BINARY VALUE 253.
    41         77  RAISE-DATE PIC X(11) VALUE "1982-06-01".
    42         77  PERCENTAGE PIC S999V99 PACKED-DECIMAL.
    43         77  COMMISSION PIC S99999V99 PACKED-DECIMAL VALUE 2000.00.
    44
    45        ***************************************************************
    46        *  Structure for report 1.                                    *
    47        ***************************************************************
    48
    49      1  01  RPT1.
    50             COPY DDS-PROJECT OF CORPDATA-PROJECT.
    51             05  EMPNO     PIC X(6).
    52             05  NAME      PIC X(30).
    53             05  SALARY    PIC S9(6)V99 PACKED-DECIMAL.
    54
    55
```

```
5722ST1 V5R4M0 060210     Create SQL COBOL Program          CBLEX              08/06/02 11:09:13         Page   3
Record  *...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8   SEQNBR  Last change
    56        ****************************************************************
    57        *  Structure for report 2.                                     *
    58        ****************************************************************
    59
    60         01  RPT2.
    61             15  PROJNO PIC X(6).
    62             15  PROJECT-NAME PIC X(36).
    63             15  EMPLOYEE-COUNT PIC S9(4) BINARY.
    64             15  TOTAL-PROJ-COST PIC S9(10)V99 PACKED-DECIMAL.
    65
    66          2 EXEC SQL
    67                  INCLUDE SQLCA
    68             END-EXEC.
    69         77  CODE-EDIT PIC ---99.
    70
    71        ****************************************************************
    72        *  Headers for reports.                                        *
    73        ****************************************************************
    74
    75         01  RPT1-HEADERS.
    76             05  RPT1-HEADER1.
    77                 10  FILLER PIC X(21) VALUE SPACES.
    78                 10  FILLER PIC X(111)
    79                       VALUE "REPORT OF PROJECTS AFFECTED BY RAISES".
    80             05  RPT1-HEADER2.
    81                 10  FILLER PIC X(9) VALUE "PROJECT".
    82                 10  FILLER PIC X(10) VALUE "EMPID".
    83                 10  FILLER PIC X(35) VALUE "EMPLOYEE NAME".
    84                 10  FILLER PIC X(40) VALUE "SALARY".
    85         01  RPT2-HEADERS.
    86             05  RPT2-HEADER1.
    87                 10  FILLER PIC X(21) VALUE SPACES.
    88                 10  FILLER PIC X(111)
    89                         VALUE "ACCUMULATED STATISTICS BY PROJECT".
    90             05  RPT2-HEADER2.
    91                 10  FILLER PIC X(9) VALUE "PROJECT".
    92                 10  FILLER PIC X(38) VALUE SPACES.
    93                 10  FILLER PIC X(16) VALUE "NUMBER OF".
    94                 10  FILLER PIC X(10) VALUE "TOTAL".
    95             05  RPT2-HEADER3.
    96                 10  FILLER PIC X(9) VALUE "NUMBER".
    97                 10  FILLER PIC X(38) VALUE "PROJECT NAME".
    98                 10  FILLER PIC X(16) VALUE "EMPLOYEES".
    99                 10  FILLER PIC X(65) VALUE "COST".
   100         01  RPT1-DATA.
   101             05  PROJNO    PIC X(6).
   102             05  FILLER    PIC XXX VALUE SPACES.
   103             05  EMPNO     PIC X(6).
   104             05  FILLER    PIC X(4) VALUE SPACES.
   105             05  NAME      PIC X(30).
   106             05  FILLER    PIC X(3) VALUE SPACES.
   107             05  SALARY    PIC ZZZZZ9.99.
   108             05  FILLER    PIC X(96) VALUE SPACES.
   109         01  RPT2-DATA.
   110             05  PROJNO PIC X(6).
   111             05  FILLER PIC XXX VALUE SPACES.
   112             05  PROJECT-NAME PIC X(36).
   113             05  FILLER PIC X(4) VALUE SPACES.
   114             05  EMPLOYEE-COUNT PIC ZZZ9.
   115             05  FILLER PIC X(5) VALUE SPACES.
   116             05  TOTAL-PROJ-COST PIC ZZZZZZZZ9.99.
   117             05  FILLER PIC X(56) VALUE SPACES.
   118
```

```
5722ST1 V5R4M0 060210       Create SQL COBOL Program        CBLEX      08/06/02 11:09:13        Page  4
Record  *...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8   SEQNBR  Last change
  119         PROCEDURE DIVISION.
  120
  121         A000-MAIN.
  122             MOVE 1.04 TO PERCENTAGE.
  123             OPEN OUTPUT PRINTFILE.
  124
  125        ***************************************************************
  126        * Update the selected employees by the new percentage. If an  *
  127        * error occurs during the update, ROLLBACK the changes,       *
  128        ***************************************************************
  129
  130          3 EXEC SQL
  131                  WHENEVER SQLERROR GO TO E010-UPDATE-ERROR
  132             END-EXEC.
  133          4 EXEC SQL
  134                  UPDATE CORPDATA/EMPLOYEE
  135                    SET SALARY = SALARY * :PERCENTAGE
  136                    WHERE COMM >= :COMMISSION
  137             END-EXEC.
  138
  139        ***************************************************************
  140        *  Commit changes.                                            *
  141        ***************************************************************
  142
  143          5 EXEC SQL
  144                  COMMIT
  145             END-EXEC.
  146
  147             EXEC SQL
  148                  WHENEVER SQLERROR GO TO E020-REPORT-ERROR
  149             END-EXEC.
  150
  151        ***************************************************************
  152        *  Report the updated statistics for each employee receiving  *
  153        *  a raise and the projects that s/he participates in         *
  154        ***************************************************************
  155
  156        ***************************************************************
  157        *  Write out the header for Report 1.                         *
  158        ***************************************************************
  159
  160             write print-record from rpt1-header1
  161                   before advancing 2 lines.
  162             write print-record from rpt1-header2
  163                   before advancing 1 line.
  164          6 exec sql
  165                  declare c1 cursor for
  166                    SELECT DISTINCT projno, empprojact.empno,
  167                            lastname||", "||firstnme ,salary
  168                    from corpdata/empprojact, corpdata/employee
  169                    where empprojact.empno =employee.empno and
  170                          comm >= :commission
  171                    order by projno, empno
  172             end-exec.
  173          7 EXEC SQL
  174                  OPEN C1
  175             END-EXEC.
  176
  177             PERFORM B000-GENERATE-REPORT1 THRU B010-GENERATE-REPORT1-EXIT
  178                 UNTIL SQLCODE NOT EQUAL TO ZERO.
  179
```

Note: 8 and 9 are located on Part 5 of this figure.

```
5722ST1 V5R4M0 060210       Create SQL COBOL Program          CBLEX            08/06/02 11:09:13   Page   5
Record  *...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8   SEQNBR  Last change
  180     10 A100-DONE1.
  181              EXEC SQL
  182                   CLOSE C1
  183              END-EXEC.
  184
  185         **************************************************************
  186         *  For all projects ending at a date later than the RAISE-   *
  187         *  DATE ( i.e. those projects potentially affected by the    *
  188         *  salary raises generate a report containing the project    *
  189         *  project number, project name, the count of employees      *
  190         *  participating in the project and the total salary cost    *
  191         *  for the project                                           *
  192         **************************************************************
  193
  194
  195         ****************************************************************
  196         *  Write out the header for Report 2.                          *
  197         ****************************************************************
  198
  199              MOVE SPACES TO PRINT-RECORD.
  200              WRITE PRINT-RECORD BEFORE ADVANCING 2 LINES.
  201              WRITE PRINT-RECORD FROM RPT2-HEADER1
  202                    BEFORE ADVANCING 2 LINES.
  203              WRITE PRINT-RECORD FROM RPT2-HEADER2
  204                    BEFORE ADVANCING 1 LINE.
  205              WRITE PRINT-RECORD FROM RPT2-HEADER3
  206                    BEFORE ADVANCING 2 LINES.
  207
  208              EXEC SQL
  209               11 DECLARE C2 CURSOR FOR
  210                     SELECT EMPPROJACT.PROJNO, PROJNAME, COUNT(*),
  211                            SUM ( (DAYS(EMENDATE)-DAYS(EMSTDATE)) *
  212                            EMPTIME * DECIMAL((SALARY / :WORK-DAYS),8,2))
  213                     FROM CORPDATA/EMPPROJACT, CORPDATA/PROJECT,
  214                          CORPDATA/EMPLOYEE
  215                     WHERE EMPPROJACT.PROJNO=PROJECT.PROJNO AND
  216                           EMPPROJACT.EMPNO =EMPLOYEE.EMPNO AND
  217                           PRENDATE > :RAISE-DATE
  218                     GROUP BY EMPPROJACT.PROJNO, PROJNAME
  219                     ORDER BY 1
  220              END-EXEC.
  221              EXEC SQL
  222                   OPEN C2
  223              END-EXEC.
  224
  225              PERFORM C000-GENERATE-REPORT2 THRU C010-GENERATE-REPORT2-EXIT
  226                   UNTIL SQLCODE NOT EQUAL TO ZERO.
  227
  228          A200-DONE2.
  229              EXEC SQL
  230                   CLOSE C2
  231              END-EXEC
  232
  233         ****************************************************************
  234         *  All done.                                                   *
  235         ****************************************************************
  236
  237          A900-MAIN-EXIT.
  238              CLOSE PRINTFILE.
  239              STOP RUN.
  240
```

```
5722ST1 V5R4M0 060210         Create SQL COBOL Program          CBLEX               08/06/02 11:09:13   Page   6
Record  *...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8   SEQNBR  Last change
  241         ***************************************************************
  242         *  Fetch and write the rows to PRINTFILE.                     *
  243         ***************************************************************
  244
  245          B000-GENERATE-REPORT1.
  246           8 EXEC SQL
  247                  WHENEVER NOT FOUND GO TO A100-DONE1
  248              END-EXEC.
  249           9 EXEC SQL
  250                  FETCH C1 INTO :PROJECT.PROJNO, :RPT1.EMPNO,
  251                                :RPT1.NAME, :RPT1.SALARY
  252              END-EXEC.
  253              MOVE CORRESPONDING RPT1 TO RPT1-DATA.
  254              MOVE PROJNO OF RPT1 TO PROJNO OF RPT1-DATA.
  255              WRITE PRINT-RECORD FROM RPT1-DATA
  256                    BEFORE ADVANCING 1 LINE.
  257
  258          B010-GENERATE-REPORT1-EXIT.
  259              EXIT.
  260
  261         ***************************************************************
  262         *  Fetch and write the rows to PRINTFILE.                     *
  263         ***************************************************************
  264
  265          C000-GENERATE-REPORT2.
  266              EXEC SQL
  267                  WHENEVER NOT FOUND GO TO A200-DONE2
  268              END-EXEC.
  269          12 EXEC SQL
  270                  FETCH C2 INTO :RPT2
  271              END-EXEC.
  272              MOVE CORRESPONDING RPT2 TO RPT2-DATA.
  273              WRITE PRINT-RECORD FROM RPT2-DATA
  274                    BEFORE ADVANCING 1 LINE.
  275
  276          C010-GENERATE-REPORT2-EXIT.
  277              EXIT.
  278
  279         ***************************************************************
  280         *  Error occurred while updating table.  Inform user and       *
  281         *  rollback changes.                                          *
  282         ***************************************************************
  283
  284          E010-UPDATE-ERROR.
  285          13 EXEC SQL
  286                  WHENEVER SQLERROR CONTINUE
  287              END-EXEC.
  288              MOVE SQLCODE TO CODE-EDIT.
  289              STRING "*** ERROR Occurred while updating table.  SQLCODE="
  290                    CODE-EDIT DELIMITED BY SIZE INTO PRINT-RECORD.
  291              WRITE PRINT-RECORD.
  292          14 EXEC SQL
  293                  ROLLBACK
  294              END-EXEC.
  295              STOP RUN.
  296
  297         ***************************************************************
  298         *  Error occurred while generating reports.  Inform user and   *
  299         *  exit.                                                      *
  300         ***************************************************************
  301
  302          E020-REPORT-ERROR.
  303              MOVE SQLCODE TO CODE-EDIT.
  304              STRING "*** ERROR Occurred while generating reports.  SQLCODE
  305         -           "=" CODE-EDIT DELIMITED BY SIZE INTO PRINT-RECORD.
  306              WRITE PRINT-RECORD.
  307              STOP RUN.
                                * * * * *  E N D  O F  S O U R C E  * * * * *
```

```
5722ST1 V5R4M0 060210        Create SQL COBOL Program          CBLEX                 08/06/02 11:09:13   Page   7
CROSS REFERENCE
Data Names                    Define    Reference
ACTNO                            168      SMALL INTEGER PRECISION(4,0) COLUMN (NOT NULL) IN CORPDATA.EMPPROJACT
A100-DONE1                       ****     LABEL
                                          247
A200-DONE2                       ****     LABEL
                                          267
BIRTHDATE                        134      DATE(10) COLUMN IN CORPDATA.EMPLOYEE
BONUS                            134      DECIMAL(9,2) COLUMN IN CORPDATA.EMPLOYEE
CODE-EDIT                        69
COMM                             ****     COLUMN
                                          136 170
COMM                             134      DECIMAL(9,2) COLUMN IN CORPDATA.EMPLOYEE
COMMISSION                       43       DECIMAL(7,2)
                                          136 170
CORPDATA                         ****     COLLECTION
                                          134 168 168 213 213 214
C1                               165      CURSOR
                                          174 182 250
C2                               209      CURSOR
                                          222 230 270
DEPTNO                           50       CHARACTER(3) IN PROJECT
DEPTNO                           213      CHARACTER(3) COLUMN (NOT NULL) IN CORPDATA.PROJECT
EDLEVEL                          134      SMALL INTEGER PRECISION(4,0) COLUMN (NOT NULL) IN CORPDATA.EMPLOYEE
EMENDATE                         168      DATE(10) COLUMN IN CORPDATA.EMPPROJACT
EMENDATE                         ****     COLUMN
                                          211
EMPLOYEE                         ****     TABLE IN CORPDATA
                                          134 168 214
EMPLOYEE                         ****     TABLE
                                          169 216
EMPLOYEE-COUNT                   63       SMALL INTEGER PRECISION(4,0) IN RPT2
EMPLOYEE-COUNT                   114      IN RPT2-DATA
EMPNO                            51       CHARACTER(6) IN RPT1
                                          250
EMPNO                            103      CHARACTER(6) IN RPT1-DATA
EMPNO                            134      CHARACTER(6) COLUMN (NOT NULL) IN CORPDATA.EMPLOYEE
EMPNO                            ****     COLUMN IN EMPPROJACT
                                          166 169 171 216
EMPNO                            ****     COLUMN IN EMPLOYEE
                                          169 216
EMPNO                            168      CHARACTER(6) COLUMN (NOT NULL) IN CORPDATA.EMPPROJACT
EMPPROJACT                       ****     TABLE
                                          166 169 210 215 216 218
EMPPROJACT                       ****     TABLE IN CORPDATA
                                          168 213
EMPTIME                          168      DECIMAL(5,2) COLUMN IN CORPDATA.EMPPROJACT
EMPTIME                          ****     COLUMN
                                          212
EMSTDATE                         168      DATE(10) COLUMN IN CORPDATA.EMPPROJACT
EMSTDATE                         ****     COLUMN
                                          211
E010-UPDATE-ERROR                ****     LABEL
                                          131
E020-REPORT-ERROR                ****     LABEL
                                          148
FIRSTNME                         134      VARCHAR(12) COLUMN (NOT NULL) IN CORPDATA.EMPLOYEE
FIRSTNME                         ****     COLUMN
                                          167
HIREDATE                         134      DATE(10) COLUMN IN CORPDATA.EMPLOYEE
JOB                              134      CHARACTER(8) COLUMN IN CORPDATA.EMPLOYEE
LASTNAME                         134      VARCHAR(15) COLUMN (NOT NULL) IN CORPDATA.EMPLOYEE
LASTNAME                         ****     COLUMN
                                          167
MAJPROJ                          50       CHARACTER(6) IN PROJECT
MAJPROJ                          213      CHARACTER(6) COLUMN IN CORPDATA.PROJECT
MIDINIT                          134      CHARACTER(1) COLUMN (NOT NULL) IN CORPDATA.EMPLOYEE
NAME                             52       CHARACTER(30) IN RPT1
                                          251
NAME                             105      CHARACTER(30) IN RPT1-DATA
```

```
5722ST1 V5R4M0 060210         Create SQL COBOL Program          CBLEX               08/06/02 11:09:13   Page   8
CROSS REFERENCE
PERCENTAGE                       42       DECIMAL(5,2)
                                          135
PHONENO                          134      CHARACTER(4) COLUMN IN CORPDATA.EMPLOYEE
PRENDATE                         50       DATE(10) IN PROJECT
PRENDATE                         ****     COLUMN
                                          217
PRENDATE                         213      DATE(10) COLUMN IN CORPDATA.PROJECT
PRINT-RECORD                     37       CHARACTER(132)
PROJECT                          50       STRUCTURE IN RPT1
PROJECT                          ****     TABLE IN CORPDATA
                                          213
PROJECT                          ****     TABLE
                                          215
PROJECT-NAME                     62       CHARACTER(36) IN RPT2
PROJECT-NAME                     112      CHARACTER(36) IN RPT2-DATA
PROJNAME                         50       VARCHAR(24) IN PROJECT
PROJNAME                         ****     COLUMN
                                          210 218
PROJNAME                         213      VARCHAR(24) COLUMN (NOT NULL) IN CORPDATA.PROJECT
PROJNO                           50       CHARACTER(6) IN PROJECT
                                          250
PROJNO                           61       CHARACTER(6) IN RPT2
PROJNO                           101      CHARACTER(6) IN RPT1-DATA
PROJNO                           110      CHARACTER(6) IN RPT2-DATA
PROJNO                           ****     COLUMN
                                          166 171
PROJNO                           168      CHARACTER(6) COLUMN (NOT NULL) IN CORPDATA.EMPPROJACT
PROJNO                           ****     COLUMN IN EMPPROJACT
                                          210 215 218
PROJNO                           ****     COLUMN IN PROJECT
                                          215
PROJNO                           213      CHARACTER(6) COLUMN (NOT NULL) IN CORPDATA.PROJECT
PRSTAFF                          50       DECIMAL(5,2) IN PROJECT
PRSTAFF                          213      DECIMAL(5,2) COLUMN IN CORPDATA.PROJECT
PRSTDATE                         50       DATE(10) IN PROJECT
PRSTDATE                         213      DATE(10) COLUMN IN CORPDATA.PROJECT
RAISE-DATE                       41       CHARACTER(11)
                                          217
RESPEMP                          50       CHARACTER(6) IN PROJECT
RESPEMP                          213      CHARACTER(6) COLUMN (NOT NULL) IN CORPDATA.PROJECT
RPT1                             49
RPT1-DATA                        100
RPT1-HEADERS                     75
RPT1-HEADER1                     76       IN RPT1-HEADERS
RPT1-HEADER2                     80       IN RPT1-HEADERS
RPT2                             60       STRUCTURE
                                          270
RPT2-DATA                        109
SS REFERENCE
RPT2-HEADERS                     85
RPT2-HEADER1                     86       IN RPT2-HEADERS
RPT2-HEADER2                     90       IN RPT2-HEADERS
RPT2-HEADER3                     95       IN RPT2-HEADERS
SALARY                           53       DECIMAL(8,2) IN RPT1
                                          251
SALARY                           107      IN RPT1-DATA
SALARY                           ****     COLUMN
                                          135 135 167 212
SALARY                           134      DECIMAL(9,2) COLUMN IN CORPDATA.EMPLOYEE
SEX                              134      CHARACTER(1) COLUMN IN CORPDATA.EMPLOYEE
TOTAL-PROJ-COST                  64       DECIMAL(12,2) IN RPT2
TOTAL-PROJ-COST                  116      IN RPT2-DATA
WORK-DAYS                        40       SMALL INTEGER PRECISION(4,0)
                                          212
WORKDEPT                         134      CHARACTER(3) COLUMN IN CORPDATA.EMPLOYEE
No errors found in source
  307 Source records processed
                               * * * * *  E N D  O F  L I S T I N G  * * * * *
```

Example: SQL statements in PL/I programs

This example program is written in the PL/I programming language.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 176.

```
5722ST1 V5R4M0 060210    Create SQL PL/I Program          PLIEX                08/06/02 12:53:36   Page   1
Source type...............PLI
Program name..............CORPDATA/PLIEX
Source file...............CORPDATA/SRC
Member....................PLIEX
To source file............QTEMP/QSQLTEMP
Options...................*SRC      *XREF
Target release............V5R4M0
INCLUDE file..............*SRCFILE
Commit....................*CHG
Allow copy of data........*YES
Close SQL cursor..........*ENDPGM
Allow blocking............*READ
Delay PREPARE.............*NO
Generation level..........10
Margins...................*SRCFILE
Printer file..............*LIBL/QSYSPRT
Date format...............*JOB
Date separator............*JOB
Time format...............*HMS
Time separator ...........*JOB
Replace...................*YES
Relational database.......*LOCAL
User .....................*CURRENT
RDB connect method........*DUW
Default collection........*NONE
Dynamic default
  collection..............*NO
Package name..............*PGMLIB/*PGM
Path......................*NAMING
SQL rules.................*DB2
User profile..............*NAMING
Dynamic user profile......*USER
Sort sequence.............*JOB
Language ID...............*JOB
IBM SQL flagging..........*NOFLAG
ANS flagging..............*NONE
Text......................*SRCMBRTXT
Source file CCSID.........65535
Job CCSID.................65535
Decimal result options:
  Maximum precision.......31
  Maximum scale...........31
  Minimum divide scale....0
Compiler options..........*NONE
Source member changed on 07/01/96  12:53:08
```

Figure 5. Sample PL/I program using SQL statements

```
5722ST1 V5R4M0 060210     Create SQL PL/I Program          PLIEX               08/06/02 12:53:36   Page   2
Record  *...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8   SEQNBR  Last change

    1    /* A sample program which updates the salaries for those employees    */             100
    2    /* whose current commission total is greater than or equal to the     */             200
    3    /* value of COMMISSION. The salaries of those who qualify are         */             300
    4    /* increased by the value of PERCENTAGE, retroactive to RAISE_DATE.  */             400
    5    /* A report is generated showing the projects which these employees  */             500
    6    /* have contributed to, ordered by project number and employee ID.   */             600
    7    /* A second report shows each project having an end date occurring   */             700
    8    /* after RAISE_DATE (i.e. is potentially affected by the retroactive */             800
    9    /* raises) with its total salary expenses and a count of employees    */             900
   10    /* who contributed to the project.                                    */            1000
   11    /**********************************************************************/            1100
   12                                                                                      1200
   13                                                                                      1300
   14    PLIEX: PROC;                                                                      1400
   15                                                                                      1500
   16      DCL RAISE_DATE CHAR(10);                                                        1600
   17      DCL WORK_DAYS  FIXED BIN(15);                                                   1700
   18      DCL COMMISSION FIXED DECIMAL(8,2);                                              1800
   19      DCL PERCENTAGE FIXED DECIMAL(5,2);                                              1900
   20                                                                                      2000
   21      /* File declaration for sysprint */                                             2100
   22      DCL SYSPRINT FILE EXTERNAL OUTPUT STREAM PRINT;                                 2200
   23                                                                                      2300
   24      /* Structure for report 1 */                                                    2400
   25      DCL 1 RPT1,                                                                     2500
   26  1%INCLUDE PROJECT (PROJECT, RECORD,,COMMA);                                         2600
   27            15 EMPNO     CHAR(6),                                                     2700
   28            15 NAME      CHAR(30),                                                    2800
   29            15 SALARY    FIXED DECIMAL(8,2);                                          2900
   30                                                                                      3000
   31      /* Structure for report 2 */                                                    3100
   32      DCL 1 RPT2,                                                                     3200
   33            15 PROJNO          CHAR(6),                                               3300
   34            15 PROJECT_NAME    CHAR(36),                                              3400
   35            15 EMPLOYEE_COUNT  FIXED BIN(15),                                         3500
   36            15 TOTL_PROJ_COST  FIXED DECIMAL(10,2);                                   3600
   37                                                                                      3700
   38    2 EXEC SQL INCLUDE SQLCA;                                                         3800
   39                                                                                      3900
   40      COMMISSION = 2000.00;                                                           4000
   41      PERCENTAGE = 1.04;                                                              4100
   42      RAISE_DATE = '1982-06-01';                                                      4200
   43      WORK_DAYS  = 253;                                                               4300
   44      OPEN FILE(SYSPRINT);                                                            4400
   45                                                                                      4500
   46      /* Update the selected employee's salaries by the new percentage. */            4600
   47      /* If an error occurs during the update, ROLLBACK the changes.    */            4700
   48   3 EXEC SQL WHENEVER SQLERROR GO TO UPDATE_ERROR;                                   4800
   49   4 EXEC SQL                                                                         4900
   50           UPDATE CORPDATA/EMPLOYEE                                                   5000
   51              SET SALARY = SALARY * :PERCENTAGE                                       5100
   52              WHERE COMM >= :COMMISSION ;                                             5200
   53                                                                                      5300
   54      /* Commit changes */                                                            5400
   55   5 EXEC SQL                                                                         5500
   56           COMMIT;                                                                    5600
   57      EXEC SQL WHENEVER SQLERROR GO TO REPORT_ERROR;                                  5700
   58                                                                                      5800
```

```
5722ST1 V5R4M0 060210     Create SQL PL/I Program          PLIEX             08/06/02 12:53:36   Page   3
Record  *...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8   SEQNBR  Last change
    59      /* Report the updated statistics for each project supported by one */             5900
    60      /* of the selected employees.                                       */             6000
    61                                                                                         6100
    62      /* Write out the header for Report 1 */                                            6200
    63      put file(sysprint)                                                                 6300
    64          edit('REPORT OF PROJECTS AFFECTED BY EMPLOYEE RAISES')                         6400
    65              (col(22),a);                                                               6500
    66      put file(sysprint)                                                                 6600
    67          edit('PROJECT','EMPID','EMPLOYEE NAME','SALARY')                               6700
    68              (skip(2),col(1),a,col(10),a,col(20),a,col(55),a);                          6800
    69                                                                                         6900
    70   6 exec sql                                                                            7000
    71           declare c1 cursor for                                                         7100
    72             select DISTINCT projno, EMPPROJACT.empno,                                   7200
    73                            lastname||', '||firstnme, salary                             7300
    74             from CORPDATA/EMPPROJACT, CORPDATA/EMPLOYEE                                 7400
    75             where EMPPROJACT.empno = EMPLOYEE.empno and                                 7500
    76                   comm >= :COMMISSION                                                   7600
    77             order by projno, empno;                                                     7700
    78   7 EXEC SQL                                                                            7800
    79           OPEN C1;                                                                      7900
    80                                                                                         8000
    81      /* Fetch and write the rows to SYSPRINT */                                         8100
    82   8 EXEC SQL WHENEVER NOT FOUND GO TO DONE1;                                            8200
    83                                                                                         8300
    84      DO UNTIL (SQLCODE ^= 0);                                                           8400
    85      9 EXEC SQL                                                                         8500
    86             FETCH C1 INTO :RPT1.PROJNO, :rpt1.EMPNO, :RPT1.NAME,                        8600
    87                           :RPT1.SALARY;                                                 8700
    88        PUT FILE(SYSPRINT)                                                               8800
    89            EDIT(RPT1.PROJNO,RPT1.EMPNO,RPT1.NAME,RPT1.SALARY)                           8900
    90                (SKIP,COL(1),A,COL(10),A,COL(20),A,COL(54),F(8,2));                      9000
    91      END;                                                                               9100
    92                                                                                         9200
    93    DONE1:                                                                               9300
    94  10 EXEC SQL                                                                            9400
    95           CLOSE C1;                                                                     9500
    96                                                                                         9600
    97      /* For all projects ending at a date later than 'raise_date'       */              9700
    98      /* (i.e. those projects potentially affected by the salary raises) */              9800
    99      /* generate a report containing the project number, project name   */              9900
   100      /* the count of employees participating in the project and the     */             10000
   101      /* total salary cost of the project.                               */             10100
   102                                                                                        10200
   103      /* Write out the header for Report 2 */                                           10300
   104      PUT FILE(SYSPRINT) EDIT('ACCUMULATED STATISTICS BY PROJECT')                      10400
   105                             (SKIP(3),COL(22),A);                                       10500
   106      PUT FILE(SYSPRINT)                                                                10600
   107          EDIT('PROJECT','NUMBER OF','TOTAL')                                           10700
   108              (SKIP(2),COL(1),A,COL(48),A,COL(63),A);                                   10800
   109      PUT FILE(SYSPRINT)                                                                10900
   110          EDIT('NUMBER','PROJECT NAME','EMPLOYEES','COST')                              11000
   111              (SKIP,COL(1),A,COL(10),A,COL(48),A,COL(63),A,SKIP);                       11100
   112                                                                                        11200
```

```
5722ST1 V5R4M0 060210         Create SQL PL/I Program          PLIEX                  08/06/02 12:53:36   Page   4
Record  *...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8   SEQNBR  Last change
  113  11 EXEC SQL                                                                       11300
  114           DECLARE C2 CURSOR FOR                                                      11400
  115             SELECT EMPPROJACT.PROJNO, PROJNAME, COUNT(*),                            11500
  116                SUM( (DAYS(EMENDATE) - DAYS(EMSTDATE)) * EMPTIME *                    11600
  117                     DECIMAL(( SALARY / :WORK_DAYS ),8,2) )                           11700
  118             FROM CORPDATA/EMPPROJACT, CORPDATA/PROJECT, CORPDATA/EMPLOYEE            11800
  119             WHERE EMPPROJACT.PROJNO=PROJECT.PROJNO  AND                              11900
  120                   EMPPROJACT.EMPNO =EMPLOYEE.EMPNO  AND                              12000
  121                   PRENDATE > :RAISE_DATE                                             12100
  122             GROUP BY EMPPROJACT.PROJNO, PROJNAME                                     12200
  123             ORDER BY 1;                                                              12300
  124      EXEC SQL                                                                        12400
  125           OPEN C2;                                                                   12500
  126                                                                                      12600
  127      /* Fetch and write the rows to SYSPRINT */                                      12700
  128      EXEC SQL WHENEVER NOT FOUND GO TO DONE2;                                        12800
  129                                                                                      12900
  130      DO UNTIL (SQLCODE ^= 0);                                                        13000
  131    12 EXEC SQL                                                                       13100
  132             FETCH C2 INTO :RPT2;                                                     13200
  133        PUT FILE(SYSPRINT)                                                            13300
  134            EDIT(RPT2.PROJNO,RPT2.PROJECT_NAME,EMPLOYEE_COUNT,                        13400
  135                 TOTL_PROJ_COST)                                                      13500
  136                (SKIP,COL(1),A,COL(10),A,COL(50),F(4),COL(62),F(8,2));                13600
  137      END;                                                                            13700
  138                                                                                      13800
  139    DONE2:                                                                            13900
  140      EXEC SQL                                                                        14000
  141           CLOSE C2;                                                                  14100
  142      GO TO FINISHED;                                                                 14200
  143                                                                                      14300
  144      /* Error occurred while updating table. Inform user and rollback   */            14400
  145      /* changes.                                                        */            14500
  146    UPDATE_ERROR:                                                                     14600
  147  13 EXEC SQL WHENEVER SQLERROR CONTINUE;                                             14700
  148      PUT FILE(SYSPRINT) EDIT('*** ERROR Occurred while updating table.'||            14800
  149       '  SQLCODE=',SQLCODE)(A,F(5));                                                 14900
  150  14 EXEC SQL                                                                         15000
  151           ROLLBACK;                                                                  15100
  152      GO TO FINISHED;                                                                 15200
  153                                                                                      15300
  154      /* Error occurred while generating reports.  Inform user and exit.  */           15400
  155    REPORT_ERROR:                                                                     15500
  156      PUT FILE(SYSPRINT) EDIT('*** ERROR Occurred while generating '||                15600
  157       'reports.  SQLCODE=',SQLCODE)(A,F(5));                                         15700
  158       GO TO FINISHED;                                                                15800
  159                                                                                      15900
  160      /* All done  */                                                                 16000
  161    FINISHED:                                                                         16100
  162      CLOSE FILE(SYSPRINT);                                                           16200
  163      RETURN;                                                                         16300
  164                                                                                      16400
  165    END PLIEX;                                                                        16500
                               * * * * *  E N D  O F  S O U R C E  * * * * *
```

```
5722ST1 V5R4M0 060210       Create SQL PL/I Program          PLIEX                 08/06/02 12:53:36   Page   5
CROSS REFERENCE
Data Names                     Define    Reference
ACTNO                             74       SMALL INTEGER PRECISION(4,0) COLUMN (NOT NULL) IN CORPDATA.EMPPROJACT
BIRTHDATE                         74       DATE(10) COLUMN IN CORPDATA.EMPLOYEE
BONUS                             74       DECIMAL(9,2) COLUMN IN CORPDATA.EMPLOYEE
COMM                              ****     COLUMN
                                           52 76
COMM                              74       DECIMAL(9,2) COLUMN IN CORPDATA.EMPLOYEE
COMMISSION                        18       DECIMAL(8,2)
                                           52 76
CORPDATA                          ****     COLLECTION
                                           50 74 74 118 118 118
C1                                71       CURSOR
                                           79 86 95
C2                                114      CURSOR
                                           125 132 141
DEPTNO                            26       CHARACTER(3) IN RPT1
DEPTNO                            118      CHARACTER(3) COLUMN (NOT NULL) IN CORPDATA.PROJECT
DONE1                             ****     LABEL
                                           82
DONE2                             ****     LABEL
                                           128
EDLEVEL                           74       SMALL INTEGER PRECISION(4,0) COLUMN (NOT NULL) IN CORPDATA.EMPLOYEE
EMENDATE                          74       DATE(10) COLUMN IN CORPDATA.EMPPROJACT
EMENDATE                          ****     COLUMN
                                           116
EMPLOYEE                          ****     TABLE IN CORPDATA
                                           50 74 118
EMPLOYEE                          ****     TABLE
                                           75 120
EMPLOYEE_COUNT                    35       SMALL INTEGER PRECISION(4,0) IN RPT2
EMPNO                             27       CHARACTER(6) IN RPT1
                                           86
EMPNO                             ****     COLUMN IN EMPPROJACT
                                           72 75 77 120
EMPNO                             ****     COLUMN IN EMPLOYEE
                                           75 120
EMPNO                             74       CHARACTER(6) COLUMN (NOT NULL) IN CORPDATA.EMPPROJACT
EMPNO                             74       CHARACTER(6) COLUMN (NOT NULL) IN CORPDATA.EMPLOYEE
EMPPROJACT                        ****     TABLE
                                           72 75 115 119 120 122
EMPPROJACT                        ****     TABLE IN CORPDATA
                                           74 118
EMPTIME                           74       DECIMAL(5,2) COLUMN IN CORPDATA.EMPPROJACT
EMPTIME                           ****     COLUMN
                                           116
EMSTDATE                          74       DATE(10) COLUMN IN CORPDATA.EMPPROJACT
EMSTDATE                          ****     COLUMN
                                           116
FIRSTNME                          ****     COLUMN
                                           73
FIRSTNME                          74       VARCHAR(12) COLUMN (NOT NULL) IN CORPDATA.EMPLOYEE
HIREDATE                          74       DATE(10) COLUMN IN CORPDATA.EMPLOYEE
JOB                               74       CHARACTER(8) COLUMN IN CORPDATA.EMPLOYEE
LASTNAME                          ****     COLUMN
                                           73
LASTNAME                          74       VARCHAR(15) COLUMN (NOT NULL) IN CORPDATA.EMPLOYEE
MAJPROJ                           26       CHARACTER(6) IN RPT1
MAJPROJ                           118      CHARACTER(6) COLUMN IN CORPDATA.PROJECT
MIDINIT                           74       CHARACTER(1) COLUMN (NOT NULL) IN CORPDATA.EMPLOYEE
NAME                              28       CHARACTER(30) IN RPT1
                                           86
PERCENTAGE                        19       DECIMAL(5,2)
                                           51
PHONENO                           74       CHARACTER(4) COLUMN IN CORPDATA.EMPLOYEE
```

```
5722ST1 V5R4M0 060210          Create SQL PL/I Program          PLIEX                08/06/02 12:53:36   Page   6
CROSS REFERENCE
PRENDATE                         26       DATE(10) IN RPT1
PRENDATE                         ****     COLUMN
                                          121
PRENDATE                         118      DATE(10) COLUMN IN CORPDATA.PROJECT
PROJECT                          ****     TABLE IN CORPDATA
                                          118
PROJECT                          ****     TABLE
                                          119
PROJECT_NAME                     34       CHARACTER(36) IN RPT2
PROJNAME                         26       VARCHAR(24) IN RPT1
PROJNAME                         ****     COLUMN
                                          115 122
PROJNAME                         118      VARCHAR(24) COLUMN (NOT NULL) IN CORPDATA.PROJECT
PROJNO                           26       CHARACTER(6) IN RPT1
                                          86
PROJNO                           33       CHARACTER(6) IN RPT2
PROJNO                           ****     COLUMN
                                          72 77
PROJNO                           74       CHARACTER(6) COLUMN (NOT NULL) IN CORPDATA.EMPPROJACT
PROJNO                           ****     COLUMN IN EMPPROJACT
                                          115 119 122
PROJNO                           ****     COLUMN IN PROJECT
                                          119
PROJNO                           118      CHARACTER(6) COLUMN (NOT NULL) IN CORPDATA.PROJECT
PRSTAFF                          26       DECIMAL(5,2) IN RPT1
PRSTAFF                          118      DECIMAL(5,2) COLUMN IN CORPDATA.PROJECT
PRSTDATE                         26       DATE(10) IN RPT1
PRSTDATE                         118      DATE(10) COLUMN IN CORPDATA.PROJECT
RAISE_DATE                       16       CHARACTER(10)
                                          121
REPORT_ERROR                     ****     LABEL
                                          57
RESPEMP                          26       CHARACTER(6) IN RPT1
RESPEMP                          118      CHARACTER(6) COLUMN (NOT NULL) IN CORPDATA.PROJECT
RPT1                             25       STRUCTURE
RPT2                             32       STRUCTURE
                                          132
SALARY                           29       DECIMAL(8,2) IN RPT1
                                          87
SALARY                           ****     COLUMN
                                          51 51 73 117
SALARY                           74       DECIMAL(9,2) COLUMN IN CORPDATA.EMPLOYEE
SEX                              74       CHARACTER(1) COLUMN IN CORPDATA.EMPLOYEE
SYSPRINT                         22
TOTL_PROJ_COST                   36       DECIMAL(10,2) IN RPT2
UPDATE_ERROR                     ****     LABEL
                                          48
WORK_DAYS                        17       SMALL INTEGER PRECISION(4,0)
                                          117
WORKDEPT                         74       CHARACTER(3) COLUMN IN CORPDATA.EMPLOYEE
No errors found in source
  165 Source records processed
                               * * * * *  E N D  O F  L I S T I N G  * * * * *
```

Example: SQL statements in RPG/400 programs

This example program is written in the RPG programming language.

Note: By using the code examples, you agree to the terms of the “Code license and disclaimer information” on page 176.

```
5722ST1 V5R4M0 060210      Create SQL RPG Program           RPGEX              08/06/02 12:55:22   Page   1
Source type...............RPG
Program name..............CORPDATA/RPGEX
Source file...............CORPDATA/SRC
Member....................RPGEX
To source file............QTEMP/QSQLTEMP
Options...................*SRC      *XREF
Target release............V5R4M0
INCLUDE file..............*SRCFILE
Commit....................*CHG
Allow copy of data........*YES
Close SQL cursor..........*ENDPGM
Allow blocking............*READ
Delay PREPARE.............*NO
Generation level..........10
Printer file..............*LIBL/QSYSPRT
Date format...............*JOB
Date separator............*JOB
Time format...............*HMS
Time separator ...........*JOB
Replace...................*YES
Relational database.......*LOCAL
User .....................*CURRENT
RDB connect method........*DUW
Default collection........*NONE
Dynamic default
  collection..............*NO
Package name..............*PGMLIB/*PGM
Path......................*NAMING
SQL rules.................*DB2
User profile..............*NAMING
Dynamic user profile......*USER
Sort sequence.............*JOB
Language ID...............*JOB
IBM SQL flagging..........*NOFLAG
ANS flagging..............*NONE
Text......................*SRCMBRTXT
Source file CCSID.........65535
Job CCSID.................65535
Decimal result options:
  Maximum precision.......31
  Maximum scale...........31
  Minimum divide scale....0
Compiler options..........*NONE
Source member changed on 07/01/96  17:06:17
```

Figure 6. Sample RPG/400 program using SQL statements

```
5722ST1 V5R4M0 060210      Create SQL RPG Program           RPGEX               08/06/02 12:55:22   Page   2
Record  *...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8   SEQNBR  Last change
    1        H                                                                                100
    2        F*  File declaration for QPRINT                                                  200
    3        F*                                                                               300
    4        FQPRINT  O   F     132            PRINTER                                        400
    5        I*                                                                               500
    6        I* Structure for report 1.                                                       600
    7        I*                                                                               700
    8      1 IRPT1      E DSPROJECT                                                           800
    9        I              PROJNAME                        PROJNM                            900
   10        I              RESPEMP                         RESEM                            1000
   11        I              PRSTAFF                         STAFF                            1100
   12        I              PRSTDATE                        PRSTD                            1200
   13        I              PRENDATE                        PREND                            1300
   14        I              MAJPROJ                         MAJPRJ                           1400
   15        I*                                                                              1500
   16        I            DS                                                                 1600
   17        I                                        1   6 EMPNO                            1700
   18        I                                        7  36 NAME                             1800
   19        I                                    P  37  412SALARY                           1900
   20        I*                                                                              2000
   21        I* Structure for report 2.                                                      2100
   22        I*                                                                              2200
   23        IRPT2        DS                                                                 2300
   24        I                                        1   6 PRJNUM                           2400
   25        I                                        7  42 PNAME                            2500
   26        I                                    B  43  440EMPCNT                           2600
   27        I                                    P  45  492PRCOST                           2700
   28        I*                                                                              2800
   29        I            DS                                                                 2900
   30        I                                    B   1   20WRKDAY                           3000
   31        I                                    P   3   62COMMI                            3100
   32        I                                        7  16 RDATE                            3200
   33        I                                    P  17  202PERCNT                           3300
   34      2 C*                                                                              3400
   35        C                     Z-ADD253       WRKDAY                                     3500
   36        C                     Z-ADD2000.00   COMMI                                      3600
   37        C                     Z-ADD1.04      PERCNT                                     3700
   38        C                     MOVEL'1982-06-'RDATE                                      3800
   39        C                     MOVE '01'      RDATE                                      3900
   40        C                     SETON                     LR                              3901
   41        C*                                                                              4000
   42        C* Update the selected projects by the new percentage. If an                    4100
   43        C* error occurs during the update, ROLLBACK the changes.                        4200
   44        C*                                                                              4300
   45      3 C/EXEC SQL WHENEVER SQLERROR GOTO UPDERR                                        4400
   46        C/END-EXEC                                                                      4500
   47        C*                                                                              4600
   48      4 C/EXEC SQL                                                                      4700
   49        C+ UPDATE CORPDATA/EMPLOYEE                                                     4800
   50        C+    SET SALARY = SALARY * :PERCNT                                             4900
   51        C+    WHERE COMM >= :COMMI                                                      5000
   52        C/END-EXEC                                                                      5100
   53        C*                                                                              5200
   54        C* Commit changes.                                                              5300
   55        C*                                                                              5400
   56      5 C/EXEC SQL COMMIT                                                               5500
   57        C/END-EXEC                                                                      5600
   58        C*                                                                              5700
   59        C/EXEC SQL WHENEVER SQLERROR GO TO RPTERR                                       5800
   60        C/END-EXEC                                                                      5900
```

```
5722ST1 V5R4M0 060210         Create SQL RPG Program           RPGEX           08/06/02 12:55:22   Page   3
Record  *...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8   SEQNBR  Last change
   61        C*                                                                                6000
   62        C* Report the updated statistics for each employee assigned to                    6100
   63        C* selected projects.                                                             6200
   64        C*                                                                                6300
   65        C* Write out the header for report 1.                                             6400
   66        C*                                                                                6500
   67        C                     EXCPTRECA                                                   6600
   68      6 C/EXEC SQL DECLARE C1 CURSOR FOR                                                  6700
   69        C+    SELECT DISTINCT PROJNO, EMPPROJACT.EMPNO,                                   6800
   70        C+           LASTNAME||', '||FIRSTNME, SALARY                                     6900
   71        C+        FROM CORPDATA/EMPPROJACT, CORPDATA/EMPLOYEE                             7000
   72        C+        WHERE EMPPROJACT.EMPNO = EMPLOYEE.EMPNO AND                             7100
   73        C+              COMM >= :COMMI                                                    7200
   74        C+        ORDER BY PROJNO, EMPNO                                                  7300
   75        C/END-EXEC                                                                        7400
   76        C*                                                                                7500
   77      7 C/EXEC SQL                                                                        7600
   78        C+  OPEN C1                                                                       7700
   79        C/END-EXEC                                                                        7800
   80        C*                                                                                7900
   81        C* Fetch and write the rows to QPRINT.                                            8000
   82        C*                                                                                8100
   83      8 C/EXEC SQL WHENEVER NOT FOUND GO TO DONE1                                         8200
   84        C/END-EXEC                                                                        8300
   85        C           SQLCOD    DOUNE0                                                      8400
   86        C/EXEC SQL                                                                        8500
   87      9 C+   FETCH C1 INTO :PROJNO, :EMPNO, :NAME, :SALARY                                8600
   88        C/END-EXEC                                                                        8700
   89        C                     EXCPTRECB                                                   8800
   90        C                     END                                                         8900
   91        C           DONE1     TAG                                                         9000
   92        C/EXEC SQL                                                                        9100
   93     10 C+  CLOSE C1                                                                      9200
   94        C/END-EXEC                                                                        9300
   95        C*                                                                                9400
   96        C* For all project ending at a date later than the raise date                     9500
   97        C* (i.e. those projects potentially affected by the salary raises)                9600
   98        C* generate a report containing the project number, project name,                 9700
   99        C* the count of employees participating in the project and the                    9800
  100        C* total salary cost of the project.                                              9900
  101        C*                                                                               10000
  102        C* Write out the header for report 2.                                            10100
  103        C*                                                                               10200
  104        C                     EXCPTRECC                                                  10300
  105     11 C/EXEC SQL                                                                       10400
  106        C+  DECLARE C2 CURSOR FOR                                                        10500
  107        C+    SELECT EMPPROJACT.PROJNO, PROJNAME, COUNT(*),                              10600
  108        C+       SUM((DAYS(EMENDATE) - DAYS(EMSTDATE)) * EMPTIME *                       10700
  109        C+           DECIMAL((SALARY/:WRKDAY),8,2))                                      10800
  110        C+    FROM CORPDATA/EMPPROJACT, CORPDATA/PROJECT, CORPDATA/EMPLOYEE              10900
  111        C+    WHERE EMPPROJACT.PROJNO = PROJECT.PROJNO AND                               11000
  112        C+          EMPPROJACT.EMPNO = EMPLOYEE.EMPNO AND                                11100
  113        C+          PRENDATE > :RDATE                                                    11200
  114        C+    GROUP BY EMPPROJACT.PROJNO, PROJNAME                                       11300
  115        C+    ORDER BY 1                                                                 11400
  116        C/END-EXEC                                                                       11500
  117        C*                                                                               11600
  118        C/EXEC SQL OPEN C2                                                               11700
  119        C/END-EXEC                                                                       11800
  120        C*                                                                               11900
  121        C* Fetch and write the rows to QPRINT.                                           12000
  122        C*                                                                               12100
  123        C/EXEC SQL WHENEVER NOT FOUND GO TO DONE2                                        12200
  124        C/END-EXEC                                                                       12300
```

```
5722ST1 V5R4M0 060210    Create SQL RPG Program           RPGEX               08/06/02 12:55:22   Page   4
  125        C           SQLCOD    DOUNE0                                                   12400
  126        C/EXEC SQL                                                                     12500
  127     12 C+   FETCH C2 INTO :RPT2                                                       12600
  128        C/END-EXEC                                                                     12700
  129        C                     EXCPTRECD                                                12800
  130        C                     END                                                      12900
  131        C           DONE2     TAG                                                      13000
  132        C/EXEC SQL CLOSE C2                                                            13100
  133        C/END-EXEC                                                                     13200
  134        C                     RETRN                                                    13300
  135        C*                                                                             13400
  136        C* Error occurred while updating table.  Inform user and rollback               13500
  137        C* changes.                                                                    13600
  138        C*                                                                             13700
  139        C           UPDERR    TAG                                                      13800
  140        C                     EXCPTRECE                                                13900
  141     13 C/EXEC SQL WHENEVER SQLERROR CONTINUE                                          14000
  142        C/END-EXEC                                                                     14100
  143        C*                                                                             14200
  144     14 C/EXEC SQL                                                                     14300
  145        C+   ROLLBACK                                                                  14400
  146        C/END-EXEC                                                                     14500
  147        C                     RETRN                                                    14600
  148        C*                                                                             14700
  149        C* Error occurred while generating reports.  Inform user and exit.              14800
  150        C*                                                                             14900
  151        C           RPTERR    TAG                                                      15000
  152        C                     EXCPTRECF                                                15100
  153        C*                                                                             15200
  154        C* All done.                                                                   15300
  155        C*                                                                             15400
  156        C           FINISH    TAG                                                      15500
  157        OQPRINT  E 0201           RECA                                                 15700
  158        O                                   45 'REPORT OF PROJECTS AFFEC'              15800
  159        O                                   64 'TED BY EMPLOYEE RAISES'                15900
  160        O        E 01             RECA                                                 16000
  161        O                                    7 'PROJECT'                               16100
  162        O                                   17 'EMPLOYEE'                              16200
  163        O                                   32 'EMPLOYEE NAME'                         16300
  164        O                                   60 'SALARY'                                16400
  165        O        E 01             RECB                                                 16500
  166        O                         PROJNO     6                                         16600
  167        O                         EMPNO     15                                         16700
  168        O                         NAME      50                                         16800
  169        O                         SALARYL   61                                         16900
  170        O        E 22             RECC                                                 17000
  171        O                                   42 'ACCUMULATED STATISTIC'                 17100
  172        O                                   54 'S BY PROJECT'                          17200
  173        O        E 01             RECC                                                 17300
  174        O                                    7 'PROJECT'                               17400
  175        O                                   56 'NUMBER OF'                             17500
  176        O                                   67 'TOTAL'                                 17600
  177        O        E 02             RECC                                                 17700
  178        O                                    6 'NUMBER'                                17800
  179        O                                   21 'PROJECT NAME'                          17900
  180        O                                   56 'EMPLOYEES'                             18000
  181        O                                   66 'COST'                                  18100
  182        O        E 01             RECD                                                 18200
  183        O                         PRJNUM     6                                         18300
  184        O                         PNAME     45                                         18400
  185        O                         EMPCNTL   54                                         18500
  186        O                         PRCOSTL   70                                         18600
  187        O        E 01             RECE                                                 18700
  188        O                                   28 '*** ERROR Occurred while'              18800
  189        O                                   52 ' updating table. SQLCODE'              18900
  190        O                                   53 '='                                     19000
  191        O                         SQLCODL   62                                         19100
  192        O        E 01             RECF                                                 19200
  193        O                                   28 '*** ERROR Occurred while'              19300
  194        O                                   52 ' generating reports. SQL'              19400
  195        O                                   57 'CODE='                                 19500
  196        O                         SQLCODL   67                                         19600
                                * * * * *  E N D  O F  S O U R C E  * * * * *
```

```
5722ST1 V5R4M0 060210      Create SQL RPG Program           RPGEX             08/06/02 12:55:22   Page   5
CROSS REFERENCE
Data Names                     Define    Reference
ACTNO                             68       SMALL INTEGER PRECISION(4,0) COLUMN (NOT NULL) IN CORPDATA.EMPPROJACT
BIRTHDATE                         48       DATE(10) COLUMN IN CORPDATA.EMPLOYEE
BONUS                             48       DECIMAL(9,2) COLUMN IN CORPDATA.EMPLOYEE
COMM                              ****     COLUMN
                                           48 68
COMM                              48       DECIMAL(9,2) COLUMN IN CORPDATA.EMPLOYEE
COMMI                             31       DECIMAL(7,2)
                                           48 68
CORPDATA                          ****     COLLECTION
                                           48 68 68 105 105 105
C1                                68       CURSOR
                                           77 86 92
C2                                105      CURSOR
                                           118 126 132
DEPTNO                            8        CHARACTER(3) IN RPT1
DEPTNO                            105      CHARACTER(3) COLUMN (NOT NULL) IN CORPDATA.PROJECT
DONE1                             91       LABEL
                                           83
DONE2                             131      LABEL
                                           123
EDLEVEL                           48       SMALL INTEGER PRECISION(4,0) COLUMN (NOT NULL) IN CORPDATA.EMPLOYEE
EMENDATE                          68       DATE(10) COLUMN IN CORPDATA.EMPPROJACT
EMENDATE                          ****     COLUMN
                                           105
EMPCNT                            26       SMALL INTEGER PRECISION(4,0) IN RPT2
EMPLOYEE                          ****     TABLE IN CORPDATA
                                           48 68 105
EMPLOYEE                          ****     TABLE
                                           68 105
EMPNO                             17       CHARACTER(6)
                                           86
EMPNO                             48       CHARACTER(6) COLUMN (NOT NULL) IN CORPDATA.EMPLOYEE
EMPNO                             ****     COLUMN IN EMPPROJACT
                                           68 68 68 105
EMPNO                             ****     COLUMN IN EMPLOYEE
                                           68 105
EMPNO                             68       CHARACTER(6) COLUMN (NOT NULL) IN CORPDATA.EMPPROJACT
EMPPROJACT                        ****     TABLE
                                           68 68 105 105 105 105
EMPPROJACT                        ****     TABLE IN CORPDATA
                                           68 105
EMPTIME                           68       DECIMAL(5,2) COLUMN IN CORPDATA.EMPPROJACT
EMPTIME                           ****     COLUMN
                                           105
EMSTDATE                          68       DATE(10) COLUMN IN CORPDATA.EMPPROJACT
EMSTDATE                          ****     COLUMN
                                           105
FINISH                            156      LABEL
FIRSTNME                          48       VARCHAR(12) COLUMN (NOT NULL) IN CORPDATA.EMPLOYEE
FIRSTNME                          ****     COLUMN
                                           68
HIREDATE                          48       DATE(10) COLUMN IN CORPDATA.EMPLOYEE
JOB                               48       CHARACTER(8) COLUMN IN CORPDATA.EMPLOYEE
LASTNAME                          48       VARCHAR(15) COLUMN (NOT NULL) IN CORPDATA.EMPLOYEE
LASTNAME                          ****     COLUMN
                                           68
MAJPRJ                            8        CHARACTER(6) IN RPT1
MAJPROJ                           105      CHARACTER(6) COLUMN IN CORPDATA.PROJECT
MIDINIT                           48       CHARACTER(1) COLUMN (NOT NULL) IN CORPDATA.EMPLOYEE
NAME                              18       CHARACTER(30)
                                           86
PERCNT                            33       DECIMAL(7,2)
                                           48
PHONENO                           48       CHARACTER(4) COLUMN IN CORPDATA.EMPLOYEE
PNAME                             25       CHARACTER(36) IN RPT2
PRCOST                            27       DECIMAL(9,2) IN RPT2
PREND                             8        DATE(10) IN RPT1
PRENDATE                          ****     COLUMN
                                           105
```

```
5722ST1 V5R4M0 060210          Create SQL RPG Program           RPGEX           08/06/02 12:55:22   Page   6
PRENDATE                         105      DATE(10) COLUMN IN CORPDATA.PROJECT
PRJNUM                           24       CHARACTER(6) IN RPT2
CROSS REFERENCE
PROJECT                          ****     TABLE IN CORPDATA
                                          105
PROJECT                          ****     TABLE
                                          105
PROJNAME                         ****     COLUMN
                                          105 105
PROJNAME                         105      VARCHAR(24) COLUMN (NOT NULL) IN CORPDATA.PROJECT
PROJNM                           8        VARCHAR(24) IN RPT1
PROJNO                           8        CHARACTER(6) IN RPT1
                                          86
PROJNO                           ****     COLUMN
                                          68 68
PROJNO                           68       CHARACTER(6) COLUMN (NOT NULL) IN CORPDATA.EMPPROJACT
PROJNO                           ****     COLUMN IN EMPPROJACT
                                          105 105 105
PROJNO                           ****     COLUMN IN PROJECT
                                          105
PROJNO                           105      CHARACTER(6) COLUMN (NOT NULL) IN CORPDATA.PROJECT
PRSTAFF                          105      DECIMAL(5,2) COLUMN IN CORPDATA.PROJECT
PRSTD                            8        DATE(10) IN RPT1
PRSTDATE                         105      DATE(10) COLUMN IN CORPDATA.PROJECT
RDATE                            32       CHARACTER(10)
                                          105
RESEM                            8        CHARACTER(6) IN RPT1
RESPEMP                          105      CHARACTER(6) COLUMN (NOT NULL) IN CORPDATA.PROJECT
RPTERR                           151      LABEL
                                          59
RPT1                             8        STRUCTURE
RPT2                             23       STRUCTURE
                                          126
SALARY                           19       DECIMAL(9,2)
                                          86
SALARY                           ****     COLUMN
                                          48 48 68 105
SALARY                           48       DECIMAL(9,2) COLUMN IN CORPDATA.EMPLOYEE
SEX                              48       CHARACTER(1) COLUMN IN CORPDATA.EMPLOYEE
STAFF                            8        DECIMAL(5,2) IN RPT1
UPDERR                           139      LABEL
                                          45
WORKDEPT                         48       CHARACTER(3) COLUMN IN CORPDATA.EMPLOYEE
WRKDAY                           30       SMALL INTEGER PRECISION(4,0)
                                          105
No errors found in source
  196 Source records processed
                               * * * * *  E N D  O F  L I S T I N G  * * * * *
```

Example: SQL statements in ILE RPG programs

This example program is written in the ILE RPG programming language.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 176.

```
5722ST1 V5R4M0 060210        Create SQL ILE RPG Object         RPGLEEX            08/06/02 16:03:02   Page   1
Source type...............RPG
Object name...............CORPDATA/RPGLEEX
Source file...............CORPDATA/SRC
Member....................*OBJ
To source file............QTEMP/QSQLTEMP1
Options...................*XREF
RPG preprocessor options..*NONE
Listing option............*PRINT
Target release............V5R4M0
INCLUDE file..............*SRCFILE
Commit....................*CHG
Allow copy of data........*YES
Close SQL cursor..........*ENDMOD
Allow blocking............*READ
Delay PREPARE.............*NO
Generation level..........10
Printer file..............*LIBL/QSYSPRT
Date format...............*JOB
Date separator............*JOB
Time format...............*HMS
Time separator ...........*JOB
Replace...................*YES
Relational database.......*LOCAL
User .....................*CURRENT
RDB connect method........*DUW
Default collection........*NONE
Dynamic default
  collection..............*NO
Package name..............*OBJLIB/*OBJ
Path......................*NAMING
SQL rules.................*DB2
Created object type.......*PGM
Debugging view............*NONE
User profile..............*NAMING
Dynamic user profile......*USER
Sort sequence.............*JOB
Language ID...............*JOB
IBM SQL flagging..........*NOFLAG
ANS flagging..............*NONE
Text......................*SRCMBRTXT
Source file CCSID.........65535
Job CCSID.................65535
Decimal result options:
  Maximum precision.......31
  Maximum scale...........31
  Minimum divide scale....0
Compiler options..........*NONE
Source member changed on 07/01/96  15:55:32
```

Figure 7. Sample ILE RPG program using SQL statements

```
5722ST1 V5R4M0 060210       Create SQL ILE RPG Object         RPGLEEX            08/06/02 16:03:02   Page   2
Record  *...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8 SEQNBR Last change Comments
    1        H                                                                                100
    2        F*  File declaration for QPRINT                                                  200
    3        F*                                                                               300
    4        FQPRINT    O    F  132        PRINTER                                            400
    5        D*                                                                               500
    6        D* Structure for report 1.                                                       600
    7        D*                                                                               700
    8      1 DRPT1           E DS                  EXTNAME(PROJECT)                           800
    9        D*                                                                               900
   10        D                 DS                                                            1000
   11        D EMPNO                   1      6                                              1100
   12        D NAME                    7     36                                              1200
   13        D SALARY                 37     41P 2                                           1300
   14        D*                                                                              1400
   15        D* Structure for report 2.                                                      1500
   16        D*                                                                              1600
   17        DRPT2             DS                                                            1700
   18        D PRJNUM                  1      6                                              1800
   19        D PNAME                   7     42                                              1900
   20        D EMPCNT                 43     44B 0                                           2000
   21        D PRCOST                 45     49P 2                                           2100
   22        D*                                                                              2200
   23        D                 DS                                                            2300
   24        D WRKDAY                  1      2B 0                                           2400
   25        D COMMI                   3      6P 2                                           2500
   26        D RDATE                   7     16                                              2600
   27        D PERCNT                 17     20P 2                                           2700
   28         *                                                                              2800
   29      2 C                   Z-ADD     253           WRKDAY                              2900
   30        C                   Z-ADD     2000.00       COMMI                               3000
   31        C                   Z-ADD     1.04          PERCNT                              3100
   32        C                   MOVEL     '1982-06-'    RDATE                               3200
   33        C                   MOVE      '01'          RDATE                               3300
   34        C                   SETON                                        LR             3400
   35        C*                                                                              3500
   36        C* Update the selected projects by the new percentage. If an                    3600
   37        C* error occurs during the update, ROLLBACK the changes.                        3700
   38        C*                                                                              3800
   39      3 C/EXEC SQL WHENEVER SQLERROR GOTO UPDERR                                        3900
   40        C/END-EXEC                                                                      4000
   41        C*                                                                              4100
   42        C/EXEC SQL                                                                      4200
   43      4 C+ UPDATE CORPDATA/EMPLOYEE                                                     4300
   44        C+    SET SALARY = SALARY * :PERCNT                                             4400
   45        C+    WHERE COMM >= :COMMI                                                      4500
   46        C/END-EXEC                                                                      4600
   47        C*                                                                              4700
   48        C* Commit changes.                                                              4800
   49        C*                                                                              4900
   50      5 C/EXEC SQL COMMIT                                                               5000
   51        C/END-EXEC                                                                      5100
   52        C*                                                                              5200
   53        C/EXEC SQL WHENEVER SQLERROR GO TO RPTERR                                       5300
   54        C/END-EXEC                                                                      5400
   55        C*                                                                              5500
   56        C* Report the updated statistics for each employee assigned to                  5600
   57        C* selected projects.                                                           5700
   58        C*                                                                              5800
                                                                   12000
```

```
5722ST1 V5R4M0 060210       Create SQL ILE RPG Object         RPGLEEX             08/06/02 16:03:02   Page   3
Record  *...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8  SEQNBR Last change Comments
 59        C* Write out the header for report 1.                                            5900
   60        C*                                                                               6000
   61        C                   EXCEPT    RECA                                               6100
   62      6 C/EXEC SQL DECLARE C1 CURSOR FOR                                                 6200
   63        C+    SELECT DISTINCT PROJNO, EMPPROJACT.EMPNO,                                  6300
   64        C+           LASTNAME||', '||FIRSTNME, SALARY                                    6400
   65        C+        FROM CORPDATA/EMPPROJACT, CORPDATA/EMPLOYEE                            6500
   66        C+        WHERE EMPPROJACT.EMPNO = EMPLOYEE.EMPNO AND                            6600
   67        C+              COMM >= :COMMI                                                   6700
   68        C+        ORDER BY PROJNO, EMPNO                                                 6800
   69        C/END-EXEC                                                                       6900
   70        C*                                                                               7000
   71      7 C/EXEC SQL                                                                       7100
   72        C+  OPEN C1                                                                      7200
   73        C/END-EXEC                                                                       7300
   74        C*                                                                               7400
   75        C* Fetch and write the rows to QPRINT.                                           7500
   76        C*                                                                               7600
   77      8 C/EXEC SQL WHENEVER NOT FOUND GO TO DONE1                                        7700
   78        C/END-EXEC                                                                       7800
   79        C     SQLCOD        DOUNE     0                                                  7900
   80        C/EXEC SQL                                                                       8000
   81      9 C+   FETCH C1 INTO :PROJNO, :EMPNO, :NAME, :SALARY                               8100
   82        C/END-EXEC                                                                       8200
   83        C                   EXCEPT    RECB                                               8300
   84        C                   END                                                          8400
   85        C     DONE1         TAG                                                          8500
   86        C/EXEC SQL                                                                       8600
   87     10 C+  CLOSE C1                                                                     8700
   88        C/END-EXEC                                                                       8800
   89        C*                                                                               8900
   90        C* For all project ending at a date later than the raise date                    9000
   91        C* (i.e. those projects potentially affected by the salary raises)               9100
   92        C* generate a report containing the project number, project name,                9200
   93        C* the count of employees participating in the project and the                   9300
   94        C* total salary cost of the project.                                             9400
   95        C*                                                                               9500
   96        C* Write out the header for report 2.                                            9600
   97        C*                                                                               9700
   98        C                   EXCEPT    RECC                                               9800
   99        C/EXEC SQL                                                                       9900
  100     11 C+  DECLARE C2 CURSOR FOR                                                       10000
  101        C+    SELECT EMPPROJACT.PROJNO, PROJNAME, COUNT(*),                             10100
  102        C+       SUM((DAYS(EMENDATE) - DAYS(EMSTDATE)) * EMPTIME *                      10200
  103        C+           DECIMAL((SALARY/:WRKDAY),8,2))                                     10300
  104        C+    FROM CORPDATA/EMPPROJACT, CORPDATA/PROJECT, CORPDATA/EMPLOYEE             10400
  105        C+    WHERE EMPPROJACT.PROJNO = PROJECT.PROJNO AND                              10500
  106        C+          EMPPROJACT.EMPNO = EMPLOYEE.EMPNO AND                               10600
  107        C+          PRENDATE > :RDATE                                                   10700
  108        C+    GROUP BY EMPPROJACT.PROJNO, PROJNAME                                      10800
  109        C+    ORDER BY 1                                                                10900
  110        C/END-EXEC                                                                      11000
  111        C*                                                                              11100
  112        C/EXEC SQL OPEN C2                                                              11200
  113        C/END-EXEC                                                                      11300
  114        C*                                                                              11400
  115        C* Fetch and write the rows to QPRINT.                                          11500
  116        C*                                                                              11600
  117        C/EXEC SQL WHENEVER NOT FOUND GO TO DONE2                                       11700
  118        C/END-EXEC                                                                      11800
  119        C     SQLCOD        DOUNE     0                                                 11900
  120        C/EXEC SQL                                                                      
  121     12 C+   FETCH C2 INTO :RPT2                                                        12100
  122        C/END-EXEC                                                                      12200
  123        C                   EXCEPT    RECD                                              12300
 5722ST1 V5R4M0 060210    Create SQL ILE RPG Object         RPGLEEX              08/06/02 16:03:02   Page   4
   124        C                  END                                                         12400
   125        C     DONE2         TAG                                                         12500
   126        C/EXEC SQL CLOSE C2                                                             12600
   127        C/END-EXEC                                                                      12700
   128        C                  RETURN                                                      12800
   129        C*                                                                              12900
   130        C* Error occurred while updating table.  Inform user and rollback                13000
   131        C* changes.                                                                     13100
   132        C*                                                                              13200
   133        C     UPDERR        TAG                                                         13300
   134        C                  EXCEPT    RECE                                              13400
   135     13 C/EXEC SQL WHENEVER SQLERROR CONTINUE                                           13500
   136        C/END-EXEC                                                                      13600
   137        C*                                                                              13700
   138     14 C/EXEC SQL                                                                      13800
   139        C+   ROLLBACK                                                                   13900
   140        C/END-EXEC                                                                      14000
   141        C                  RETURN                                                      14100
   142        C*                                                                              14200
   143        C* Error occurred while generating reports.  Inform user and exit.               14300
   144        C*                                                                              14400
   145        C     RPTERR        TAG                                                         14500
   146        C                  EXCEPT    RECF                                              14600
   147        C*                                                                              14700
   148        C* All done.                                                                    14800
   149        C*                                                                              14900
   150        C     FINISH        TAG                                                         15000
   151        OQPRINT    E            RECA        0  2 01                                     15100
   152        O                                          42 'REPORT OF PROJECTS AFFEC'      15200
   153        O                                          64 'TED BY EMPLOYEE RAISES'        15300
   154        O          E            RECA        0  1                                        15400
   155        O                                           7 'PROJECT'                       15500
   156        O                                          17 'EMPLOYEE'                      15600
   157        O                                          32 'EMPLOYEE NAME'                 15700
   158        O                                          60 'SALARY'                        15800
   159        O          E            RECB        0  1                                        15900
   160        O                       PROJNO              6                                  16000
   161        O                       EMPNO              15                                  16100
   162        O                       NAME               50                                  16200
   163        O                       SALARY        L    61                                  16300
   164        O          E            RECC        2  2                                        16400
   165        O                                          42 'ACCUMULATED STATISTIC'         16500
   166        O                                          54 'S BY PROJECT'                  16600
   167        O          E            RECC        0  1                                        16700
   168        O                                           7 'PROJECT'                       16800
   169        O                                          56 'NUMBER OF'                     16900
   170        O                                          67 'TOTAL'                         17000
   171        O          E            RECC        0  2                                        17100
   172        O                                           6 'NUMBER'                        17200
   173        O                                          21 'PROJECT NAME'                  17300
   174        O                                          56 'EMPLOYEES'                     17400
   175        O                                          66 'COST'                          17500
   176        O          E            RECD        0  1                                        17600
   177        O                       PRJNUM              6                                  17700
   178        O                       PNAME              45                                  17800
   179        O                       EMPCNT        L    54                                  17900
   180        O                       PRCOST        L    70                                  18000
   181        O          E            RECE        0  1                                        18100
   182        O                                          28 '*** ERROR Occurred while'      18200
   183        O                                          52 ' updating table. SQLCODE'      18300
   184        O                                          53 '='                             18400
   185        O                       SQLCOD        L    62                                  18500
   186        O          E            RECF        0  1                                        18600
   187        O                                          28 '*** ERROR Occurred while'      18700
   188        O                                          52 ' generating reports. SQL'      18800
   189        O                                          57 'CODE='                         18900
   190        O                       SQLCOD        L    67                                  19000
                                   * * * * *  E N D  O F  S O U R C E  * * * * *
```

```
5722ST1 V5R4M0 060210        Create SQL ILE RPG Object         RPGLEEX           08/06/02 16:03:02   Page   5
CROSS REFERENCE
Data Names                    Define    Reference
ACTNO                            62       SMALL INTEGER PRECISION(4,0) COLUMN (NOT NULL) IN CORPDATA.EMPPROJACT
BIRTHDATE                        42       DATE(10) COLUMN IN CORPDATA.EMPLOYEE
BONUS                            42       DECIMAL(9,2) COLUMN IN CORPDATA.EMPLOYEE
COMM                             ****     COLUMN
                                          42 62
COMM                             42       DECIMAL(9,2) COLUMN IN CORPDATA.EMPLOYEE
COMMI                            25       DECIMAL(7,2)
                                          42 62
CORPDATA                         ****     COLLECTION
                                          42 62 62 99 99 99
C1                               62       CURSOR
                                          71 80 86
C2                               99       CURSOR
                                          112 120 126
DEPTNO                           8        CHARACTER(3) IN RPT1
DEPTNO                           99       CHARACTER(3) COLUMN (NOT NULL) IN CORPDATA.PROJECT
DONE1                            85
DONE1                            ****     LABEL
                                          77
DONE2                            125
DONE2                            ****     LABEL
                                          117
EDLEVEL                          42       SMALL INTEGER PRECISION(4,0) COLUMN (NOT NULL) IN CORPDATA.EMPLOYEE
EMENDATE                         62       DATE(10) COLUMN IN CORPDATA.EMPPROJACT
EMENDATE                         ****     COLUMN
                                          99
EMPCNT                           20       SMALL INTEGER PRECISION(4,0) IN RPT2
EMPLOYEE                         ****     TABLE IN CORPDATA
                                          42 62 99
EMPLOYEE                         ****     TABLE
                                          62 99
EMPNO                            11       CHARACTER(6) DBCS-open
                                          80
EMPNO                            42       CHARACTER(6) COLUMN (NOT NULL) IN CORPDATA.EMPLOYEE
EMPNO                            ****     COLUMN IN EMPPROJACT
                                          62 62 62 99
EMPNO                            ****     COLUMN IN EMPLOYEE
                                          62 99
EMPNO                            62       CHARACTER(6) COLUMN (NOT NULL) IN CORPDATA.EMPPROJACT
EMPPROJACT                       ****     TABLE
                                          62 62 99 99 99 99
EMPPROJACT                       ****     TABLE IN CORPDATA
                                          62 99
EMPTIME                          62       DECIMAL(5,2) COLUMN IN CORPDATA.EMPPROJACT
EMPTIME                          ****     COLUMN
                                          99
EMSTDATE                         62       DATE(10) COLUMN IN CORPDATA.EMPPROJACT
EMSTDATE                         ****     COLUMN
                                          99
FINISH                           150
FIRSTNME                         42       VARCHAR(12) COLUMN (NOT NULL) IN CORPDATA.EMPLOYEE
FIRSTNME                         ****     COLUMN
                                          62
HIREDATE                         42       DATE(10) COLUMN IN CORPDATA.EMPLOYEE
JOB                              42       CHARACTER(8) COLUMN IN CORPDATA.EMPLOYEE
LASTNAME                         42       VARCHAR(15) COLUMN (NOT NULL) IN CORPDATA.EMPLOYEE
LASTNAME                         ****     COLUMN
                                          62
MAJPROJ                          8        CHARACTER(6) IN RPT1
MAJPROJ                          99       CHARACTER(6) COLUMN IN CORPDATA.PROJECT
MIDINIT                          42       CHARACTER(1) COLUMN (NOT NULL) IN CORPDATA.EMPLOYEE
NAME                             12       CHARACTER(30) DBCS-open
                                          80
PERCNT                           27       DECIMAL(7,2)
                                          42
PHONENO                          42       CHARACTER(4) COLUMN IN CORPDATA.EMPLOYEE
PNAME                            19       CHARACTER(36) DBCS-open IN RPT2
PRCOST                           21       DECIMAL(9,2) IN RPT2
PRENDATE                         8        DATE(8) IN RPT1
```

```
5722ST1 V5R4M0 060210    Create SQL ILE RPG Object         RPGLEEX             08/06/02 16:03:02   Page   6
PRENDATE                         ****     COLUMN
                                          99
PRENDATE                         99       DATE(10) COLUMN IN CORPDATA.PROJECT
PRJNUM                           18       CHARACTER(6) DBCS-open IN RPT2
CROSS REFERENCE
PROJECT                          ****     TABLE IN CORPDATA
                                          99
PROJECT                          ****     TABLE
                                          99
PROJNAME                         8        VARCHAR(24) IN RPT1
PROJNAME                         ****     COLUMN
                                          99 99
PROJNAME                         99       VARCHAR(24) COLUMN (NOT NULL) IN CORPDATA.PROJECT
PROJNO                           8        CHARACTER(6) IN RPT1
                                          80
PROJNO                           ****     COLUMN
                                          62 62
PROJNO                           62       CHARACTER(6) COLUMN (NOT NULL) IN CORPDATA.EMPPROJACT
PROJNO                           ****     COLUMN IN EMPPROJACT
                                          99 99 99
PROJNO                           ****     COLUMN IN PROJECT
                                          99
PROJNO                           99       CHARACTER(6) COLUMN (NOT NULL) IN CORPDATA.PROJECT
PRSTAFF                          8        DECIMAL(5,2) IN RPT1
PRSTAFF                          99       DECIMAL(5,2) COLUMN IN CORPDATA.PROJECT
PRSTDATE                         8        DATE(8) IN RPT1
PRSTDATE                         99       DATE(10) COLUMN IN CORPDATA.PROJECT
RDATE                            26       CHARACTER(10) DBCS-open
                                          99
RESPEMP                          8        CHARACTER(6) IN RPT1
RESPEMP                          99       CHARACTER(6) COLUMN (NOT NULL) IN CORPDATA.PROJECT
RPTERR                           145
RPTERR                           ****     LABEL
                                          53
RPT1                             8        STRUCTURE
RPT2                             17       STRUCTURE
                                          120
SALARY                           13       DECIMAL(9,2)
                                          80
SALARY                           ****     COLUMN
                                          42 42 62 99
SALARY                           42       DECIMAL(9,2) COLUMN IN CORPDATA.EMPLOYEE
SEX                              42       CHARACTER(1) COLUMN IN CORPDATA.EMPLOYEE
UPDERR                           133
UPDERR                           ****     LABEL
                                          39
WORKDEPT                         42       CHARACTER(3) COLUMN IN CORPDATA.EMPLOYEE
WRKDAY                           24       SMALL INTEGER PRECISION(4,0)
                                          99
No errors found in source
   190 Source records processed
                               * * * * *  E N D  O F  L I S T I N G  * * * * *
```

Related concepts

"Coding SQL statements in ILE RPG applications" on page 91
You need to be aware of the unique application and coding requirements for embedding SQL statements in an ILE RPG program. In this topic, the coding requirements for host variables are defined.

Example: SQL statements in REXX programs

This example program is written in the REXX programming language.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 176.

```
Record  *...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
         /*********************************************************************/
         /* A sample program which updates the salaries for those employees    */
         /* whose current commission total is greater than or equal to the    */
         /* value of COMMISSION. The salaries of those who qualify are        */
         /* increased by the value of PERCENTAGE, retroactive to RAISE_DATE.  */
         /* A report is generated and dumped to the display which shows the   */
         /* projects which these employees have contributed to, ordered by    */
         /* project number and employee ID. A second report shows each        */
         /* project having an end date occurring after RAISE DATE (i.e. is    */
         /* potentially affected by the retroactive raises) with its total    */
         /* salary expenses and a count of employees who contributed to the   */
         /* project.                                                          */
         /*********************************************************************/


         /* Initialize RC variable */
         RC = 0

         /* Initialize HV for program usage */
         COMMISSION = 2000.00;
         PERCENTAGE = 1.04;
         RAISE_DATE = '1982-06-01';
         WORK_DAYS  = 253;

         /* Create the output file to dump the 2 reports. Perform an OVRDBF   */
         /* to allow us to use the SAY REXX command to write to the output    */
         /* file.                                                             */
         ADDRESS '*COMMAND',
                'DLTF FILE(CORPDATA/REPORTFILE)'
         ADDRESS '*COMMAND',
                'CRTPF FILE(CORPDATA/REPORTFILE) RCDLEN(80)'
         ADDRESS '*COMMAND',
                'OVRDBF FILE(STDOUT) TOFILE(CORPDATA/REPORTFILE) MBR(REPORTFILE)'

         /* Update the selected employee's salaries by the new percentage. */
         /* If an error occurs during the update, ROLLBACK the changes.    */
         3SIGNAL ON ERROR
         ERRLOC = 'UPDATE_ERROR'
         UPDATE_STMT = 'UPDATE CORPDATA/EMPLOYEE ',
                       'SET SALARY = SALARY * ?  ',
                       'WHERE COMM >= ?          '
         EXECSQL,
                'PREPARE S1 FROM :UPDATE_STMT'
         4EXECSQL,
                'EXECUTE S1 USING :PERCENTAGE,',
                '                 :COMMISSION '
         /* Commit changes */
         5EXECSQL,
                'COMMIT'
         ERRLOC = 'REPORT_ERROR'

```

Figure 8. Sample REXX Procedure Using SQL Statements

```
Record  *...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
         /* Report the updated statistics for each project supported by one */
         /* of the selected employees.                                      */

         /* Write out the header for Report 1 */
         SAY '   '
         SAY '   '
         SAY '   '
         SAY '         REPORT OF PROJECTS AFFECTED BY EMPLOYEE RAISES'
         SAY '   '
         SAY 'PROJECT  EMPID     EMPLOYEE NAME                      SALARY'
         SAY '-------  -----     -------------                      ------'
         SAY '   '

         SELECT_STMT =  'SELECT DISTINCT PROJNO, EMPPROJACT.EMPNO, ',
                        '       LASTNAME||'', ''||FIRSTNME, SALARY ',
                        'FROM CORPDATA/EMPPROJACT, CORPDATA/EMPLOYEE  ',
                        'WHERE EMPPROJACT.EMPNO = EMPLOYEE.EMPNO AND  ',
                        '      COMM >= ?                             ',
                        'ORDER BY PROJNO, EMPNO                     '
         EXECSQL,
                'PREPARE S2 FROM :SELECT_STMT'
         6EXECSQL,
                'DECLARE C1 CURSOR FOR S2'
         7EXECSQL,
                'OPEN C1 USING :COMMISSION'

         /* Handle the FETCH errors and warnings inline */
         SIGNAL OFF ERROR

         /* Fetch all of the rows */
         DO UNTIL (SQLCODE <> 0)
           9EXECSQL,
                  'FETCH C1 INTO :RPT1.PROJNO, :RPT1.EMPNO,',
                  '              :RPT1.NAME, :RPT1.SALARY  '

           /* Process any errors that may have occurred. Continue so that   */
           /* we close the cursor for any warnings.                         */
           IF SQLCODE < 0 THEN
             SIGNAL ERROR

           /* Stop the loop when we hit the EOF. Don't try to print out the */
           /* fetched values.                                               */
          8IF SQLCODE = 100 THEN
             LEAVE

           /* Print out the fetched row */
           SAY RPT1.PROJNO '   ' RPT1.EMPNO '   ' RPT1.NAME '     ' RPT1.SALARY
         END;

         10EXECSQL,
                 'CLOSE C1'

         /* For all projects ending at a date later than 'raise_date'               */
         /* (that is, those projects potentially affected by the salary raises)    */
         /* generate a report containing the project number, project name,          */
         /* the count of employees participating in the project, and the            */
         /* total salary cost of the project.                                       */

```

```
Record  *...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
           /* Write out the header for Report 2 */
         SAY '   '
         SAY '   '
         SAY '   '
         SAY '         ACCUMULATED STATISTICS BY PROJECT'
         SAY '   '
         SAY 'PROJECT  PROJECT NAME                          NUMBER OF      TOTAL'
         SAY 'NUMBER                                         EMPLOYEES      COST'
         SAY '-------  ------------                          ---------      -----'
         SAY '   '


         /* Go to the common error handler */
         SIGNAL ON ERROR

         SELECT_STMT = 'SELECT EMPPROJACT.PROJNO, PROJNAME, COUNT(*),                ',
                       '   SUM( (DAYS(EMENDATE) - DAYS(EMSTDATE)) * EMPTIME *     ',
                       '         DECIMAL(( SALARY / ? ),8,2) )                    ',
                       'FROM CORPDATA/EMPPROJACT, CORPDATA/PROJECT, CORPDATA/EMPLOYEE',
                       'WHERE EMPPROJACT.PROJNO = PROJECT.PROJNO  AND                ',
                       '      EMPPROJACT.EMPNO = EMPLOYEE.EMPNO  AND                 ',
                       '      PRENDATE > ?                                ',
                       'GROUP BY EMPPROJACT.PROJNO, PROJNAME                         ',
                       'ORDER BY 1                                              '
         EXECSQL,
                'PREPARE S3 FROM :SELECT_STMT'
         11EXECSQL,
                'DECLARE C2 CURSOR FOR S3'
         EXECSQL,
                'OPEN C2 USING :WORK_DAYS, :RAISE_DATE'

         /* Handle the FETCH errors and warnings inline */
         SIGNAL OFF ERROR

         /* Fetch all of the rows */
         DO UNTIL (SQLCODE <> 0)
           12EXECSQL,
                  'FETCH C2 INTO :RPT2.PROJNO, :RPT2.PROJNAME,    ',
                  '              :RPT2.EMPCOUNT, :RPT2.TOTAL_COST '

           /* Process any errors that may have occurred. Continue so that   */
           /* we close the cursor for any warnings.                         */
           IF SQLCODE < 0 THEN
             SIGNAL ERROR

           /* Stop the loop when we hit the EOF. Don't try to print out the */
           /* fetched values.                                               */
           IF SQLCODE = 100 THEN
             LEAVE

           /* Print out the fetched row */
           SAY RPT2.PROJNO '   ' RPT2.PROJNAME '  ' ,
               RPT2.EMPCOUNT '        ' RPT2.TOTAL_COST
         END;

         EXECSQL,
                'CLOSE C2'

```

```
/* Delete the OVRDBF so that we will continue writing to the output  */
/* display.                                                          */
ADDRESS '*COMMAND',
       'DLTOVR FILE(STDOUT)'

/* Leave procedure with a successful or warning RC */
EXIT RC


/* Error occurred while updating the table or generating the         */
/* reports. If the error occurred on the UPDATE, rollback all of     */
/* the changes. If it occurred on the report generation, display the */
/* REXX RC variable and the SQLCODE and exit the procedure.          */
ERROR:

  13SIGNAL OFF ERROR

  /* Determine the error location */
  SELECT
    /* When the error occurred on the UPDATE statement */
    WHEN ERRLOC = 'UPDATE_ERROR' THEN
      DO
        SAY '*** ERROR Occurred while updating table.',
            'SQLCODE = ' SQLCODE
        14EXECSQL,
                'ROLLBACK'
      END
    /* When the error occurred during the report generation */
    WHEN ERRLOC = 'REPORT_ERROR' THEN
      SAY '*** ERROR Occurred while generating reports. ',
          'SQLCODE = ' SQLCODE
    OTHERWISE
      SAY '*** Application procedure logic error occurred '
  END

  /* Delete the OVRDBF so that we will continue writing to the       */
  /* output display.                                                 */
  ADDRESS '*COMMAND',
       'DLTOVR FILE(STDOUT)'

  /* Return the error RC received from SQL. */
  EXIT RC
                       * * * * *  E N D  O F  S O U R C E  * * * * *
```

Related concepts

"Coding SQL statements in REXX applications" on page 114
REXX procedures do not have to be preprocessed. At run time, the REXX interpreter passes statements that it does not understand to the current active command environment for processing.

Report produced by example programs that use SQL

This report is produced by each of the example programs.

```
                       REPORT OF PROJECTS AFFECTED BY RAISES

PROJECT  EMPID     EMPLOYEE NAME                      SALARY

AD3100   000010    HAAS, CHRISTINE                  54860.00
AD3110   000070    PULASKI, EVA                     37616.80
AD3111   000240    MARINO, SALVATORE                29910.40
AD3113   000270    PEREZ, MARIA                     28475.20
IF1000   000030    KWAN, SALLY                      39780.00
IF1000   000140    NICHOLLS, HEATHER                29556.80
IF2000   000030    KWAN, SALLY                      39780.00
IF2000   000140    NICHOLLS, HEATHER                29556.80
MA2100   000010    HAAS, CHRISTINE                  54860.00
MA2100   000110    LUCCHESSI, VICENZO               48360.00
MA2110   000010    HAAS, CHRISTINE                  54860.00
MA2111   000200    BROWN, DAVID                     28849.60
MA2111   000220    LUTZ, JENNIFER                   31033.60
MA2112   000150    ADAMSON, BRUCE                   26291.20
```

```
OP1000   000050    GEYER, JOHN                       41782.00
OP1010   000090    HENDERSON, EILEEN                 30940.00
OP1010   000280    SCHNEIDER, ETHEL                  27300.00
OP2010   000050    GEYER, JOHN                       41782.00
OP2010   000100    SPENSER, THEODORE                 27196.00
OP2012   000330    LEE, WING                         26384.80
PL2100   000020    THOMPSON, MICHAEL                 42900.00


                     ACCUMULATED STATISTICS BY PROJECT

PROJECT                                        NUMBER OF       TOTAL
NUMBER   PROJECT NAME                          EMPLOYEES       COST

AD3100   ADMIN SERVICES                             1         19623.11
AD3110   GENERAL ADMIN SYSTEMS                      1         58877.28
AD3111   PAYROLL PROGRAMMING                        7         66407.56
AD3112   PERSONNEL PROGRAMMING                      9         28845.70
AD3113   ACCOUNT PROGRAMMING                       14         72114.52
IF1000   QUERY SERVICES                             4         35178.99
IF2000   USER EDUCATION                             5         55212.61
MA2100   WELD LINE AUTOMATION                       2        114001.52
MA2110   W L PROGRAMMING                            1         85864.68
MA2111   W L PROGRAM DESIGN                         3         93729.24
MA2112   W L ROBOT DESIGN                           6        166945.84
MA2113   W L PROD CONT PROGS                        5         71509.11
OP1000   OPERATION SUPPORT                          1         16348.86
OP1010   OPERATION                                  5        167828.76
OP2010   SYSTEMS SUPPORT                            2         91612.62
OP2011   SCP SYSTEMS SUPPORT                        2         31224.60
OP2012   APPLICATIONS SUPPORT                       2         41294.88
OP2013   DB/DC SUPPORT                              2         37311.12
PL2100   WELD LINE PLANNING                         1         43576.92
```

CL command descriptions for host language precompilers

The DB2 UDB for iSeries database provides commands for precompiling programs coded in these programming languages.

Related concepts

"Non-ILE SQL precompiler commands" on page 129
The DB2 UDB Query Manager and SQL Development Kit licensed program includes non-ILE precompiler commands for the following host languages: CRTSQLCBL (for OPM COBOL), CRTSQLPLI (for PL/I PRPQ), and CRTSQLRPG (for RPG III, which is part of RPG/400).

Related reference

"ILE SQL precompiler commands" on page 130
In the DB2 UDB Query Manager and SQL Development Kit licensed program, these ILE precompiler commands exist: CRTSQLCI, CRTSQLCPPI, CRTSQLCBLI, and CRTSQLRPGI.

CRTSQLCBL (Create Structured Query Language COBOL) command

The Create Structured Query Language COBOL (CRTSQLCBL) command calls the Structured Query Language (SQL) precompiler.

It precompiles COBOL source containing SQL statements, produces a temporary source member, and then optionally calls the COBOL compiler to compile the program.

Related reference

Create Structured Query Language COBOL (CRTSQLCBL) command

CRTSQLCBLI (Create SQL ILE COBOL Object) command

The Create Structured Query Language ILE COBOL Object (CRTSQLCBLI) command calls the Structured Query Language (SQL) precompiler, which precompiles COBOL source containing SQL statements, produces a temporary source member, and then optionally calls the ILE COBOL compiler to create a module, a program, or a service program.

Related reference

Create Structured Query Language ILE COBOL Object (CRTSQLCBLI) command

CRTSQLCI (Create Structured Query Language ILE C Object) command

The Create Structured Query Language ILE C Object (CRTSQLCI) command calls the Structured Query Language (SQL) precompiler, which precompiles C source containing SQL statements, produces a temporary source member, and then optionally calls the ILE C compiler to create a module, create a program, or create a service program.

Related reference

Create Structured Query Language ILE C Object (CRTSQLCI) command

CRTSQLCPPI (Create Structured Query Language C++ Object) command

The Create Structured Query Language C++ Object (CRTSQLCPPI) command calls the Structured Query Language (SQL) precompiler, which precompiles C++ source containing SQL statements, produces a temporary source member, and then optionally calls the C++ compiler to create a module.

Related reference

Create Structured Query Language C++ Object (CRTSQLCPPI) command

CRTSQLPLI (Create Structured Query Language PL/I) command

The Create Structured Query Language PL/I (CRTSQLPLI) command calls a Structured Query Language (SQL) precompiler, which precompiles PL/I source containing SQL statements, produces a temporary source member, and optionally calls the PL/I compiler to compile the program.

Related reference

Create Structured Query Language PL/I (CRTSQLPLI) command

CRTSQLRPG (Create Structured Query Language RPG) command

The Create Structured Query Language RPG (CRTSQLRPG) command calls the Structured Query Language (SQL) precompiler, which precompiles the RPG source containing the SQL statements, produces a temporary source member, and then optionally calls the RPG compiler to compile the program.

Related reference

Create Structured Query Language RPG (CRTSQLRPG) command

Related information for embedded SQL programming

Listed here are the product manuals and information center topics that relate to the Embedded SQL programming topic collection. You can view or print any of the PDFs.

Manuals

- COBOL/400 User's Guide
- COBOL/400 Reference
- RPG/400 User's Guide

- RPG/400 Reference
- ILE RPG Programmer's Guide
- ILE RPG Reference
- ILE COBOL Programmer's Guide (5661 KB)
- ILE COBOL Reference (6630 KB)
- REXX/400 Programmer's Guide (854 KB)
- REXX/400 Reference (515 KB)
- SQL reference PDF (13 343 KB)

Other information

You can view or download these related topics:

- SQL programming
- Performance and query optimization
- SQL call level interface

Saving PDF files

To save a PDF on your workstation for viewing or printing:

1. Right-click the PDF in your browser (right-click the link above).
2. Click the option that saves the PDF locally.
3. Navigate to the directory in which you want to save the PDF.
4. Click **Save**.

Downloading Adobe Reader

You need Adobe Reader installed on your system to view or print these PDFs. You can download a free copy from the Adobe Web site (www.adobe.com/products/acrobat/readstep.html) .

Code license and disclaimer information

IBM grants you a nonexclusive copyright license to use all programming code examples from which you can generate similar function tailored to your own specific needs.

SUBJECT TO ANY STATUTORY WARRANTIES WHICH CANNOT BE EXCLUDED, IBM, ITS PROGRAM DEVELOPERS AND SUPPLIERS MAKE NO WARRANTIES OR CONDITIONS EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OR CONDITIONS OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT, REGARDING THE PROGRAM OR TECHNICAL SUPPORT, IF ANY.

UNDER NO CIRCUMSTANCES IS IBM, ITS PROGRAM DEVELOPERS OR SUPPLIERS LIABLE FOR ANY OF THE FOLLOWING, EVEN IF INFORMED OF THEIR POSSIBILITY:

1. LOSS OF, OR DAMAGE TO, DATA;
2. DIRECT, SPECIAL, INCIDENTAL, OR INDIRECT DAMAGES, OR FOR ANY ECONOMIC CONSEQUENTIAL DAMAGES; OR
3. LOST PROFITS, BUSINESS, REVENUE, GOODWILL, OR ANTICIPATED SAVINGS.

SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF DIRECT, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, SO SOME OR ALL OF THE ABOVE LIMITATIONS OR EXCLUSIONS MAY NOT APPLY TO YOU.

Appendix. 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 grant 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 106-0032, 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.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

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

Software Interoperability Coordinator, Department YBWA
3605 Highway 52 N
Rochester, MN 55901
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, IBM License Agreement for Machine Code, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

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 illustrate 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.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. _enter the year or years_. All rights reserved.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Programming Interface Information

This Embedded SQL programming publication documents intended Programming Interfaces that allow the customer to write programs to obtain the services of IBM i5/OS.

Trademarks

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

COBOL/400
DB2
DB2 Universal Database
Distributed Relational Database Architecture
DRDA
i5/OS
IBM
IBM (logo)
Integrated Language Environment
iSeries
RPG/400
System i

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

Terms and conditions

Permissions for the use of these publications is granted subject to the following terms and conditions.

Personal Use: You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative works of these publications, or any portion thereof, without the express consent of IBM.

Commercial Use: You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of IBM.

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any information, data, software or other intellectual property contained therein.

IBM reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by IBM, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED ″AS-IS″ AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

Printed in USA