eserver

iSeries

DB2 Universal Database for iSeries SQL Programming with Host Languages

Version 5

eserver

iSeries

DB2 Universal Database for iSeries SQL Programming with Host Languages

Version 5

Contents

Chapter 4. Coding SQL Statements in PL/I Applications 71

Chapter 5. Coding SQL Statements in RPG for iSeries Applications 89

Chapter 6. Coding SQL Statements in ILE RPG for iSeries Applications . . . 103

Chapter 7. Coding SQL Statements in REXX Applications 123

Chapter 8. Preparing and Running a Program with SQL Statements 131

Appendix A. Sample Programs Using DB2 UDB for iSeries Statements . . . 147

About DB2 UDB for iSeries SQL Programming with Host Languages

This book explains to programmers and database administrators how to create database applications in host languages that use DB2 UDB for iSeries SQL statements and functions.

For more information about DB2 UDB for iSeries SQL guidelines and examples for implementation in an application programming environment, see the following books in the **Database** category of the Information Center:
- SQL Reference
- SQL Programming Concepts
- Database Performance and Query Optimization
- SQL Call Level Interface (ODBC)

For more infomration about this guide, see the following sections:
- "Who should read the SQL Programming with Host Languages book"
- "Assumptions relating to examples of SQL statements in the SQL Programming with Host Languages book"
- "How to interpret syntax diagrams in the SQL Programming with Host Languages book" on page viii
- "What's new for Version 5 Release 2 in the SQL Programming with Host Languages book" on page x

Who should read the SQL Programming with Host Languages book

This book should be used by application programmers and database administrators who are familiar with and can program with COBOL for iSeries, ILE COBOL for iSeries, iSeries PL/I, ILE C for iSeries, ILE C++, REXX, RPG III (part of RPG for iSeries), or ILE RPG for iSeries language and who can understand basic database applications.

Assumptions relating to examples of SQL statements in the SQL Programming with Host Languages book

The examples of SQL statements shown in this guide are based on the sample tables, found in Appendix A, "DB2 UDB for iSeries Sample Tables," of the SQL Programming Concepts book found in the iSeries Information Center and assume the following:
- They are shown in the interactive SQL environment or they are written in ILE C or in COBOL. EXEC SQL and END-EXEC are used to delimit an SQL statement in a COBOL program. A description of how to use SQL statements in a COBOL program is provided in Chapter 3, "Coding SQL Statements in COBOL Applications" on page 43. A description of how to use SQL statements in an ILE C program is provided in Chapter 2, "Coding SQL Statements in C and C++ Applications" on page 11.
- Each SQL example is shown on several lines, with each clause of the statement on a separate line.
- SQL keywords are highlighted.

- Table names provided in Sample Tables use the collection CORPDATA. Table names that are not found in these sample tables should use collections you create. See Appendix A, ″DB2 UDB for iSeries Sample Tables,″ of the SQL Programming Concepts book for a definition of these tables and how to create them.
- Calculated columns are enclosed in parentheses, (), and brackets, [].
- The SQL naming convention is used.
- The APOST and APOSTSQL precompiler options are assumed although they are not the default options in COBOL. Character string literals within SQL and host language statements are delimited by apostrophes (′).
- A sort sequence of *HEX is used, unless otherwise noted.
- The complete syntax of the SQL statement is usually not shown in any one example. For the complete description and syntax of any of the statements described in this guide, see the SQL Reference book.

Whenever the examples vary from these assumptions, it is stated.

Because this guide is for the application programmer, most of the examples are shown as if they were written in an application program. However, many examples can be slightly changed and run interactively by using interactive SQL. The syntax of an SQL statement, when using interactive SQL, differs slightly from the format of the same statement when it is embedded in a program.

See “Code disclaimer information” for more information about using program examples.

Code disclaimer information

This document contains programming examples.

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.

All sample code is provided by IBM for illustrative purposes only. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

All programs contained herein are provided to you ″AS IS″ without any warranties of any kind. The implied warranties of non-infringement, merchantability and fitness for a particular purpose are expressly disclaimed.

How to interpret syntax diagrams in the SQL Programming with Host Languages book

Throughout this book, syntax is described using the structure defined as follows:

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

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

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

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

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

Diagrams of syntactical units other than complete statements start with the ►─── symbol and end with the ───► symbol.

- Required items appear on the horizontal line (the main path).

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

- Optional items appear below the main path.

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

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

```
                   ┌─optional_item─┐
►►──required_item──┴───────────────┴─────────────────────────────────►◄
```

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

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

```
►►──required_item──┬─required_choice1─┬──────────────────────────────►◄
                   └─required_choice2─┘
```

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

```
►►──required_item──┬──────────────────┬──────────────────────────────►◄
                   ├─optional_choice1─┤
                   └─optional_choice2─┘
```

If one of the items is the default, it will appear above the main path and the remaining choices will be shown below.

```
                   ┌─default_choice──┐
►►──required_item──┼─────────────────┼───────────────────────────────►◄
                   ├─optional_choice─┤
                   └─optional_choice─┘
```

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

```
                   ┌─────────────────┐
                   ▼                 │
►►──required_item────repeatable_item─┴───────────────────────────────►◄
```

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

```
                   ┌─,───────────────┐
                   ▼                 │
►►──required_item────repeatable_item─┴───────────────────────────────►◄
```

A repeat arrow above a stack indicates that you can repeat the items in the stack.

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

What's new for Version 5 Release 2 in the SQL Programming with Host Languages book

New host variable type ROWID for C, C++, COBOL, ILE COBOL, ILE RPG, and PL/I. See the following:

- "ROWID host variables in C and C++ applications that use SQL" on page 27
- "ROWID host variables in COBOL applications that use SQL" on page 56
- "ROWID host variables in PL/I applications that use SQL" on page 79
- "ROWID variables in ILE RPG for iSeries applications that use SQL" on page 113

SQL VARCHAR type for C and C++. See "Character host variables in C and C++ applications that use SQL" on page 18 for more information.

Chapter 1. Common concepts and rules for using SQL with Host Languages

This chapter describes some concepts and rules that are common to using SQL statements in a host language that involve:

- Using host variables in SQL statements
- Handling SQL error and return codes
- Handling exception conditions with the WHENEVER statement

Note: See "Code disclaimer information" on page viii information for information pertaining to code examples.

Writing applications that use SQL

You can create database applications in host languages that use DB2 UDB for iSeries SQL statements and functions. Select the following for more information about application requirements and coding requirements for each of the host languages:

- Chapter 2, "Coding SQL Statements in C and C++ Applications" on page 11
- Chapter 3, "Coding SQL Statements in COBOL Applications" on page 43
- Chapter 4, "Coding SQL Statements in PL/I Applications" on page 71
- Chapter 5, "Coding SQL Statements in RPG for iSeries Applications" on page 89
- Chapter 6, "Coding SQL Statements in ILE RPG for iSeries Applications" on page 103
- Chapter 7, "Coding SQL Statements in REXX Applications" on page 123
- Chapter 8, "Preparing and Running a Program with SQL Statements" on page 131

Note: For information about using Java™ as a host language, see the IBM Developer Kit for Java.

Using host variables in SQL statements

When your program retrieves data, the values are put into data items defined by your program and 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 be data type compatible. Host variables may not 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 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 it needs 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:

1. **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.
```
2. **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.
3. **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

4. **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

For more information about these statements, see the SQL Reference book.

For more information about using host variables, see the following sections:
- "Assignment rules for host variables in SQL statements"
- "Indicator variables in applications that use SQL" on page 6

Assignment rules for host variables in SQL statements

SQL values are set to (or assigned to) host variables during the running of FETCH, SELECT INTO, SET, and VALUES INTO statements. SQL values are set from (or assigned from) host variables during the running of INSERT, UPDATE, and CALL statements. All assignment operations observe the following rules:

- Numbers and strings are not compatible:

  Numbers cannot be assigned to string columns or string host variables.

  Strings cannot be assigned to numeric columns or numeric host variables.
- All character and DBCS graphic strings are compatible with UCS-2 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 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 parameters if conversion is supported.
- 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 [1]. The insert or update value of a date column must be a date or a string representation of a date.

  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.

Rules for string assignment of host variables in SQL statements

Rules regarding character string assignment are:

- When a 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

1. 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. See DECLARE VARIABLE in the SQL Reference book.

included in the length of the string. However, for string assignment trailing blanks 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 SQLCA) are set to W.
- When the value of a result column is assigned to a fixed-length host variable or when the value of a host variable is assigned to a fixed-length CHAR 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 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.

For more information about CCSIDs, see the Globalization topic in the Information Center.

Rules for numeric assignment of host variables in SQL statements

Rules regarding numeric assignment are:

- **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 binary number** is assigned to a decimal, numeric, or binary 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 a **binary 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

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 32767, the value that is stored in the indicator variable is 32767, since 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.

Indicator variables used with host structures

You can also 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 3 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 above example, SQL selects the column values of the row 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

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. For more information, see the SQL Reference book.

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.

For more information about the SQLCA, see Appendix B, “SQL Communication Area” in the SQL Reference book. To find a specific SQLCODE or SQLSTATE, use the SQL Message finder. For a listing of DB2 UDB for iSeries SQLCODEs and SQLSTATEs, see SQL Messages and Codes in the iSeries Information Center.

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. For information on handling exception conditions in REXX, see Chapter 7, “Coding SQL Statements in REXX Applications”.

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.

Chapter 2. Coding SQL Statements in C and C++ Applications

This chapter describes the unique application and coding requirements for embedding SQL statements in a C or C++ program. C program refers to ILE C for iSeries programs. C++ program refers to ILE C++ programs. This chapter also defines the requirements for host structures and host variables. For more details, see the following sections:

For a detailed sample C program that shows how SQL statements can be used, see Appendix A, "Sample Programs Using DB2 UDB for iSeries Statements".

Note: See "Code disclaimer information" on page viii information for information pertaining to code examples.

Defining the SQL Communications Area in C and C++ applications that use SQL

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 executed. 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. 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 static 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;
```

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.

For more information about SQLCA, see Appendix B, SQL Communication Area in the SQL Reference book.

Defining SQL Descriptor Areas in C and C++ applications that use SQL

The following statements require an SQLDA:

EXECUTE...USING DESCRIPTOR *descriptor-name*

FETCH...USING DESCRIPTOR *descriptor-name*

OPEN...USING DESCRIPTOR *descriptor-name*

DESCRIBE *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. 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 newvalue.

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 datatype.

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 described in Dynamic SQL Applications in the *DB2® UDB for iSeries Programming Concepts* information. 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.

For more information about the SQLDA, see the topic ″SQL Descriptor Area″ in the SQL Reference book.

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

An SQL statement can be placed wherever a C or C++ statement that can be run can be placed.

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 ;
```

See the following sections for more details:

- "Comments in C and C++ applications that use SQL"
- "Continuation for SQL statements in C and C++ applications that use SQL"
- "Including code in C and C++ applications that use SQL" on page 16
- "Margins in C and C++ applications that use SQL" on page 16
- "Names in C and C++ applications that use SQL" on page 16
- "NULLs and NULs in C and C++ applications that use SQL" on page 16
- "Statement labels in C and C++ applications that use SQL" on page 16
- "Preprocessor sequence for C and C++ applications that use SQL" on page 16
- "Trigraphs in C and C++ applications that use SQL" on page 16
- "WHENEVER Statement in C and C++ applications that use SQL" on page 17

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 on 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. For more information about CRTSQLCI or CRTSQLCPPI, see Appendix B, "DB2 UDB for iSeries CL Command Descriptions for Host Language Precompilers".

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 the following 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.

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 nonnull 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. A host variable used in an SQL statement must be declared prior to the first use of the host variable in an SQL statement.

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.

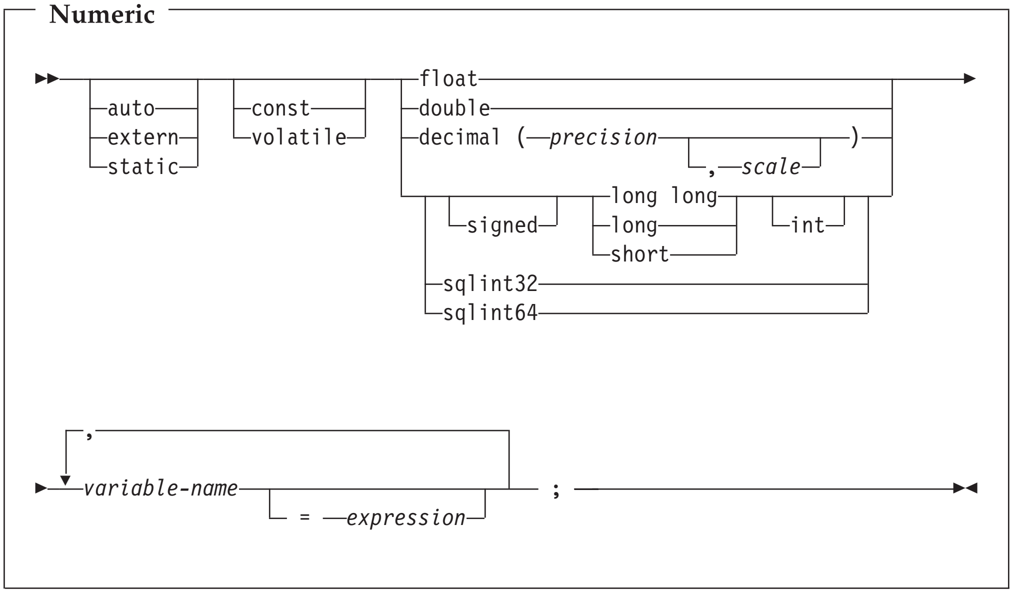
For more information, see “Declaring host variables in C and C++ applications that use SQL”.

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

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



Notes:

1. Precision and scale must be integer constants. Precision may be in the range from 1 to 31. 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:

- 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

```
>>-+--------+--+----------+--+----------+--char------------------->
   +-auto---+  +-const----+  +-unsigned-+
   +-extern-+  +-volatile-+  +-signed---+
   +-static-+

   .-,-------------------------------------------.
   V                                             |
>----variable-name--+-------+--+----------------+-+-- ; ---------><
                    +-[-1-]-+  +- = -expression-+
```

NUL-Terminated Character Form

```
>>-+--------+--+----------+--+----------+--char------------------->
   +-auto---+  +-const----+  +-unsigned-+
   +-extern-+  +-volatile-+  +-signed---+
   +-static-+

   .-,----------------------------------------------.
   V                                                |
>----variable-name--[--length--]--+----------------+-+-- ; ------><
                                  +- = -expression-+
```

Notes:

1. The length must be an integer constant that is greater than 1 and not greater than 32741.
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

```
►►──┬────────┬──┬──────────┬──┬─────────┬──struct──┬─────┬── { ──────────────────►
    ├─auto───┤  ├─const────┤  └─_Packed─┘          └─tag─┘
    ├─extern─┤  └─volatile─┘
    └─static─┘

►──┬────────┬──short──┬─────┬──var-1── ; ──┬──────────┬──char──var-2──[──length──]── ; ── } ──────►
   └─signed─┘         └─int─┘              ├─unsigned─┤
                                           └─signed───┘

   ┌─,──────────────────────────────────────────────────────┐
   ▼                                                        │
►────variable-name──┬───────────────────────────────────────┬┴── ; ──────────────────►◄
                    └─ = ──{──expression ,──expression──}───┘
```

Notes:

1. *length* must be an integer constant that is greater than 0 and not greater than 32740.
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 may use #pragma pack (reset) instead of #pragma pack() since 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

```
                ┌─,──────────────────────────────────────────┐
                ▼                                            │
►►──VARCHAR────variable-name──[──length──]──┬─────────────────┬┴── ; ──────────►◄
                                            └─ = ──"init-data"─┘
```

Notes:

1. VARCHAR can be in mixed case.
2. Length must be an integer constant that is greater than 0 and not greater than 32740.
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";
```

Resuts 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----------------------------->
   +-auto---+  +-const----+
   +-extern-+  '-volatile-'
   '-static-'

   .-,----------------------------------.
   V                                    |
>----variable-name--+----------------+--+-- ; ------------------><
                    '- = --expression-'
```

NUL-Terminated Graphic Form

```
>>-+--------+--+----------+--wchar_t----------------------------->
   +-auto---+  +-const----+
   +-extern-+  '-volatile-'
   '-static-'

   .-,--------------------------------------------------.
   V                                                    |
>----variable-name--[--length--]--+----------------+--+-- ; ----><
                                  '- = --expression-'
```

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

```
>>──┬────────┬──┬──────────┬──┬─────────┬──struct──┬─────┬── { ──────────────────────>
    ├─auto───┤  ├─const────┤  └─_Packed─┘          └─tag─┘
    ├─extern─┤  └─volatile─┘
    └─static─┘

>──┬────────┬──short──┬─────┬──var-1── ; ──wchar_t──var-2──[──length──]── ; ── } ───────>
   └─signed─┘         └─int─┘

    ┌─,──────────────────────────────────────────────────┐
    V                                                    │
>───variable-name──┬──────────────────────────────────────┬──┴── ;──────────────────────><
                   └─ =── {──expression ,──expression── }─┘
```

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;
```

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--+-BLOB---+------------------>
    +-auto-----+  +-const------+               +-CLOB---+
    +-extern---+  +-volatile---+               +-DBCLOB-+
    +-static---+

>--(--lob-length--+---+--)------------------------------------------------->
                  +-K-+
                  +-M-+
                  +-G-+

   .-,-------------------------------------------------------.
   V                                                         |
>----variable-name--+-----------------------------------------+-+-- ; ---><
                    +- = --{--init-len,"init-data" --}--------+
                    +- = --SQL_BLOB_INIT("init-data") --------+
                    +- = --SQL_CLOB_INIT("init-data") --------+
                    +- = --SQL_DBCLOB_INIT("init-data") ------+
```

Notes:

1. K multiplies *lob-length* by 1024. M multiplies *lob-length* by 1,048,576. G multiplies *lob-length* by 1,073,741,824.
2. For BLOB and CLOB, 1 <= *lob-length* <= 2,147,483,647
3. For DBCLOB, 1 <= *lob-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. A length for the LOB must be specified; that is, the following declaration is not permitted

   ```
   SQL TYPE IS BLOB my_blob;
   ```
8. If the LOB is not initialized within the declaration, then no initialization will be done within the precompiler generated code.
9. The precompiler generates a structure tag which can be used to cast to the host variable's type.
10. Pointers to LOB host variables can be declared, with the same rules and restrictions as for pointers to other host variable types.
11. CCSID processing for LOB host variables will be the same as the processing for other character and graphic host variable types.
12. If a DBCLOB is initialized, it is the user's responsibility to prefix the string with an 'L' (indicating a wide-character string).

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");
```

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;
```

LOB Locator

```
>>--+--------+--+----------+--SQL TYPE IS--+-BLOB_LOCATOR---+-------->
    +-auto---+  +-const----+               +-CLOB_LOCATOR---+
    +-extern-+  +-volatile-+               +-DBCLOB_LOCATOR-+
    +-static-+

    .-,---------------------------------.
    V                                   |
>---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--+-BLOB_FILE---+----------->
    +-auto---+  +-const----+               +-CLOB_FILE---+
    +-extern-+  +-volatile-+               +-DBCLOB_FILE-+
    +-static-+

    .-,---------------------------------.
    V                                   |
>---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 pre-compiler will generate 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. See LOB file reference variables in the SQL Programming Concepts book for more information about these values.

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

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

Notes:

1. 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:

In C:

```
_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.

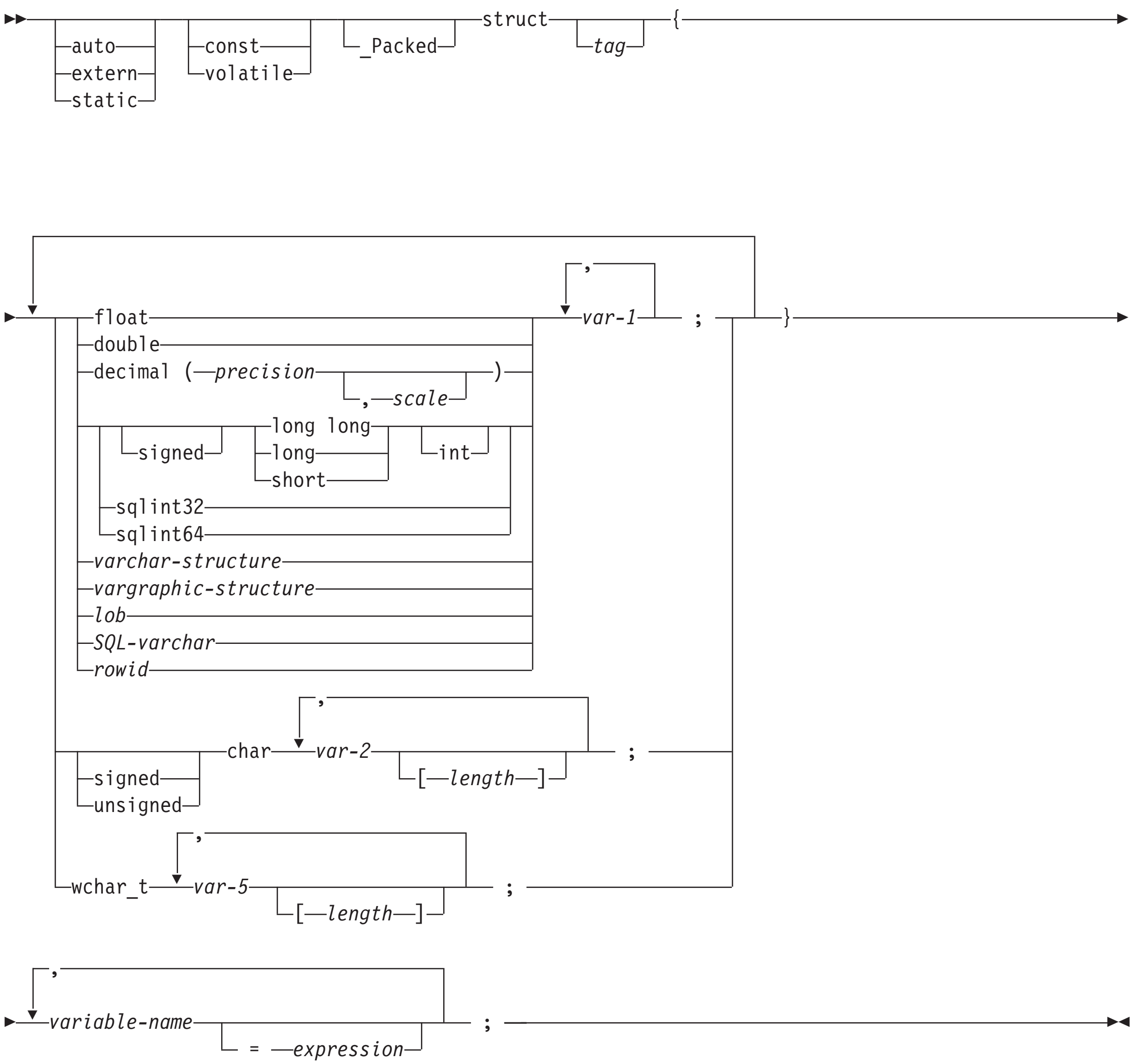
For more details, see the following sections:

- “Host structure declarations in C and C++ applications that use SQL”
- “Host structure indicator array in C and C++ applications that use SQL” on page 32

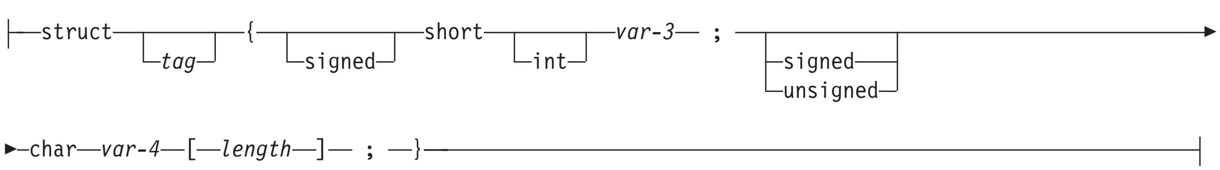
Host structure declarations in C and C++ applications that use SQL

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

Host Structures



varchar-structure:



Host Structures (continued)

vargraphic-structure:

```
├──struct──┬─────┬──{──┬────────┬──short──┬─────┬──────────────────────────►
           └─tag─┘     └─signed─┘         └─int─┘

►─var-6── ; ──wchar_t──var-7──[──length──]── ; ──}──────────────────────────┤
```

lob:

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

SQL-varchar:

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

rowid:

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

Notes:

1. For details on declaring numeric, character, graphic, LOB, and ROWID host variables, see the notes under numeric host variables, character host variables, graphic host variables, LOB host variables, and ROWID 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.

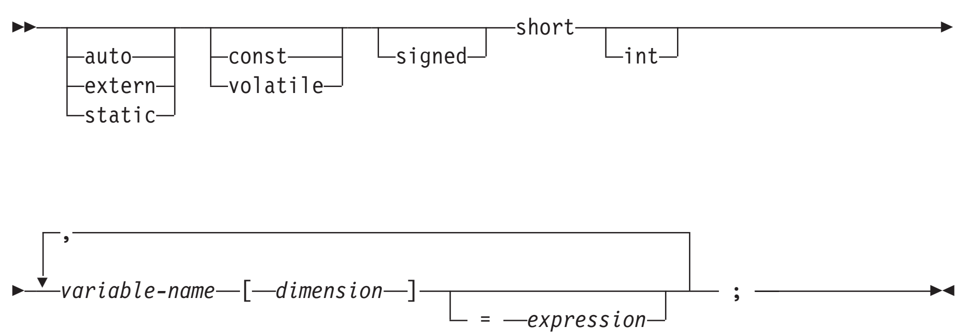
```
#pragma pack(1)
 struct {
       short myshort;
       long mylong;
       char mychar[5];
        } a_st;
#pragma pack()
```

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

The following 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;
...
```

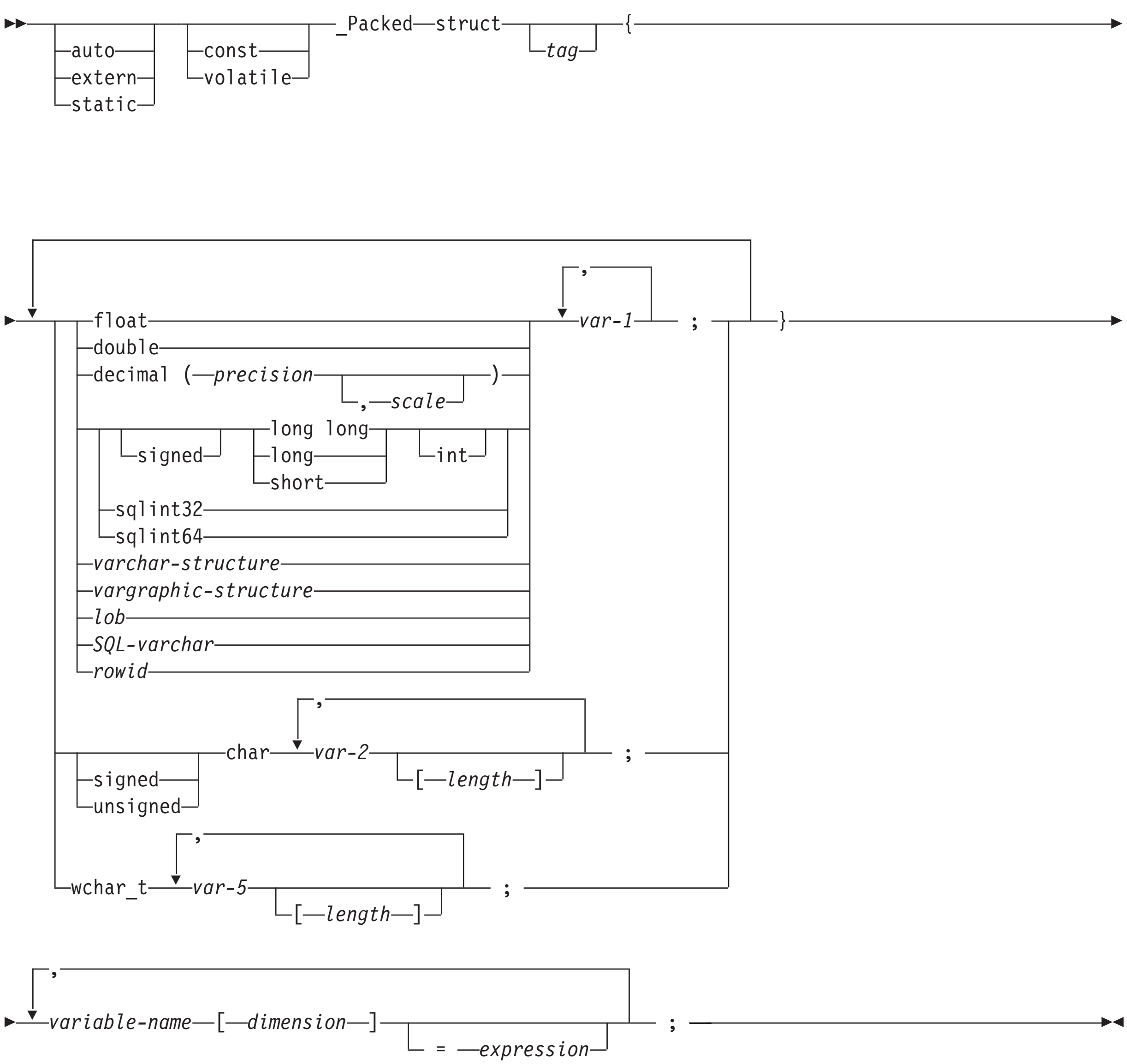
For more details, see the following sections:

- "Host structure array in C and C++ applications that use SQL"
- "Host structure array indicator structure in C and C++ applications that use SQL" on page 35

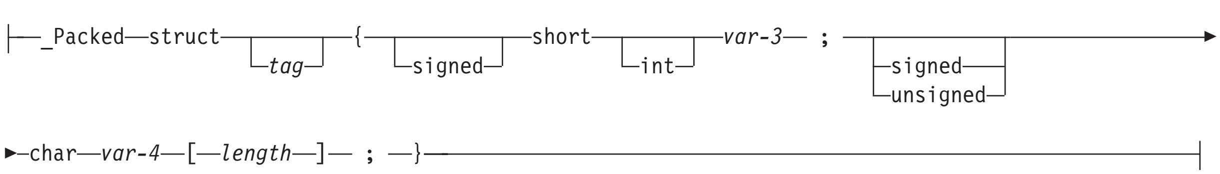
Host structure array in C and C++ applications that use SQL

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

Host Structure Array



varchar-structure:



Host Structure Array (continued)

vargraphic-structure:

```
├──_Packed──struct──┬─────┬──{──┬────────┬──short──┬─────┬──var-6── ; ──────────────►
                    └─tag─┘     └─signed─┘         └─int─┘

►─wchar_t──var-7──[──length──]── ; ──}──────────────────────────────────────────────┤
```

lob:

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

SQL-varchar:

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

rowid:

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

Notes:

1. For details on declaring numeric, character, graphic, LOB, and ROWID host variables, see the notes under numeric-host variables, character-host, graphic-host variables, LOB host variables, and ROWID 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 following figure shows the valid syntax for host structure array indicator structure declarations.

Host Structure Array Indicator Structure

```
>>-+--------+--+----------+--+---------+--struct--+-----+--{------------->
   +-auto---+  +-const----+  '-_Packed-'          '-tag-'
   +-extern-+  '-volatile-'
   '-static-'

>--+--------+--short--+-----+--var-1--[--dimension-1--]--;--}------------>
   '-signed-'         '-int-'

   .-,---------------------------------------------------.
   V                                                     |
>----variable-name--[--dimension-2--]--+-----------------+--;----------><
                                       '- = --expression-'
```

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 indentifier 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]l  /* 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 Programming Concepts* information, 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 only compared against a DATE column or a character string which 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.

Although zoned, binary (with non-zero scale fields), and optionally decimal are mapped to character fields in ILE C for iSeries, 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 for iSeries™ Language Reference book.

Determining equivalent SQL and C or C++ data types

The precompiler determines the base SQLTYPE and SQLLEN of host variables based on the following 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 where length < 255	448	length	VARCHAR (length)
VARCHAR structure form where length > 254	456	length	VARCHAR(length)
single-graphic form	468	1	GRAPHIC(1)
NUL-terminated single-graphic form	400	length	VARGRAPHIC (length - 1)
VARGRAPHIC structured form where length < 128	464	length	VARGRAPHIC (length)
VARGRAPHIC structured form where length > 127	472	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	

Table 2. SQL Data Types Mapped to Typical C or C++ Declarations (continued)

SQL Data Type	C or C++ Data Type	Notes
DECIMAL(p,s)	decimal(p,s)	p is a positive integer from 1 to 31, and s is a positive integer from 0 to 31.
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.
BLOB	None	Use SQL TYPE IS to declare a BLOB in C or C++.
CLOB	None	Use SQL TYPE IS to declare a CLOB in C or C++.
GRAPHIC (1)	single-graphic form	
GRAPHIC (n)	No exact equivalent	If *n* > 1, use NUL-terminated graphic form.
VARGRAPHIC(n)	NUL-terminated graphic 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.
	VARGRAPHIC structured form	*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++.

Table 2. SQL Data Types Mapped to Typical C or C++ Declarations (continued)

SQL Data Type	C or C++ Data Type	Notes
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.
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++.

For more details, see "Notes on C and C++ variable declaration and usage".

Notes on C and C++ variable declaration and usage

Apostrophes and quotation marks have different meanings in C, C++, and SQL. C and C++ use quotation marks to delimit string constants and apostrophes to delimit character constants. SQL does not have this distinction, but uses quotation marks for delimited identifiers and uses apostrophes 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.

See the indicator variables topic in the SQL Reference book for more information.

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;
```

Chapter 3. Coding SQL Statements in COBOL Applications

The iSeries system supports more than one COBOL compiler. The DB2 UDB Query Manager and SQL Development Kit licensed program only supports the COBOL for iSeries and ILE COBOL for iSeries languages. This chapter describes the unique application and coding requirements for embedding SQL statements in a COBOL program. Requirements for host structures and host variables are defined.

For more details, see the following sections:

- "Defining the SQL Communications Area in COBOL applications that use SQL"
- "Defining SQL Descriptor Areas in COBOL applications that use SQL" on page 44
- "Embedding SQL statements in COBOL applications that use SQL" on page 45
- "Using host variables in COBOL applications that use SQL" on page 48
- "Using host structures in COBOL applications that use SQL" on page 57
- "Using external file descriptions in COBOL applications that use SQL" on page 65
- "Determining equivalent SQL and COBOL data types" on page 67
- "Using indicator variables in COBOL applications that use SQL" on page 70

A detailed sample COBOL program, showing how SQL statements can be used, is provided in Appendix A, "Sample Programs Using DB2 UDB for iSeries Statements".

Note: See "Code disclaimer information" on page viii information for information pertaining to code examples.

Defining the SQL Communications Area in COBOL applications that use SQL

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 executed. 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. 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).
   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 for iSeries, the SQLCA is declared using the GLOBAL clause. 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.

For more information about SQLCA, see SQL Communication Area in the SQL Reference book.

Defining SQL Descriptor Areas in COBOL applications that use SQL

The following statements require 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*

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 for iSeries, the SQLDA is declared using the GLOBAL clause.

Dynamic SQL is an advanced programming technique described in Dynamic SQL Applications in the *DB2 UDB for iSeries Programming Concepts* information. 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.

For more information about SQLDA, refer to SQL Descriptor Area in the SQL Reference book.

Embedding SQL statements in COBOL applications that use SQL

SQL statements can be coded in COBOL program sections as follows:

SQL Statement	Program Section
BEGIN DECLARE SECTION END DECLARE SECTION DECLARE VARIABLE DECLARE STATEMENT	WORKING-STORAGE SECTION or LINKAGE SECTION
INCLUDE SQLCA INCLUDE SQLDA	WORKING-STORAGE SECTION or LINKAGE SECTION
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.
```

For more details, see the following sections:

- "Comments in COBOL applications that use SQL"
- "Continuation for SQL statements in COBOL applications that use SQL"
- "Including code in COBOL applications that use SQL" on page 47
- "Margins in COBOL applications that use SQL" on page 47
- "Sequence numbers in COBOL applications that use SQL" on page 47
- "Names in COBOL applications that use SQL" on page 47
- "COBOL compile-time options in COBOL applications that use SQL" on page 47
- "Statement labels in COBOL applications that use SQL" on page 47
- "WHENEVER Statement in COBOL applications that use SQL" on page 47
- "Multiple source COBOL programs and the SQL COBOL precompiler" on page 48

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 at the point 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

Code SQL statements in columns 12 through 72. If EXEC SQL starts before the specified margin (that is, before column 12), the SQL precompiler will 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. A host variable used in an SQL statement must be declared prior to the first use of the host variable in an SQL statement.

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.

For more details, see "Declaring host variables in COBOL applications that use SQL".

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

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

BIGINT and INTEGER and SMALLINT

```
►►──┬─01──────┬──variable-name──┬─PICTURE─┬──┬────┬──picture-string──────►
    ├─77──────┤                 └─PIC─────┘  └─IS─┘
    └─level-1─┘

►──┬──────────────┬──┬─BINARY───────────┬──────────────────────────────►
   └─USAGE─┬────┬─┘  ├─COMPUTATIONAL-4──┤
           └─IS─┘    └─COMP-4───────────┘

►──┬──────────────────────────────────┬─.──────────────────────────────►◄
   └─VALUE─┬────┬──numeric-constant───┘
           └─IS─┘
```

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

```
>>-+-01------+--variable-name--+-PICTURE-+--+----+--picture-string-->
   +-77------+                 '-PIC-----'  '-IS-'
   '-level-1-'

>--+--------------------+--+-PACKED-DECIMAL----+------------------->
   '-USAGE--+----+------'  +-COMPUTATIONAL-3---+
            '-IS-'         +-COMP-3------------+
                           +-COMPUTATIONAL-----+
                           '-COMP--------------'

>--+---------------------------------+-- . ----------------------><
   '-VALUE--+----+--numeric-constant-'
            '-IS-'
```

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 18.
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 the COBOL for iSeries 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 18.
3. level-1 indicates a COBOL level between 2 and 48.

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

Numeric

```
>>──┬─01──────┬──variable-name──┬─PICTURE─┬──┬────┬──picture-string──────►
    ├─77──────┤                 └─PIC─────┘  └─IS─┘
    └─level-1─┘

►──┬──────────────────────────────────────────┬───────────────────────►
   └─USAGE──┬────┬──┬─DISPLAY────────┬────────┘
            └─IS─┘  └─display clause─┘

►──┬──────────────────────────────────┬── . ──────────────────────────►◄
   └─VALUE──┬────┬──numeric-constant──┘
            └─IS─┘
```

display clause:

```
├──┬─────────┬──SIGN──┬────┬──LEADING SEPARATE──┬───────────┬──────────┤
   └─DISPLAY─┘        └─IS─┘                    └─CHARACTER─┘
```

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

The following figure shows the syntax for valid floating point host variable declarations. Floating point host variables are only supported for ILE COBOL for iSeries.

Floating-point

```
>>──┬─01──────┬──variable-name──┬──────────────────┬──┬─COMPUTATIONAL-1─┬──►
    ├─77──────┤                 └─USAGE──┬────┬────┘  ├─COMP-1──────────┤
    └─level-1─┘                          └─IS─┘       ├─COMPUTATIONAL-2─┤
                                                      └─COMP-2──────────┘

►──┬──────────────────────────────────┬── . ──────────────────────────►◄
   └─VALUE──┬────┬──numeric-constant──┘
            └─IS─┘
```

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
- Varying-Length Strings

Fixed-Length Character Strings

```
>>──┬─01──────┬─variable-name──┬─PICTURE─┬──┬────┬─picture-string──────────>
    ├─77──────┤                └─PIC─────┘  └─IS─┘
    └─level-1─┘

>──┬──────────────────────────┬──┬──────────────────────────────┬──────────>
   └─┬─────────────┬─DISPLAY──┘  └─VALUE──┬────┬─string-constant─┘
     └─USAGE─┬────┬┘                      └─IS─┘
             └─IS─┘

>── . ─────────────────────────────────────────────────────────────────><
```

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

```
>>──┬─01──────┬─variable-name── . ─49─var-1──┬─PICTURE─┬──────────────────>
    └─level-1─┘                              └─PIC─────┘

>──┬────┬─picture-string-1──┬──────────────────┬─┬─BINARY──────────┬──────>
   └─IS─┘                   └─USAGE─┬────┬─────┘ ├─COMPUTATIONAL-4─┤
                                    └─IS─┘       └─COMP-4──────────┘

>──┬───────────────────────────────────────┬─ . ─49─var-2──┬─PICTURE─┬────>
   └─VALUE──┬────┬─numeric-constant────────┘               └─PIC─────┘
            └─IS─┘

>──┬────┬─picture-string-2──┬────────────────────────────┬────────────────>
   └─IS─┘                   └─┬──────────────┬─DISPLAY───┘
                              └─USAGE─┬────┬─┘
                                      └─IS─┘

>──┬─────────────────────────────────────┬─ . ─────────────────────────><
   └─VALUE──┬────┬─string-constant───────┘
            └─IS─┘
```

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 will use the full size of the S9(m) variable even though COBOL on the iSeries only recognizes values up to the specified precision. This can cause data truncation errors when COBOL statements are being run and may 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 for iSeries.

There are two valid forms of graphic host variables:

- Fixed-Length Graphic Strings
- Varying-Length Graphic Strings

Fixed-Length Graphic Strings

```
>>-+-01------+--variable-name--+-PICTURE-+--+----+--picture-string------>
   +-77------+                 '-PIC-----'  '-IS-'
   '-level-1-'

>--+--------------------------DISPLAY-1-+--+-VALUE--+----+--string-constant-+-->
   '-USAGE--+----+---------'               '-IS-'
            '-IS-'

>-- . ------------------------------------------------------------------><
```

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

```
►►──┬─01──────┬──variable-name── . ──49──var-1──┬─PICTURE─┬──────────────────►
    └─level-1─┘                                 └─PIC─────┘

►──┬────┬──picture-string-1──┬──────────────────┬──┬─BINARY──────────┬──────►
   └─IS─┘                    └─USAGE──┬────┬────┘  ├─COMPUTATIONAL-4─┤
                                      └─IS─┘       └─COMP-4──────────┘

►──┬──────────────────────────────────────┬── . ──49──var-2──┬─PICTURE─┬────►
   └─VALUE──┬────┬──numeric-constant──────┘                  └─PIC─────┘
            └─IS─┘

►──┬────┬──picture-string-2──┬────────────────────────────────┬─────────────►
   └─IS─┘                    └─┬──────────────────┬─DISPLAY-1─┘
                               └─USAGE──┬────┬────┘
                                        └─IS─┘

►──┬─────────────────────────────────────┬── . ─────────────────────────────►◄
   └─VALUE──┬────┬──string-constant──────┘
            └─IS─┘
```

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 will use the full size of the S9(m) variable even though COBOL on the iSeries only recognizes values up to the specified precision. This can cause data truncation errors when COBOL statements are being run and may 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.

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 for iSeries.

LOB Host Variables

```
>>--01--variable-name--+---------------+--SQL TYPE IS--+-BLOB---+---------->
                       '-USAGE--+----+-'               +-CLOB---+
                                '-IS-'                 '-DBCLOB-'

>--(--lob-length--+---+--)-- . ---------------------------------------><
                  +-K-+
                  '-M-'
```

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.

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).
```

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.
```

LOB Locator

```
>>--01--variable-name--+--------------+----------------------------->
                       '-USAGE-+----+-'
                               '-IS-'

>--SQL TYPE IS--+-BLOB-LOCATOR---+-- . ---------------------------><
                +-CLOB-LOCATOR---+
                '-DBCLOB-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.

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.
```

CLOB and DBCLOB locators have similar syntax.

LOB File Reference Variable

```
>>--01--variable-name--+--------------+--SQL TYPE IS--+-BLOB-FILE---+---->
                       '-USAGE-+----+-'               +-CLOB-FILE---+
                               '-IS-'                 '-DBCLOB-FILE-'

>-- . -----------------------------------------------------------------><
```

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 pre-compiler will generate 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. See LOB file reference variables in the SQL Programming Concepts book for more information about these values.

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

Datetime host variables in COBOL applications that use SQL

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

Datetime Host Variable

```
►►──┬─01──────┬──variable-name──FORMAT──┬──────┬──┬─DATE──────┬──┬────┬──────────►
    ├─77──────┤                         └─OF───┘  ├─TIME──────┤  └─IS─┘
    └─level-1─┘                                   └─TIMESTAMP─┘

►─format-options───────────────────────────────────────────────────────────────►◄
```

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 Reference book 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 (that is, PL/I style). For example, specify B.C1 rather than C1 OF B or C1 IN B. However, PL/I style 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.

For more details, see the following sections:

- “Host structure in COBOL applications that use SQL”
- “Host structure indicator array in COBOL applications that use SQL” on page 61
- “Using host structure arrays in COBOL applications that use SQL” on page 61
- “Host structure array in COBOL applications that use SQL” on page 62
- “Host array indicator structure in COBOL applications that use SQL” on page 65

Host structure in COBOL applications that use SQL

The following figure shows the syntax for the valid host structure.

Host Structure

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

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

floating-point:

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

usage-clause:

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

display-clause:

```
├──┬───────────┬──SIGN──┬────┬──LEADING──SEPARATE──┬─────────────┬──────────────────────────┤
   └─DISPLAY───┘        └─IS─┘                     └─CHARACTER───┘
```

Host Structure (continued)

varchar-string:

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

►──┬───────────────────────────────────┬──.──49──var-3──┬─PICTURE─┬──┬────┬──picture-string-2──────►
   └─VALUE──┬────┬──numeric-constant───┘                └─PIC─────┘  └─IS─┘
            └─IS─┘

►──┬───────────────────────────┬──┬────────────────────────────┬─────────────────────────────────┤
   └─┬─────────────┬──DISPLAY──┘  └─VALUE──┬────┬──constant────┘
     └─USAGE─┬────┬┘                       └─IS─┘
             └─IS─┘
```

vargraphic-string:

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

►──┬───────────────────────────────────┬──.──49──var-3──┬─PICTURE─┬──┬────┬──picture-string-2──────►
   └─VALUE──┬────┬──numeric-constant───┘                └─PIC─────┘  └─IS─┘
            └─IS─┘

►──┬─────────────────────────────┬──┬────────────────────────────┬───────────────────────────────┤
   └─┬─────────────┬──DISPLAY-1──┘  └─VALUE──┬────┬──constant────┘
     └─USAGE─┬────┬┘                         └─IS─┘
             └─IS─┘
```

lob:

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

datetime:

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

rowid:

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

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 for iSeries.
4. For details on declaring numeric, character, graphic, LOB, and ROWID host variables, see the notes under numeric-host variables, character-host variables, graphic-host variables, LOB host variables, and ROWID host variables..
5. *format-options* indicates valid datetime options that are supported by the COBOL compiler. See the ILE COBOL Reference book for details.

Host structure indicator array in COBOL applications that use SQL

The following figure shows the syntax for valid indicator array declarations.

Host Structure Indicator Array

```
>>--level-1--variable-name--+-PICTURE-+--+----+--picture-string-------------->
                            '-PIC-----'  '-IS-'

>--+---------------+--+-BINARY----------------+--OCCURS--dimension--+-------+-->
   '-USAGE--+----+-'  +-COMPUTATIONAL-4-------+                     '-TIMES-'
            '-IS-'    '-COMP-4----------------'

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

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

The following figures show the syntax for valid host structure array declarations.

Host Structure Array

```
>>--level-1--variable-name--OCCURS--dimension--+-------+--.------------------------>
                                               '-TIMES-'

   .--------------------------------------------------------------------------.
   V                                                                          |
>----level-2--var-1--+--+-PICTURE-+--+----+--picture-string-1--usage-clause--.-+-+--------><
                     |  '-PIC-----'  '-IS-'                                  |
                     +-floating-point--.-------------------------------------+
                     +-.--varchar-string--.----------------------------------+
                     +-.--vargraphic-string--.-------------------------------+
                     +-lob--.------------------------------------------------+
                     +-datetime--.-------------------------------------------+
                     '-rowid--.----------------------------------------------'
```

floating-point:

```
|--+--------------------------------------------------+--+---------------------------+--|
   '-+------------------+--+-COMPUTATIONAL-1-+--------'  '-VALUE--+----+--constant--'
     '-USAGE--+----+----'  +-COMP-1----------+                    '-IS-'
              '-IS-'       +-COMPUTATIONAL-2-+
                           '-COMP-2----------'
```

usage-clause:

```
|--+--------------------------------------------------------+--+---------------------------+--|
   '-+------------------+--+-+-BINARY----------+-----------+'  '-VALUE--+----+--constant--'
     '-USAGE--+----+----'  | +-COMPUTATIONAL-4-+           |            '-IS-'
              '-IS-'       | '-COMP-4----------'           |
                           +-+-PACKED-DECIMAL--+-----------+
                           | +-COMPUTATIONAL-3-+           |
                           | +-COMP-3----------+           |
                           | +-COMPUTATIONAL---+           |
                           | '-COMP------------'           |
                           +-+-DISPLAY--------+------------+
                           | '-display-clause-'            |
                           '-DISPLAY-1---------------------'
```

display-clause:

```
|--+---------+--SIGN--+----+--LEADING--SEPARATE--+-----------+----------------------|
   '-DISPLAY-'        '-IS-'                     '-CHARACTER-'
```

Host Structure Array (continued)

varchar-string:

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

>--+-------------------------------+--.--49--var-3--+-PICTURE-+--+----+--picture-string-3-->
   '-VALUE--+----+--numeric-constant-'              '-PIC-----'  '-IS-'
            '-IS-'

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

vargraphic-string:

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

>--+-------------------------------+--.--49--var-3--+-PICTURE-+--+----+--picture-string-3-->
   '-VALUE--+----+--numeric-constant-'              '-PIC-----'  '-IS-'
            '-IS-'

>--+---------------------------+--+-------------------------+------------------------|
   '-+--------------+-DISPLAY-1-'  '-VALUE--+----+--constant-'
     '-USAGE-+----+-'                       '-IS-'
             '-IS-'
```

lob:

```
|--+--------------+--SQL TYPE IS--+-+-BLOB---+--(--lob-length--+---+--)-+----------------|
   '-USAGE-+----+-'               | +-CLOB---+                 +-K-+    |
           '-IS-'                 | '-DBCLOB-'                 '-M-'    |
                                  +-+-BLOB-LOCATOR---+------------------+
                                  | +-CLOB-LOCATOR---+                  |
                                  | '-DBCLOB-LOCATOR-'                  |
                                  '-+-BLOB-FILE---+---------------------'
                                    +-CLOB-FILE---+
                                    '-DBCLOB-FILE-'
```

datetime:

```
|--variable-name--FORMAT--+----+--+-DATE------+--+----+--format-options--------------|
                          '-OF-'  +-TIME------+  '-IS-'
                                  '-TIMESTAMP-'
```

rowid:

```
|--SQL TYPE IS ROWID-----------------------------------------------------------------|
```

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 for iSeries.
4. For details on declaring numeric, character, graphic, LOB host variables, see the notes under numeric-host variables, character-host variables, graphic-host variables, and LOB 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 Reference book for details.

Host array indicator structure in COBOL applications that use SQL

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

Host Structure Array Indicator Structure

```
>>──level-1──variable-name──OCCURS──dimension──┬───────┬──.──────────────>
                                               └─TIMES─┘

>─level-2──var-1──┬─PICTURE─┬──┬────┬──picture-string──┬──────────────┬──>
                  └─PIC─────┘  └─IS─┘                  └─USAGE─┬────┬─┘
                                                               └─IS─┘

>──┬─BINARY───────────┬──┬──────────────────────────┬──.─────────────><
   ├─COMPUTATIONAL-4──┤  └─VALUE──┬────┬──constant──┘
   └─COMP-4───────────┘           └─IS─┘
```

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 Programming Concepts* information, 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 book and the ILE COBOL Reference book.

For more details on external file descriptions, see "Using external file descriptions for host structure arrays in COBOL applications that use SQL".

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 was 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 COBOL for iSeries, 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.

Determining equivalent SQL and COBOL data types

The precompiler determines the base SQLTYPE and SQLLEN of host variables based on the following 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 for COBOL for iSeries.	492	8	BIGINT
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 for COBOL for iSeries.	480	4	FLOAT(single precision)
COMP-2 Not supported for COBOL for iSeries.	480	8	FLOAT(double precision)
Fixed-length character data	452	m	CHAR(m)
Varying-length character data where m < 255	448	m	VARCHAR(m)
Varying-length character data where m > 254	456	m	VARCHAR(m)

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
Fixed-length graphic data Not supported for COBOL for iSeries.	468	m	GRAPHIC(m)
Varying-length graphic data where m < 128 Not supported for COBOL for iSeries.	464	m	VARGRAPHIC(m)
Varying-length graphic data where m > 127 Not supported for COBOL for iSeries.	472	m	VARGRAPHIC(m)
DATE Not supported for COBOL for iSeries.	384		DATE
TIME Not supported for COBOL for iSeries.	388		TIME
TIMESTAMP Not supported for COBOL for iSeries.	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 for iSeries. Not supported for COBOL for iSeries.	m is from 10 to 18
DECIMAL(p,s)	If p<19: S9(p-s)V9(s) PACKED-DECIMAL or S9(p-s)V9(s) COMP or S9(p-s)V9(s) COMP-3 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).
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).
FLOAT(single precision)	COMP-1 for ILE COBOL for iSeries. Not supported for COBOL for iSeries.	
FLOAT(double precision)	COMP-2 for ILE COBOL for iSeries. Not supported for COBOL for iSeries.	

Table 4. SQL Data Types Mapped to Typical COBOL Declarations (continued)

SQL Data Type	COBOL Data Type	Notes
CHAR(n)	Fixed-length character string	32766≥n≥1
VARCHAR(n)	Varying-length character string	32740≥n≥1
BLOB	None	Use SQL TYPE IS to declare a BLOB. For ILE COBOL for iSeries. Not supported for COBOL for iSeries.
CLOB	None	Use SQL TYPE IS to declare a CLOB. For ILE COBOL for iSeries. Not supported for COBOL for iSeries.
GRAPHIC(n)	Fixed-length graphic string for ILE COBOL for iSeries. Not supported for COBOL for iSeries.	16383≥n≥1
VARGRAPHIC(n)	Varying-length graphic string for ILE COBOL for iSeries. Not supported for COBOL for iSeries.	16370≥n≥1
DBCLOB	None	Use SQL TYPE IS to declare a DBCLOB. For ILE COBOL for iSeries. Not supported for COBOL for iSeries.
DATE	Fixed-length character string or DATE (for ILE COBOL for iSeries)	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 for iSeries)	Allow at least 6 characters; 8 to include seconds.
TIMESTAMP	Fixed-length character string or TIMESTAMP (for ILE COBOL for iSeries)	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.

For more details, see “Notes on COBOL variable declaration and usage”.

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, BIGINT, and INTEGER 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.

See the indicator variables topic in the SQL Reference book for more information.

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.
```

Chapter 4. Coding SQL Statements in PL/I Applications

This chapter describes the unique application and coding requirements for embedding SQL statements in an iSeries PL/I program. Requirements for host structures and host variables are defined.

For more details, see the following sections:

A detailed sample PL/I program, showing how SQL statements can be used, is provided in Appendix A, "Sample Programs Using DB2 UDB for iSeries Statements".

Note: See "Code disclaimer information" on page viii information for information pertaining to code examples.

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 the following:

- 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 executed. 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.

For more information about SQLCA, see SQL Communication Area in the SQL Reference book.

Defining SQL Descriptor Areas in PL/I applications that use SQL

The following statements require 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*

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 described in Dynamic SQL Applications in the *DB2 UDB for iSeries Programming Concepts* information. 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.

For more information about SQLDA, see SQL Descriptor Area in the SQL Reference book.

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.

For more details, see the following sections:
- "Example: Embedding SQL statements in PL/I applications that use SQL"
- "Comments in PL/I applications that use SQL"
- "Continuation for SQL statements in PL/I applications that use SQL" on page 74
- "Including code in PL/I applications that use SQL" on page 74
- "Margins in PL/I applications that use SQL" on page 74
- "Names in PL/I applications that use SQL" on page 74
- "Statement labels in PL/I applications that use SQL" on page 74
- "WHENEVER Statement in PL/I applications that use SQL" on page 74

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

An UPDATE statement coded in a PL/I program might be coded as follows:

```
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

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. For more information about the CRTSQLPLI command, see Appendix B, "DB2 UDB for iSeries CL Command Descriptions for Host Language Precompilers".

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 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.

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.

For more details, see "Declaring host variables in PL/I applications that use SQL".

Declaring host variables in PL/I applications that use SQL

The PL/I precompilers only recognize 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

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

Numeric

```
>>─┬─DECLARE─┬─┬─variable-name──────────────────┬──────────────────────>
   └─DCL─────┘ │    ┌─,──────────────┐          │
               └─(──▼─variable-name──┴──)───────┘

>─┬─┬─BINARY─┬──┬─FIXED─┬─────────────────┬──┬───────────────┬─────────>
  │ └─BIN────┘  │       └─(──precision──)─┘  │               │
  │             └─FLOAT─┬─────────────────┬──┘               │
  │                     └─(──precision──)─┘                  │
  ├─┬─DECIMAL─┬──┬─FIXED─┬──────────────────────────────┬─┬──┤
  │ └─DEC─────┘  │       └─(──precision──┬─────────┬──)─┘ │  │
  │              │                       └─,scale──┘      │  │
  │              └─FLOAT─┬─────────────────┬──────────────┘  │
  │                      └─(──precision──)─┘                 │
  └─PICTURE──picture-string──────────────────────────────────┘

>─┬───────────────────────────────────────────┬─ ;────────────────────><
  └─Alignment and/or Scope and/or Storage─────┘
```

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

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

Character

```
>>─┬─DECLARE─┬─┬─variable-name──────────────────┬─┬─CHARACTER─┬───────>
   └─DCL─────┘ │    ┌─,──────────────┐          │ └─CHAR──────┘
               └─(──▼─variable-name──┴──)───────┘

>─┬──────────────┬─┬───────────┬──────────────────────────────────────>
  └─(──length──)─┘ ├─VARYING───┤
                   └─VAR───────┘

>─┬───────────────────────────────────────────┬─ ;────────────────────><
  └─Alignment and/or Scope and/or Storage─────┘
```

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.

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

```
>>--+-DECLARE-+--+-variable-name-------------------+--SQL TYPE IS--+-BLOB-+------->
    '-DCL-----'  |           .-,-------------.      |               '-CLOB-'
                 |           V               |      |
                 '-(---------variable-name---+--)---'

>--(--lob-length--+---+--)------------------------------------------------------><
                  '-K-'
```

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

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);
```

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);
```

The following figure shows the syntax for valid LOB locators.

LOB locator

```
>>──┬─DECLARE─┬──┬─variable-name──────────────────┬──────────────────────>
    └─DCL─────┘  │       ┌─,─────────────┐         │
                 │       ▼               │         │
                 └─(───variable-name─────┴──)──────┘

>─SQL TYPE IS──┬─BLOB_LOCATOR───┬────────────────────────────────────────><
               ├─CLOB_LOCATOR───┤
               └─DBCLOB_LOCATOR─┘
```

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──────────────────┬──────────────────────>
    └─DCL─────┘  │       ┌─,─────────────┐         │
                 │       ▼               │         │
                 └─(───variable-name─────┴──)──────┘

>─SQL TYPE IS──┬─BLOB_FILE───┬───────────────────────────────────────────><
               ├─CLOB_FILE───┤
               └─DBCLOB_FILE─┘
```

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 locators 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)

See LOB file reference variables in the SQL Programming Concepts book for more information about these values.

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;
```

For more information, see the following sections:

- “Host structures in PL/I applications that use SQL”
- “Host structure indicator arrays in PL/I applications that use SQL” on page 81

Host structures in PL/I applications that use SQL

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

Host Structures

```
>>-+-+-DECLARE-+--1--variable-name--+-----------------------+--,-+----------->
   | '-DCL-----'                    '-Scope and/or storage--'    |
   '-level-1--variable-name--,-----------------------------------'

   .-,--------------------------------------------.
   V                                              |
>----level-2--+-var-1-------------+--data-types---+--;-----------------------><
              |    .-,-----.      |
              |    V       |      |
              '-(----var-2-+--)---'
```

data-types:

```
|--+-+-BINARY-+--+-FIXED-+--+-------------------+--+-----------+-+-------------|
   | '-BIN----'  '-FLOAT-'  '-(--precision--)--'  '-UNALIGNED-' |
   +-+-DECIMAL-+--+-FIXED--+-----------------------------+---+--+
   | '-DEC-----'  |        '-(--precision--+----------+--)-'   |  |
   |              |                        '-,--scale-'        |  |
   |              '-FLOAT--+-------------------+--+-----------+-'  |
   |                       '-(--precision--)--'  '-UNALIGNED-'    |
   +-PICTURE--picture-string-------------------------------------+
   +-+-CHARACTER-+--+----------------+--+-----------------------------+-+
   | '-CHAR------'  '-(--length--)--'  '-+-VARYING-+--+---------+-'  |
   |                                     '-VAR-----'  '-ALIGNED-'    |
   +-SQL TYPE IS--+-+-BLOB-+--(--lob-length--+---+--)-+--------------+
   |              | '-CLOB-'                 '-K-'     |              |
   |              +-+-BLOB_LOCATOR----+----------------+              |
   |              | +-CLOB_LOCATOR----+                |              |
   |              | '-DBCLOB_LOCATOR--'                |              |
   |              '-+-BLOB_FILE----+-------------------'              |
   |                +-CLOB_FILE----+                                  |
   |                '-DBCLOB_FILE--'                                  |
   '-SQL TYPE IS ROWID------------------------------------------------'
```

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, and ROWID host variables, see the notes under numeric-host variables, character-host variables, LOB host variables, and ROWID host variables.

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

The following figure shows the syntax for valid indicator arrays.

Host Structure Indicator Array

```
>>──┬─DECLARE─┬──┬─variable-name──(──dimension──)───────────────────────┬──>
    └─DCL─────┘  │           ┌─,──────────────────────────────────┐      │
                 └─(────────▼─variable-name──(──dimension──)──────┴──)──┘

>──┬─BINARY─┬──FIXED──┬──────────────────────┬────────────────────────────>
   └─BIN────┘         └─(──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;
```

For more details, see "Host structure array in PL/I applications that use SQL" on page 83.

Host structure array in PL/I applications that use SQL

The following syntax diagram shows the syntax for valid structure array declarations.

Host Structure Array

```
>>─┬─┬─DECLARE─┬──1──variable-name──(──dimension──)──┬─────────────────────┬─,─┬──────────────>
   │ └─DCL─────┘                                      └─Scope and/or storage─┘   │
   └─level-1──variable-name──,───────────────────────────────────────────────────┘

  ┌─,────────────────────────────────────────┐
  V                                           │
>───level-2──┬─var-1────────────────┬─data-types──┴──;────────────────────────────────────><
             │      ┌─,─────┐       │
             │      V       │       │
             └─(──────var-2─┴──)────┘
```

data-types:

```
├──┬─┬─BINARY─┬──┬─FIXED─┬──┬──────────────────┬──UNALIGNED──┬──────────────────────────────┤
   │ └─BIN────┘  └─FLOAT─┘  └─(──precision──)─┘              │
   ├─┬─DECIMAL─┬──┬─FIXED──┬──────────────────────────────┬─┬─┤
   │ └─DEC─────┘  │        └─(──precision──┬──────────┬──)─┘ │ │
   │              │                        └─,──scale─┘      │ │
   │              └─FLOAT──┬──────────────────┬──UNALIGNED───┘ │
   │                       └─(──precision──)─┘                 │
   ├─PICTURE──picture-string───────────────────────────────────┤
   ├─┬─CHARACTER─┬──┬───────────────┬──┬─────────────┬─────────┤
   │ └─CHAR──────┘  └─(──length──)─┘  └─┬─VARYING─┬─┘         │
   │                                    └─VAR─────┘           │
   ├─SQL TYPE IS──┬─┬─BLOB─┬──(──lob-length──┬───┬──)──┬───────┤
   │              │ └─CLOB─┘                 └─K─┘     │       │
   │              ├─┬─BLOB_LOCATOR───┬─────────────────┤       │
   │              │ ├─CLOB_LOCATOR───┤                 │       │
   │              │ └─DBCLOB_LOCATOR─┘                 │       │
   │              └─┬─BLOB_FILE───┬────────────────────┘       │
   │                ├─CLOB_FILE───┤                            │
   │                └─DBCLOB_FILE─┘                            │
   └─SQL TYPE IS ROWID─────────────────────────────────────────┘
```

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, and ROWID host variables, see the notes under numeric-host variables, character-host variables, LOB host variables, and ROWID host variables.
5. Dimension must be an integer constant between 1 and 32767.

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

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

Host Structure Array Indicator Structure

```
>>--+-+-DECLARE-+--1--variable-name--(--dimension--)--+--------------------+--,--+--------->
    | '-DCL-----'                                     '-Scope and/or storage-'    |
    '-level-1--variable-name--,---------------------------------------------------'

>--level-2--identifier--(--dimension-2--)--+-BINARY-+--FIXED--+-------------------+--;----><
                                           '-BIN----'         '-(--precision--)-'
```

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 Programming Concepts* information, 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 was 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.

Determining equivalent SQL and PL/I data types

The precompiler determines the base SQLTYPE and SQLLEN of host variables based on the following 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 where n <255	448	n	VARCHAR(n)
CHAR(n) varying where n > 254	456	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	Explanatory 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) VAR	*n* is a positive integer from 1 to 32740.
BLOB	None	Use SQL TYPE IS to declare a BLOB.
CLOB	None	Use SQL TYPE IS to declare a CLOB.
GRAPHIC(n)	Not supported	Not supported.
VARGRAPHIC(n)	Not supported	Not supported.
DBCLOB	None	Use SQL TYPE IS to declare a DBCLOB.
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.

Table 6. SQL Data Types Mapped to Typical PL/I Declarations (continued)

SQL Data Type	PL/I Equivalent	Explanatory Notes
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.

See the indicator variables topic in the SQL Reference book for more information.

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;
```

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.

For more information about the structure parameter passing technique, see Database application design tips: Use structure parameter passing techniques in the *DB2 UDB for iSeries Database Performance and Query Optimization* information.

Chapter 5. Coding SQL Statements in RPG for iSeries Applications

The RPG for iSeries 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 chapter describes the unique application and coding requirements for embedding SQL statements in a RPG for iSeries program. Requirements for host variables are defined.

For more details, see the following sections:

- “Defining the SQL Communications Area in RPG for iSeries applications that use SQL” on page 90
- “Defining SQL Descriptor Areas in RPG for iSeries applications that use SQL” on page 90
- “Embedding SQL statements in RPG for iSeries applications that use SQL” on page 91
- “Using host variables in RPG for iSeries applications that use SQL” on page 93
- “Using host structures in RPG for iSeries applications that use SQL” on page 94
- “Using host structure arrays in RPG for iSeries applications that use SQL” on page 94
- “Using external file descriptions in RPG for iSeries applications that use SQL” on page 95
- “Determining equivalent SQL and RPG for iSeries data types” on page 96
- “Using indicator variables in RPG for iSeries applications that use SQL” on page 99
- “Differences in RPG for iSeries because of structure parameter passing techniques” on page 100
- “Correctly ending a called RPG for iSeries program that uses SQL” on page 100

A detailed sample RPG for iSeries program, showing how SQL statements can be used, is provided in Appendix A, “Sample Programs Using DB2 UDB for iSeries Statements”.

Note: See “Code disclaimer information” on page viii information for information pertaining to code examples.

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

Defining the SQL Communications Area in RPG for iSeries applications that use SQL

The SQL precompiler automatically places the SQLCA in the input specifications of the RPG for iSeries 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 for iSeries:

```
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 for iSeries are limited to 6 characters. The standard SQLCA names have been changed to a length of 6. RPG for iSeries 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.

See SQL Communication Area in the SQL Reference book for more information.

Defining SQL Descriptor Areas in RPG for iSeries applications that use SQL

The following statements require 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*

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 described in Dynamic SQL Applications in the *DB2 UDB for iSeries Programming Concepts* information. 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 which are not supported by RPG for iSeries, an INCLUDE SQLDA statement cannot be specified in an RPG for iSeries program. An SQLDA must be set up by a C, COBOL, PL/I, or ILE RPG program and passed to the RPG program in order to use it.

For more information about SQLDA, see SQL Description Area in the *SQL Reference* book.

Embedding SQL statements in RPG for iSeries applications that use SQL

SQL statements coded in an RPG for iSeries 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 for iSeries subroutine. The SQL statements are executed based on the logic of the RPG for iSeries 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.

For more details, see the following sections:

- "Example: Embedding SQL statements in RPG for iSeries applications that use SQL" on page 92
- "Comments in RPG for iSeries applications that use SQL" on page 92
- "Continuation for SQL statements in RPG for iSeries applications that use SQL" on page 92
- "Including code in RPG for iSeries applications that use SQL" on page 92
- "Sequence numbers in RPG for iSeries applications that use SQL" on page 92
- "Names in RPG for iSeries applications that use SQL" on page 92
- "Statement labels in RPG for iSeries applications that use SQL" on page 93
- "WHENEVER statement in RPG for iSeries applications that use SQL" on page 93

Example: Embedding SQL statements in RPG for iSeries applications that use SQL

An UPDATE statement coded in an RPG for iSeries program might be coded as follows:

```
*...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 for iSeries applications that use SQL

In addition to SQL comments (--), RPG for iSeries comments can be included within SQL statements wherever a blank is allowed, except between the keywords EXEC and SQL. To embed an RPG for iSeries comment within the SQL statement, place an asterisk (*) in position 7.

Continuation for SQL statements in RPG for iSeries 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 for iSeries applications that use SQL

SQL statements and RPG for iSeries 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 for iSeries specifications.

Sequence numbers in RPG for iSeries 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 for iSeries applications that use SQL

Any valid RPG 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 'SQ', 'SQL', 'RDI', or 'DSN'. These names are reserved for the database manager.

Statement labels in RPG for iSeries 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 for iSeries 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 for iSeries applications that use SQL

All host variables used in SQL statements must be explicitly declared. LOB and ROWID host variables are not supported in RPG for iSeries.

SQL embedded in RPG for iSeries 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.

For more details, see “Declaring host variables in RPG for iSeries applications that use SQL”.

Declaring host variables in RPG for iSeries applications that use SQL

The SQL RPG for iSeries precompiler only recognizes a subset of RPG for iSeries declarations as valid host variable declarations.

All variables defined in RPG for iSeries can be used in SQL statements, except for 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 for iSeries. 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 for iSeries applications that use SQL

The RPG for iSeries 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 list of subfield names making 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 for iSeries 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 for iSeries 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 for iSeries applications that use SQL

The SQL precompiler processes the RPG for iSeries source in much the same manner as the ILE RPG for iSeries 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 ILE RPG for iSeries 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 for iSeries 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 which 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 which 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 for iSeries, 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.

For another example, see "External file description considerations for host structure arrays in RPG for iSeries applications that use SQL".

External file description considerations for host structure arrays in RPG for iSeries applications that use SQL

If the file contains floating-point fields, it cannot be used as a host structure array. 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 not be contiguous.

In the following example, the DEPARTMENT table is included in the RPG for iSeries 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 for iSeries data types

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

Table 7. RPG for iSeries Declarations Mapped to Typical SQL Data Types

RPG for iSeries Data Type	Col 43	Col 52	Other RPG for iSeries 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
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 for iSeries data type that is equivalent to a given SQL data type.

Table 8. SQL Data Types Mapped to Typical RPG for iSeries Declarations

SQL Data Type	RPG for iSeries 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.
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.
BLOB	Not supported	Not supported
CLOB	Not supported	Not supported
GRAPHIC(n)	Not supported	Not supported
VARGRAPHIC(n)	Not supported	Not supported
DBCLOB	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.

Table 8. SQL Data Types Mapped to Typical RPG for iSeries Declarations (continued)

SQL Data Type	RPG for iSeries Data Type	Notes
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.
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

For more information, see "Notes on RPG for iSeries variable declaration and usage in RPG for iSeries applications that use SQL".

Notes on RPG for iSeries variable declaration and usage in RPG for iSeries applications that use SQL

Assignment rules in RPG for iSeries applications that use SQL

RPG for iSeries associates precision and scale with all numeric types. RPG for iSeries 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 for iSeries 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 97).

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.

See the indicator variables topic in the SQL Reference book for more information.

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.

For an example of using indicator variables, see "Example: Using indicator variables in RPG for iSeries applications that use SQL" on page 100.

Example: Using indicator variables in RPG for iSeries applications that use SQL

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 for iSeries because of structure parameter passing techniques

The SQL RPG for iSeries 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.

For more information about the structure parameter passing technique, see Database application design tips: Use structure parameter passing techniques in the *DB2 UDB for iSeries Database Performance and Query Optimization* information.

Correctly ending a called RPG for iSeries program that uses SQL

SQL run time builds and maintains data areas (internal SQLDAs) for each SQL statement which 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 for iSeries program containing SQL is called from another program which also contains SQL, the RPG for iSeries 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 for iSeries program is run. Re-initializing the static storage causes the internal SQLDAs to be rebuilt, thus causing a performance degradation.

An RPG for iSeries 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.

Chapter 6. Coding SQL Statements in ILE RPG for iSeries Applications

This chapter describes the unique application and coding requirements for embedding SQL statements in an ILE RPG for iSeries program. The coding requirements for host variables are defined.

For more details, see the following sections:

For a detailed ILE RPG program that shows how SQL statements can be used, see "Example: SQL Statements in ILE RPG for iSeries Programs" on page 173.

Note: See "Code disclaimer information" on page viii information for information pertaining to code examples.

For more information about programing using ILE RPG, see the ILE RPG Programmer's Guide book and the ILE RPG Reference book.

Defining the SQL Communications Area in ILE RPG for iSeries applications that use SQL

The SQL precompiler automatically places the SQLCA in the definition specifications of the ILE RPG for iSeries 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 ILE RPG for iSeries:

```
D*      SQL Communications area
D SQLCA           DS
D  SQLAID                 1      8A
D  SQLABC                 9     12B 0
D  SQLCOD                13     16B 0
D  SQLERL                17     18B 0
D  SQLERM                19     88A
D  SQLERP                89     96A
D  SQLERRD               97    120B 0 DIM(6)
D  SQLERR                97    120A
D   SQLER1               97    100B 0
D   SQLER2              101    104B 0
D   SQLER3              105    108B 0
D   SQLER4              109    112B 0
D   SQLER5              113    116B 0
D   SQLER6              117    120B 0
D  SQLWRN               121    131A
D   SQLWN0              121    121A
D   SQLWN1              122    122A
D   SQLWN2              123    123A
D   SQLWN3              124    124A
D   SQLWN4              125    125A
D   SQLWN5              126    126A
D   SQLWN6              127    127A
D   SQLWN7              128    128A
D   SQLWN8              129    129A
D   SQLWN9              130    130A
D   SQLWNA              131    131A
D  SQLSTT               132    136A
D*  End of SQLCA
```

Note: Variable names in RPG for iSeries for are limited to 6 characters. The standard SQLCA names were changed to a length of 6 for RPG for iSeries. To maintain compatibility with RPG for iSeries programs which are converted to ILE RPG for iSeries, the names for the SQLCA will remain as used with RPG for iSeries. The SQLCA defined for the ILE RPG for iSeries has added the field SQLERRD which is defined as an array of six integers. SQLERRD is defined to overlay the SQLERR definition.

For more information about SQLCA, see SQL Communication Area in the *SQL Reference* book.

Defining SQL Descriptor Areas in ILE RPG for iSeries applications that use SQL

The following statements require 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*

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 described in the SQL programmers guide. 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.

An INCLUDE SQLDA statement can be specified in an ILE RPG for iSeries program. The format of the statement is:

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....8.
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/reference the part of the SQLDA which 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 does a MOVEA of 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 does a MOVEA of SQL_VAR,n to SQLVAR where n is the number of the field description to be processed.

For more information about SQLDA, see SQL Descriptor Area in the *SQL Reference* book.

Embedding SQL statements in ILE RPG for iSeries 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 an RPG subroutines. The SQL statements are executed based on the logic of the RPG 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 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.

Both uppercase and lowercase letters are acceptable in SQL statements.

An UPDATE statement coded in an ILE RPG for iSeries program might be coded as follows:

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....8.
C/EXEC SQL UPDATE DEPARTMENT
C+            SET MANAGER = :MGRNUM
C+            WHERE DEPTNO = :INTDEP
C/END-EXEC
```

For more details, see the following sections:

- “Comments in ILE RPG for iSeries applications that use SQL” on page 107
- “Continuation for SQL statements in ILE RPG for iSeries applications that use SQL” on page 107
- “Including code in ILE RPG for iSeries applications that use SQL” on page 107
- “Using directives in ILE RPG for iSeries applications that use SQL” on page 107
- “Sequence numbers in ILE RPG for iSeries applications that use SQL” on page 107
- “Names in ILE RPG for iSeries applications that use SQL” on page 108
- “Statement labels in ILE RPG for iSeries applications that use SQL” on page 108
- “WHENEVER statement in ILE RPG for iSeries applications that use SQL” on page 108

For information on locking rows between a SELECT and an UPDATE statement, see Commitment control in the *SQL Programming Concepts* book.

Comments in ILE RPG for iSeries applications that use SQL

In addition to SQL comments (--), ILE RPG for iSeries comments can be included within SQL statements wherever SQL allows a blank character. To embed an ILE RPG for iSeries comment within the SQL statement, place an asterisk (*) in position 7.

Continuation for SQL statements in ILE RPG for iSeries applications that use SQL

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>'.

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

Including code in ILE RPG for iSeries applications that use SQL

SQL statements and RPG calculation specifications can be included by using the SQL statement:

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

The RPG /COPY directive can be used to include SQL statements or RPG specifications. Nested /COPY statements are not supported by the precompiler. The RPG /INCLUDE directive is not recognized by the precompiler. It can be used to include RPG code that doesn't need to be processed by SQL. This can be useful for code that contains conditional directives and for nesting in other /COPY blocks.

Using directives in ILE RPG for iSeries applications that use SQL

Directives other than /COPY are ignored by the SQL precompiler. They are passed along to the compiler to be processed. This means that all RPG and SQL statements within conditional logic blocks will be processed unconditionally by the precompiler.

Sequence numbers in ILE RPG for iSeries 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 for iSeries applications that use SQL

Any valid ILE RPG for iSeries 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 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.

Statement labels in ILE RPG for iSeries 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 for iSeries 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 for iSeries applications that use SQL

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

SQL embedded in ILE RPG for iSeries 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.

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

For more details, see “Declaring host variables in ILE RPG for iSeries applications that use SQL”.

Declaring host variables in ILE RPG for iSeries applications that use SQL

The SQL ILE RPG for iSeries precompiler only recognizes a subset of valid ILE RPG for iSeries declarations as valid host variable declarations.

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

- 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 for iSeries. 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. 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.

Using host structures in ILE RPG for iSeries applications that use SQL

The ILE RPG for iSeries 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 list of subfield names making 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. While this support does not provide additional function since a field can be defined with a maximum length of 32766, it is required for compatibility with RPG for iSeries programs.

In the following example, BIGCHR is an ILE RPG for iSeries 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....+....8
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. The referral 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.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
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, 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 ILE RPG for iSeries 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 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, 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 DEPT and a blocked FETCH statement to retrieve 10 rows from the DEPARTMENT table.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
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
```

Declaring LOB host variables in ILE RPG for iSeries applications that use SQL

ILE RPG for iSeries 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 for iSeries language structure in the output source member. LOB declarations can be either standalone or within a data structure.

For more details, see the following sections:
- "LOB host variables in ILE RPG for iSeries applications that use SQL"
- "LOB locators in ILE RPG for iSeries applications that use SQL" on page 112
- "LOB file reference variables in ILE RPG for iSeries applications that use SQL" on page 112

LOB host variables in ILE RPG for iSeries applications that use SQL

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
```

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
```

Notes:
1. For BLOB, CLOB, 1 <= lob-length <= 32,766
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 standalone host variable, position 24 must contain the character ’S’ and position 25 must be blank.
9. The standalone 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 for iSeries applications that use SQL

BLOB Locator Example

The following declaration:

```
D MYBLOB          S          SQLTYPE(BLOB_LOCATOR)
```

Results in the following generation:

```
D MYBLOB          S          10U
```

CLOB and DBCLOB locators have similar syntax.

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 for iSeries applications that use SQL

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. See LOB file reference variables in the SQL Programming Concepts book for more information about these values.
- SQFRD (2)
- SQFCRT (8)
- SQFOVR (16)
- SQFAPP (32)

ROWID variables in ILE RPG for iSeries applications that use SQL

ILE RPG for iSeries 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 for iSeries 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 external file descriptions in ILE RPG for iSeries applications that use SQL

The SQL precompiler processes the ILE RPG for iSeries source in much the same manner as the ILE RPG for iSeries 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.

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 RPG for iSeries precompiler.

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

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
FDEPARTMENTIP   E             DISK    RENAME(ORIGREC:DEPTREC)
```

Note: Code an F-spec for a file in your ILE RPG for iSeries program only if you use ILE RPG for iSeries 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.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
DTDEPT          E DS                  EXTNAME(DEPARTMENT)
D DEPTN         E                     EXTFLD(DEPTNAME)
D ADMRD         E                     EXTFLD(ADMRDEPT)
```

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

For more details, see “External file description considerations for host structure arrays in ILE RPG for iSeries applications that use SQL”.

External file description considerations for host structure arrays in ILE RPG for iSeries 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 for iSeries 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.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
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 RPG data types

The precompiler will determine the base SQLTYPE and SQLLEN of host variables according to the following table. If a host variable appears with an indicator variable, the SQLTYPE is the base SQLTYPE plus one.

Table 9. ILE RPG for iSeries Declarations Mapped to Typical SQL Data Types

RPG Data Type	D spec Pos 40	D spec Pos 41,42	Other RPG Coding	SQLTYPE of Host Variable	SQLLEN of Host Variable	SQL Data Type
Data structure (without subfields)	blank	blank	Length = n where n ≤ 32766	452	n	CHAR(n)
Calculation result field (pos 69,70 = blank)	n/a	n/a	Length = n where n ≤ 32766 (pos 59-63)	452	n	CHAR(n)
Definition specification	A	blank	length=n where n is 1 to 254. VARYING in columns 44-80.	448	n	VARCHAR (n)
Definition specification	A	blank	length=n where n > 254. VARYING in columns 44-80	456	n	VARCHAR (n)
Definition specification	B	0	Length ≤ 4	500	2	SMALLINT
Definition specification	I	0	Length = 5	500	2	SMALLINT
Definition specification	B	0	Length ≤ 9 and ≥ 5	496	4	INTEGER
Definition specification	I	0	Length = 10	496	4	INTEGER
Definition specification	I	0	Length = 20	492	8	BIGINT
Definition specification	B	1-4	Length = 2	500	2	DECIMAL(4,s) s=col 41, 42
Definition specification	B	1-9	Length = 4	496	4	DECIMAL(9,s) s=col 41, 42
Definition specification	P	0 to 30	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 = pos 41, 42

Table 9. ILE RPG for iSeries Declarations Mapped to Typical SQL Data Types (continued)

RPG Data Type	D spec Pos 40	D spec Pos 41,42	Other RPG Coding	SQLTYPE of Host Variable	SQLLEN of Host Variable	SQL Data Type
Definition specification	F	blank	Length = 4	480	4	FLOAT (single precision)
Definition specification	F	blank	Length = 8	480	8	FLOAT (double precision)
Definition specification not a subfield	blank	0 to 30	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 = pos 41, 42
Input field (pos 36 = P)	n/a	n/a	Length = n where n is 1 to 16 (pos 37-46)	484	p in byte 1, s in byte 2	DECIMAL(p,s) where p = n*2-1 and s = pos 47, 48
Input field (pos 36 = blank or S)	n/a	n/a	Length = n where n is 1 to 30 (pos 37-46)	484	p in byte 1, s in byte 2	DECIMAL(p,s) where p = n and s = pos 47, 48
Input field (pos 36 = B)	n/a	n/a	Length = n where n is 2 or 4 (pos 37-46)	484	p in byte 1, s in byte 2	DECIMAL(p,s) where p=4 if n=2 or 9 if n=4 s = pos 47, 48
Calculation result field (pos 69,70 ≠ blank)	n/a	n/a	Length = n where n is 1 to 30 (pos 59-63)	484	p in byte 1, s in byte 2	DECIMAL(p,s) where p = n and s = pos 64, 65
Data Structure subfield	blank	0 to 30	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 = pos 41, 42
Definition specification	S	0 to 30	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 = pos 41, 42
Input field (pos 36 = G)	n/a	n/a	Length = n where n is 1 to 32766 (pos 37-46)	468	m	GRAPHIC(m) where m = n/2 m = (TO-FROM-1)/2
Definition specification	G	blank	length=n where n is 1 to 127. VARYING in columns 44-80.	464	n	VARGRAPHIC (n)
Defintion specification	C	blank	length=n where n<16383	468	n	GRAPHIC(n) with CCSID 13488
Definition specification	G	blank	length=n where n > 127. VARYING in columns 44-80.	472	n	VARGRAPHIC (n)
Definition specification	D	blank	Length = n where n is 6, 8 or 10	384	n	DATE (DATFMT, DATSEP specified in pos 44-80)
Input field (pos 36 = D)	n/a	n/a	Length = n where n is 6, 8, or 10 (pos 37-46)	384	n	DATE (format specified in pos 31-34)

Table 9. ILE RPG for iSeries Declarations Mapped to Typical SQL Data Types (continued)

RPG Data Type	D spec Pos 40	D spec Pos 41,42	Other RPG Coding	SQLTYPE of Host Variable	SQLLEN of Host Variable	SQL Data Type
Definition specification	T	blank	Length = n where n is 8	388	n	TIME (TIMFMT, TIMSEP specified in pos 44-80)
Input field (pos 36 = T)	n/a	n/a	Length = n where n is 8 (pos 37-46)	388	n	TIME (format specified in pos 31-34)
Definition specification	Z	blank	Length = n where n is 26	392	n	TIMESTAMP
Input field (pos 36 = Z)	n/a	n/a	Length = n where n is 26 (pos 37-46)	392	n	TIMESTAMP

Notes:

1. In the first column the term ″definition specification″ includes data structure subfields unless explicitly stated otherwise.
2. In definition specifications the length of binary fields (B in pos 40) is determined by the following:
   - FROM (pos 26-32) is not blank, then length = TO-FROM+1.
   - FROM (pos 26-32) is blank, then length = 2 if pos 33-39 < 5, or length = 4 if pos 33-39 > 4.
3. SQL will create the date/time subfield using the DATE/TIME format specified on the CRTSQLRPGI command. The conversion to the host variable DATE/TIME format will occur 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.
BLOB	Not supported	Use SQLTYPE keyword to declare a BLOB.
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.

Table 10. SQL Data Types Mapped to Typical RPG Declarations (continued)

SQL Data Type	RPG Data Type	Notes
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.

For more details, see “Notes on ILE RPG for iSeries variable declaration and usage”.

Notes on ILE RPG for iSeries variable declaration and usage

Assignment rules in ILE RPG for iSeries applications that use SQL

ILE RPG for iSeries associates precision and scale with all numeric types. ILE RPG for iSeries 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 for iSeries applications that use SQL

An indicator variable is a binary field with length less then 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.

See the indicator variables topic in the SQL Reference book for more information.

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.

For an example of using indicator variables in ILE RPG, see "Example: Using indicator variables in ILE RPG for iSeries applications that use SQL".

Example: Using indicator variables in ILE RPG for iSeries applications that use SQL

Given the statement:

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
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....+....8
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 of the SQLDA for a multiple row-area fetch in ILE RPG for iSeries applications that use SQL

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....8.
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 of dynamic SQL in an ILE RPG for iSeries application that uses SQL

```
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
```

Chapter 7. Coding SQL Statements in REXX Applications

REXX procedures do not have to be preprocessed. At runtime, 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 details, see the following sections:

- "Using the SQL Communications Area in REXX applications"
- "Using SQL Descriptor Areas in REXX applications" on page 124
- "Embedding SQL statements in REXX applications" on page 125
- "Using host variables in REXX applications that use SQL" on page 128
- "Using indicator variables in REXX applications that use SQL" on page 130

For more information about the STRREXPRC CL command or the ADDRESS REXX command, see the REXX/400 Programmer's Guide book and the REXX/400 Reference book.

For a detailed sample REXX program that shows how SQL statements can be used, see "Example: SQL Statements in REXX Programs" on page 179.

Note: See "Code disclaimer information" on page viii information for information pertaining to code examples.

Using the SQL Communications Area in REXX applications

The fields that make up the SQL Communications 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 executed. 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.

For more information about SQLCA, see SQL Communication Area in the *SQL Reference* book.

Using SQL Descriptor Areas in REXX applications

The following statements require 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.

For more information about SQLDA, see SQL Descriptor Area in the SQL Reference book.

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
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.

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 quotes, 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 quotes, for example:

```
'EXECSQL COMMIT'
```

The SQL/REXX interface supports the following SQL statements:

ALTER TABLE
CALL [3]
CLOSE
COMMENT ON
COMMIT
CREATE ALIAS
CREATE DISTINCT TYPE
CREATE FUNCTION
CREATE INDEX
CREATE PROCEDURE
CREATE SCHEMA
CREATE TABLE
CREATE TRIGGER
CREATE VIEW
DECLARE CURSOR [3]
DECLARE GLOBAL TEMPORARY TABLE
DELETE [3]
DESCRIBE
DESCRIBE TABLE
DROP
EXECUTE
EXECUTE IMMEDIATE
FETCH [2]
GRANT
INSERT [2], [3]
LABEL ON
LOCK TABLE
OPEN
PREPARE
RELEASE SAVEPOINT
RENAME
REVOKE
ROLLBACK
SAVEPOINT
SET OPTION [4]
SET PATH
SET SCHEMA
SET TRANSACTION
SET variable [3]
UPDATE [3]
VALUES INTO [3]

The following SQL statements are not supported by the SQL/REXX interface:

BEGIN DECLARE SECTION
CONNECT
DECLARE PROCEDURE
DECLARE STATEMENT
DECLARE VARIABLE
DISCONNECT
END DECLARE SECTION
FREE LOCATOR
HOLD LOCATOR
INCLUDE
RELEASE
SELECT INTO
SET CONNECTION
SET RESULT SETS
WHENEVER[5]

2. The blocked form of this statement is not supported.

3. These statements cannot be executed directly if they contain host variables; they must be the object of a PREPARE and then an EXECUTE.

4. 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 book for more information about the SET OPTION statement.

For more details, see the following sections:

- "Comments in REXX applications that use SQL"
- "Continuation of SQL statements in REXX applications that use SQL"
- "Including code in REXX applications that use SQL"
- "Margins in REXX applications that use SQL"
- "Names in REXX applications that use SQL"
- "Nulls in REXX applications that use SQL"
- "Statement labels in REXX applications that use SQL"
- "Handling errors and warnings in REXX applications that use SQL" on page 128

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.

5. See "Handling errors and warnings in REXX applications that use SQL" on page 128 for more information.

Handling errors and warnings in REXX applications that use SQL

The WHENEVER statement is not supported by the SQL/REXX interface. 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 and ROWID 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.

For more details, see the following sections:

- "Determining data types of input host variables in REXX applications that use SQL"
- "The format of output host variables in REXX applications that use SQL" on page 130
- "Avoiding REXX conversion in REXX applications that use SQL" on page 130

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 Table 11 on page 129.

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
Undefined Variable	Variable for which a value has not been assigned	None	Data that is not valid was detected.
A string with leading and trailing apostrophes (′) or quotation marks (″), which has length n after removing the two delimiters, or a string with a leading X or x followed by an apostrophe (′) or quotation mark (″), and a trailing apostrophe (′) 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)
A string with a leading and trailing apostrophe (′) or quotation marks (″) preceded by: [6] • A string that starts with a G, g, N or n. This is followed by an apostrophe or quote 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 an apostrophe or quote (whichever the string started with). • A string with a leading GX, Gx, gX, or gx, followed by an apostrophe or quote 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 an apostrophe or quote (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)
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	FLOAT
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 with neither decimal point nor exponent. It can have a leading plus or minus sign.	Signed integers	496/497	INTEGER

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 the following: "'". Simply enclosing the string in apostrophes does not work. For example:

```
stringvar = '100'
```

causes REXX to set the variable *stringvar* to the string of characters 100 (without the apostrophes). 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 apostrophes). 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.

See the indicator variables topic in the SQL Reference book for more information.

6. The byte immediately following the leading apostrophe is a X'0E' shift-out, and the byte immediately preceding the trailing apostrophe is a X'0F' shift-in.

Chapter 8. Preparing and Running a Program with SQL Statements

This chapter describes some of the tasks for preparing and running an application program. For more details, see the following sections:

Note: See "Code disclaimer information" on page viii information for information pertaining to code examples.

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. For more information, see the Database Programming book, and the File Management book.
  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. For more information about any SQL statement, see the SQL Reference book.
  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:
  - QSQROUTE
  - QSQLOPEN
  - QSQLCLSE
  - QSQLCMIT

  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)

For more details, see the following sections:
- "Input to the SQL precompiler"
- "Source file CCSIDs in the SQL precompiler" on page 133
- "Output from the SQL precompiler" on page 133

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 [7]. The SQL INCLUDE statement causes input to be read from the specified member until it reaches the end of the member. The included member may not contain other precompiler INCLUDE statements, but can contain both application program and SQL statements.

Another preprocessor may process source statements before the SQL precompiler. However, any preprocessor run before the SQL precompile must be able to pass through SQL statements.

---

7. The xxx in this command refers to the host language indicators: CBL for the COBOL for iSeries language, CBLI for the ILE COBOL for iSeries language, PLI for the iSeries PL/I language, CI for the ILE C for iSeries language, RPG for the RPG for iSeries language, RPGI for the ILE RPG for iSeries language, CPPI for the ILE C++/400 language.

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. See the SQL Reference book for the SET OPTION syntax.

Source file CCSIDs in the SQL precompiler

The SQL precompiler will read the source records by using the CCSID of the source file. When processing SQL INCLUDE statements, the include source will be 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 will occur.

The SQL precompiler will process 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 following sections describe the various kinds of output supplied by the precompiler.

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 runtime. Includes 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 for iSeries) 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 V5R2M0 020719         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............V5R2M0
INCLUDE file..............*LIBL/*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
Package name..............*PGMLIB/*PGM
Path......................*NAMING
Created object type.......*PGM
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
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 (Part 1 of 4)

```
5722ST1 V5R2M0 020719           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
    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
```

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.

Figure 2. Sample COBOL Precompiler Output Format (Part 2 of 4)

```
5722ST1 V5R2M0 020719            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
   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  * * * * *
```

Figure 2. Sample COBOL Precompiler Output Format (Part 3 of 4)

```
5722ST1 V5R2M0 0020719          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.

Figure 2. Sample COBOL Precompiler Output Format (Part 4 of 4)

Non-ILE SQL precompiler commands

DB2 UDB Query Manager and SQL Development Kit includes non-ILE precompiler commands for the following host languages: CRTSQLCBL (for COBOL for iSeries), CRTSQLPLI (for iSeries PL/I), and CRTSQLRPG (for RPG III, which is part of RPG for iSeries). 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. Refer to Appendix B, "DB2 UDB for iSeries CL Command Descriptions for Host Language Precompilers" on page 185 for more information.

For more details, see "Compiling a non-ILE application program that uses SQL".

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, the following ILE precompiler commands exist: CRTSQLCI, CRTSQLCBLI, CRTSQLRPGI, and CRTSQLCPPI. There is a precompiler command for each of the host languages: ILE C for iSeries, ILE COBOL for iSeries, and ILE RPG for iSeries. 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. Refer to Appendix B, "DB2 UDB for iSeries CL Command Descriptions for Host Language Precompilers" on page 185 for more information.

For more details, see the following sections:
- "Compiling an ILE application program that uses SQL"

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 create *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 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.

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.

Interpreting compile errors in applications that use SQL

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++
- 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.

For more details, see “Error and warning messages during a compile of application programs that use SQL” on page 142.

Error and warning messages during a compile of application programs that use SQL

The conditions described in the following paragraphs could produce an error or warning message during an attempted compile process.

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 an 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 Chapter 2, "Coding SQL Statements in C and C++ Applications", through Chapter 7, "Coding SQL Statements in REXX Applications".

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 was 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.

For example, if a program contains an SQL statement that refers to COLUMNA in TABLEA and the user deletes and recreates 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.

For more details, see "Program references in applications that use SQL".

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. Type:

```
CALL pgm-name
```

on the system command line. For more information about running programs, see CL Programming .

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.

For more details, see the following sections:

- "Running a program with embedded SQL: OS/400 DDM considerations"
- "Running a program with embedded SQL: override considerations"
- "Running a program with embedded SQL: SQL return codes" on page 145

Running a program with embedded SQL: OS/400 DDM considerations

SQL does not support remote file access through DDM (distributed data management) files. SQL does support remote access through DRDA® (Distributed Relational Database Architecture™.)

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.

For more information about overrides, see the Database Programming book, and the File Management book.

Running a program with embedded SQL: SQL return codes

A list of SQL return codes is provided in SQL Messages and Codes topic in the iSeries Information Center.

Appendix A. Sample Programs Using DB2 UDB for iSeries Statements

This appendix contains a sample application showing how to code SQL statements in each of the languages supported by the DB2 UDB for iSeries system.

Note: See "Code disclaimer information" on page viii information for information pertaining to code examples.

Examples of programs that use SQL statements

Programs that provide examples of how to code SQL statements with host languages are provided for the following programming languages:

- ILE C and C++
- COBOL and ILE COBOL
- PL/I
- RPG for iSeries
- ILE RPG for iSeries
- REXX

The sample application gives raises based on commission.

Each sample program produces the same report, which is shown at the end of this appendix. 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 upper 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 for iSeries, 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 executed 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.

Example: SQL Statements in ILE C and C++ Programs

This sample 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: See "Code disclaimer information" on page viii information for information pertaining to code examples.

```
5722ST1 V5R2M0 020719            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............V5R2M0
INCLUDE file..............*LIBL/*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
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
Source member changed on 06/06/00  17:15:17

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
```

Figure 3. Sample C Program Using SQL Statements (Part 1 of 5)

```
Record  *...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8   SEQNBR  Last change
   short work_days  = 253;          /* work days during in one year */
   float commission = 2000.00;      /* cutoff to qualify for raise  */
   float percentage = 1.04;         /* raised salary as percentage  */
   char raise_date??(12??) = "1982-06-01"; /*  effective raise date */

   /* File declaration for qprint */
   FILE *qprint;

   /* Structure for report 1 */
   1 #pragma mapinc ("project","CORPDATA/PROJECT(PROJECT)","both","p z")
   #include "project"
   struct {
           CORPDATA_PROJECT_PROJECT_both_t Proj_struct;
           char  empno??(7??);
           char  name??(30??);
           float salary;
           } rpt1;

   /* Structure for report 2 */
   struct {
           char projno??(7??);
           char project_name??(37??);
           short employee_count;
           double total_proj_cost;
          } rpt2;

   2 exec sql include SQLCA;

   qprint=fopen("QPRINT","w");

   /* Update the selected projects by the new percentage. If an error */
   /* occurs during the update, ROLLBACK the changes.                */
   3 EXEC SQL WHENEVER SQLERROR GO TO update_error;
   4 EXEC SQL
       UPDATE CORPDATA/EMPLOYEE
          SET SALARY = SALARY * :percentage
          WHERE COMM >= :commission ;

   /* Commit changes */
   5 EXEC SQL
       COMMIT;
   EXEC SQL WHENEVER SQLERROR GO TO report_error;

   /* Report the updated statistics for each employee assigned to the */
   /* selected projects.                                              */

   /* Write out the header for Report 1 */
   fprintf(qprint,"                    REPORT OF PROJECTS AFFECTED \
BY RAISES");
   fprintf(qprint,"\n\nPROJECT  EMPID     EMPLOYEE NAME    ");
   fprintf(qprint,  "                  SALARY\n");

   6 exec sql
       declare c1 cursor for
         select distinct projno, empprojact.empno,
                lastname||', '||firstnme, salary
         from corpdata/empprojact, corpdata/employee
         where empprojact.empno = employee.empno and comm >= :commission
         order by projno, empno;
   7 EXEC SQL
       OPEN C1;

   /* Fetch and write the rows to QPRINT */
   8 EXEC SQL WHENEVER NOT FOUND GO TO done1;

   do {
   10 EXEC SQL
         FETCH C1 INTO :Proj_struct.PROJNO, :rpt1.empno,
                       :rpt1.name,:rpt1.salary;
     fprintf(qprint,"\n%6s   %6s    %-30s    %8.2f",
             rpt1.Proj_struct.PROJNO,rpt1.empno,
             rpt1.name,rpt1.salary);
     }
   while (SQLCODE==0);

 done1:
   EXEC SQL
        CLOSE C1;
```

Figure 3. Sample C Program Using SQL Statements (Part 2 of 5)

```
Record  *...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8   SEQNBR  Last change

  /* For all projects ending at a date later than the 'raise_date'   * /
  /* (i.e. 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.                               */

  /* Write out the header for Report 2 */
  fprintf(qprint,"\n\n\n                    ACCUMULATED STATISTICS\
BY PROJECT");
  fprintf(qprint,  "\n\nPROJECT                                 \
   NUMBER OF       TOTAL");
  fprintf(qprint,    "\nNUMBER   PROJECT NAME                     \
   EMPLOYEES       COST\n");

11  EXEC SQL
       DECLARE C2 CURSOR FOR
         SELECT EMPPROJACT.PROJNO, PROJNAME, COUNT(*),
            SUM ( ( DAYS(EMENDATE) - DAYS(EMSTDATE) ) * EMPTIME *
                  (DECIMAL( SALARY / :work_days ,8,2)))
         FROM CORPDATA/EMPPROJACT, CORPDATA/PROJECT, CORPDATA/EMPLOYEE
         WHERE EMPPROJACT.PROJNO=PROJECT.PROJNO  AND
               EMPPROJACT.EMPNO =EMPLOYEE.EMPNO  AND
               PRENDATE > :raise_date
         GROUP BY EMPPROJACT.PROJNO, PROJNAME
         ORDER BY 1;
  EXEC SQL
       OPEN C2;

  /* Fetch and write the rows to QPRINT */
  EXEC SQL WHENEVER NOT FOUND GO TO done2;

  do {
   12 EXEC SQL
         FETCH C2 INTO :rpt2;
    fprintf(qprint,"\n%6s   %-36s  %6d       %9.2f",
            rpt2.projno,rpt2.project_name,rpt2.employee_count,
            rpt2.total_proj_cost);
  }
  while (SQLCODE==0);

done2:
  EXEC SQL
       CLOSE C2;
  goto finished;

  /* Error occured while updating table.  Inform user and rollback   */
  /* changes.                                                        */
update_error:
 13 EXEC SQL WHENEVER SQLERROR CONTINUE;
  fprintf(qprint,"*** ERROR Occurred while updating table.  SQLCODE="
          "%5d\n",SQLCODE);
 14 EXEC SQL
       ROLLBACK;
  goto finished;

  /* Error occured while generating reports.  Inform user and exit.  */
report_error:
  fprintf(qprint,"*** ERROR Occurred while generating reports.  "
          "SQLCODE=%5d\n",SQLCODE);
  goto finished;

  /* All done  */
finished:
  fclose(qprint);
  exit(0);

}
* * * * *  E N D  O F  S O U R C E  * * * * *
```

Figure 3. Sample C Program Using SQL Statements (Part 3 of 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
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
```

Figure 3. Sample C Program Using SQL Statements (Part 4 of 5)

```
CROSS REFERENCE

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  * * * * *
```

Figure 3. Sample C Program Using SQL Statements (Part 5 of 5)

Example: SQL Statements in COBOL and ILE COBOL Programs

Note: See "Code disclaimer information" on page viii information for information pertaining to code examples.

```
5722ST1 V5R2M0 020719           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............V5R2M0
INCLUDE file..............*LIBL/*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
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
Source member changed on 07/01/96  09:44:58

              ****************************************************************
              * 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 showing the projects    *
              * which these employees have contributed to ordered by the     *
              * project number and employee ID. A second report shows each   *
              * project having an end date occurring  after RAISE-DATE       *
              * (i.e. potentially affected by the retroactive raises ) with  *
              * its total salary expenses and a count of employees who       *
              * contributed to the project.                                  *
              ****************************************************************


               IDENTIFICATION DIVISION.

               PROGRAM-ID.  CBLEX.
               ENVIRONMENT DIVISION.
               CONFIGURATION SECTION.
               SOURCE-COMPUTER. IBM-AS400.
               OBJECT-COMPUTER. IBM-AS400.
               INPUT-OUTPUT SECTION.

               FILE-CONTROL.
                   SELECT PRINTFILE ASSIGN TO PRINTER-QPRINT
                      ORGANIZATION IS SEQUENTIAL.

               DATA DIVISION.

```

Figure 4. Sample COBOL Program Using SQL Statements (Part 1 of 7)

```
5722ST1 V5R2M0 020719           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
               FILE SECTION.

               FD  PRINTFILE
                   BLOCK CONTAINS 1 RECORDS
                   LABEL RECORDS ARE OMITTED.
               01  PRINT-RECORD PIC X(132).

               WORKING-STORAGE SECTION.
               77  WORK-DAYS PIC S9(4) BINARY VALUE 253.
               77  RAISE-DATE PIC X(11) VALUE "1982-06-01".
               77  PERCENTAGE PIC S999V99 PACKED-DECIMAL.
               77  COMMISSION PIC S99999V99 PACKED-DECIMAL VALUE 2000.00.

              ***************************************************************
              *  Structure for report 1.                                    *
              ***************************************************************

           1    01  RPT1.
                   COPY DDS-PROJECT OF CORPDATA-PROJECT.
                   05  EMPNO     PIC X(6).
                   05  NAME      PIC X(30).
                   05  SALARY    PIC S9(6)V99 PACKED-DECIMAL.


              ***************************************************************
              *  Structure for report 2.                                    *
              ***************************************************************

               01  RPT2.
                   15  PROJNO PIC X(6).
                   15  PROJECT-NAME PIC X(36).
                   15  EMPLOYEE-COUNT PIC S9(4) BINARY.
                   15  TOTAL-PROJ-COST PIC S9(10)V99 PACKED-DECIMAL.

               2  EXEC SQL
                        INCLUDE SQLCA
                   END-EXEC.
               77  CODE-EDIT PIC ---99.

              ***************************************************************
              *  Headers for reports.                                       *
              ***************************************************************

               01  RPT1-HEADERS.
                   05  RPT1-HEADER1.
                       10  FILLER PIC X(21) VALUE SPACES.
                       10  FILLER PIC X(111)
                             VALUE "REPORT OF PROJECTS AFFECTED BY RAISES".
                   05  RPT1-HEADER2.
                       10  FILLER PIC X(9) VALUE "PROJECT".
                       10  FILLER PIC X(10) VALUE "EMPID".
                       10  FILLER PIC X(35) VALUE "EMPLOYEE NAME".
                       10  FILLER PIC X(40) VALUE "SALARY".
               01  RPT2-HEADERS.
                   05  RPT2-HEADER1.
                       10  FILLER PIC X(21) VALUE SPACES.
                       10  FILLER PIC X(111)
                               VALUE "ACCUMULATED STATISTICS BY PROJECT".
                   05  RPT2-HEADER2.
                       10  FILLER PIC X(9) VALUE "PROJECT".
                       10  FILLER PIC X(38) VALUE SPACES.
                       10  FILLER PIC X(16) VALUE "NUMBER OF".
                       10  FILLER PIC X(10) VALUE "TOTAL".
                   05  RPT2-HEADER3.
                       10  FILLER PIC X(9) VALUE "NUMBER".
                       10  FILLER PIC X(38) VALUE "PROJECT NAME".
                       10  FILLER PIC X(16) VALUE "EMPLOYEES".
                       10  FILLER PIC X(65) VALUE "COST".
               01  RPT1-DATA.
                   05  PROJNO    PIC X(6).
                   05  FILLER    PIC XXX VALUE SPACES.
                   05  EMPNO     PIC X(6).
                   05  FILLER    PIC X(4) VALUE SPACES.
                   05  NAME      PIC X(30).
                   05  FILLER    PIC X(3) VALUE SPACES.
                   05  SALARY    PIC ZZZZZ9.99.
                   05  FILLER    PIC X(96) VALUE SPACES.
```

Figure 4. Sample COBOL Program Using SQL Statements (Part 2 of 7)

```
5722ST1 V5R2M0 020719           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
       01  RPT2-DATA.
           05  PROJNO PIC X(6).
           05  FILLER PIC XXX VALUE SPACES.
           05  PROJECT-NAME PIC X(36).
           05  FILLER PIC X(4) VALUE SPACES.
           05  EMPLOYEE-COUNT PIC ZZZ9.
           05  FILLER PIC X(5) VALUE SPACES.
           05  TOTAL-PROJ-COST PIC ZZZZZZZZ9.99.
           05  FILLER PIC X(56) VALUE SPACES.

       PROCEDURE DIVISION.

       A000-MAIN.
           MOVE 1.04 TO PERCENTAGE.
           OPEN OUTPUT PRINTFILE.

      ***************************************************************
      * Update the selected employees by the new percentage. If an  *
      * error occurs during the update, ROLLBACK the changes,       *
      ***************************************************************

         3  EXEC SQL
                WHENEVER SQLERROR GO TO E010-UPDATE-ERROR
           END-EXEC.
         4  EXEC SQL
                UPDATE CORPDATA/EMPLOYEE
                  SET SALARY = SALARY * :PERCENTAGE
                  WHERE COMM >= :COMMISSION
           END-EXEC.

      ***************************************************************
      *  Commit changes.                                            *
      ***************************************************************

         5  EXEC SQL
                COMMIT
           END-EXEC.

           EXEC SQL
                WHENEVER SQLERROR GO TO E020-REPORT-ERROR
           END-EXEC.

      ***************************************************************
      *  Report the updated statistics for each employee receiving  *
      *  a raise and the projects that s/he participates in         *
      ***************************************************************

      ***************************************************************
      *  Write out the header for Report 1.                         *
      ***************************************************************

           write print-record from rpt1-header1
                 before advancing 2 lines.
           write print-record from rpt1-header2
                 before advancing 1 line.
         6  exec sql
                declare c1 cursor for
                  SELECT DISTINCT projno, empprojact.empno,
                          lastname||", "||firstnme ,salary
                  from corpdata/empprojact, corpdata/employee
                  where empprojact.empno =employee.empno and
                        comm >= :commission
                  order by projno, empno
           end-exec.
         7  EXEC SQL
                OPEN C1
           END-EXEC.

           PERFORM B000-GENERATE-REPORT1 THRU B010-GENERATE-REPORT1-EXIT
               UNTIL SQLCODE NOT EQUAL TO ZERO.

```

Note: 8 and 9 are located on Part 5 of this figure.

Figure 4. Sample COBOL Program Using SQL Statements (Part 3 of 7)

```
5722ST1 V5R2M0 020719           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
   10  A100-DONE1.
           EXEC SQL
                CLOSE C1
           END-EXEC.

      *************************************************************
      *  For all projects ending at a date later than the RAISE-  *
      *  DATE ( i.e. those projects potentially affected by the   *
      *  salary raises generate a report containing the project   *
      *  project number, project name, the count of employees     *
      *  participating in the project and the total salary cost   *
      *  for the project                                          *
      *************************************************************


      ***************************************************************
      *  Write out the header for Report 2.                         *
      ***************************************************************

           MOVE SPACES TO PRINT-RECORD.
           WRITE PRINT-RECORD BEFORE ADVANCING 2 LINES.
           WRITE PRINT-RECORD FROM RPT2-HEADER1
                 BEFORE ADVANCING 2 LINES.
           WRITE PRINT-RECORD FROM RPT2-HEADER2
                 BEFORE ADVANCING 1 LINE.
           WRITE PRINT-RECORD FROM RPT2-HEADER3
                 BEFORE ADVANCING 2 LINES.

           EXEC SQL
             11  DECLARE C2 CURSOR FOR
                  SELECT EMPPROJACT.PROJNO, PROJNAME, COUNT(*),
                         SUM ( (DAYS(EMENDATE)-DAYS(EMSTDATE)) *
                         EMPTIME * DECIMAL((SALARY / :WORK-DAYS),8,2))
                  FROM CORPDATA/EMPPROJACT, CORPDATA/PROJECT,
                       CORPDATA/EMPLOYEE
                  WHERE EMPPROJACT.PROJNO=PROJECT.PROJNO AND
                        EMPPROJACT.EMPNO =EMPLOYEE.EMPNO AND
                        PRENDATE > :RAISE-DATE
                  GROUP BY EMPPROJACT.PROJNO, PROJNAME
                  ORDER BY 1
           END-EXEC.
           EXEC SQL
                OPEN C2
           END-EXEC.

           PERFORM C000-GENERATE-REPORT2 THRU C010-GENERATE-REPORT2-EXIT
                UNTIL SQLCODE NOT EQUAL TO ZERO.

       A200-DONE2.
           EXEC SQL
                CLOSE C2
           END-EXEC

      ***************************************************************
      *  All done.                                                  *
      ***************************************************************

       A900-MAIN-EXIT.
           CLOSE PRINTFILE.
           STOP RUN.

```

Figure 4. Sample COBOL Program Using SQL Statements (Part 4 of 7)

```
5722ST1 V5R2M0 020719            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
       ***************************************************************
       *  Fetch and write the rows to PRINTFILE.                     *
       ***************************************************************

        B000-GENERATE-REPORT1.
       8 EXEC SQL
                WHENEVER NOT FOUND GO TO A100-DONE1
           END-EXEC.
       9 EXEC SQL
                FETCH C1 INTO :PROJECT.PROJNO, :RPT1.EMPNO,
                              :RPT1.NAME, :RPT1.SALARY
           END-EXEC.
           MOVE CORRESPONDING RPT1 TO RPT1-DATA.
           MOVE PROJNO OF RPT1 TO PROJNO OF RPT1-DATA.
           WRITE PRINT-RECORD FROM RPT1-DATA
                 BEFORE ADVANCING 1 LINE.

        B010-GENERATE-REPORT1-EXIT.
           EXIT.

       ***************************************************************
       *  Fetch and write the rows to PRINTFILE.                     *
       ***************************************************************

        C000-GENERATE-REPORT2.
           EXEC SQL
                WHENEVER NOT FOUND GO TO A200-DONE2
           END-EXEC.
      12 EXEC SQL
                FETCH C2 INTO :RPT2
           END-EXEC.
           MOVE CORRESPONDING RPT2 TO RPT2-DATA.
           WRITE PRINT-RECORD FROM RPT2-DATA
                 BEFORE ADVANCING 1 LINE.

        C010-GENERATE-REPORT2-EXIT.
           EXIT.

       ***************************************************************
       *  Error occured while updating table.  Inform user and      *
       *  rollback changes.                                          *
       ***************************************************************

        E010-UPDATE-ERROR.
      13 EXEC SQL
                WHENEVER SQLERROR CONTINUE
           END-EXEC.
           MOVE SQLCODE TO CODE-EDIT.
           STRING "*** ERROR Occurred while updating table.  SQLCODE="
                 CODE-EDIT DELIMITED BY SIZE INTO PRINT-RECORD.
           WRITE PRINT-RECORD.
      14 EXEC SQL
                ROLLBACK
           END-EXEC.
           STOP RUN.

       ***************************************************************
       *  Error occured while generating reports.  Inform user and   *
       *  exit.                                                      *
       ***************************************************************

        E020-REPORT-ERROR.
           MOVE SQLCODE TO CODE-EDIT.
           STRING "*** ERROR Occurred while generating reports.  SQLCODE
       -           "=" CODE-EDIT DELIMITED BY SIZE INTO PRINT-RECORD.
           WRITE PRINT-RECORD.
           STOP RUN.
                       * * * * *  E N D  O F  S O U R C E  * * * * *
```

Figure 4. Sample COBOL Program Using SQL Statements (Part 5 of 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
```

Figure 4. Sample COBOL Program Using SQL Statements (Part 6 of 7)

```
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  * * * * *
```

Figure 4. Sample COBOL Program Using SQL Statements (Part 7 of 7)

Example: SQL Statements in PL/I

Note: See “Code disclaimer information” on page viii information for information pertaining to code examples.

```
5722ST1 V5R2M0 020719           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............V5R2M0
INCLUDE file..............*LIBL/*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
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
Source member changed on 07/01/96  12:53:08

/* 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 showing 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.                                   */
/*********************************************************************/
```

Figure 5. Sample PL/I Program Using SQL Statements (Part 1 of 6)

```
5722ST1 V5R2M0 020719            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

 PLIEX: PROC;

   DCL RAISE_DATE CHAR(10);
   DCL WORK_DAYS  FIXED BIN(15);
   DCL COMMISSION FIXED DECIMAL(8,2);
   DCL PERCENTAGE FIXED DECIMAL(5,2);

   /* File declaration for sysprint */
   DCL SYSPRINT FILE EXTERNAL OUTPUT STREAM PRINT;

   /* Structure for report 1 */
   DCL 1 RPT1,
1 %INCLUDE PROJECT (PROJECT, RECORD,,COMMA);
         15 EMPNO     CHAR(6),
         15 NAME      CHAR(30),
         15 SALARY    FIXED DECIMAL(8,2);

   /* Structure for report 2 */
   DCL 1 RPT2,
         15 PROJNO          CHAR(6),
         15 PROJECT_NAME    CHAR(36),
         15 EMPLOYEE_COUNT  FIXED BIN(15),
         15 TOTL_PROJ_COST  FIXED DECIMAL(10,2);

  2  EXEC SQL INCLUDE SQLCA;

   COMMISSION = 2000.00;
   PERCENTAGE = 1.04;
   RAISE_DATE = '1982-06-01';
   WORK_DAYS  = 253;
   OPEN FILE(SYSPRINT);

   /* Update the selected employee's salaries by the new percentage. */
   /* If an error occurs during the update, ROLLBACK the changes.    */
  3 EXEC SQL WHENEVER SQLERROR GO TO UPDATE_ERROR;
  4 EXEC SQL
        UPDATE CORPDATA/EMPLOYEE
           SET SALARY = SALARY * :PERCENTAGE
           WHERE COMM >= :COMMISSION ;

   /* Commit changes */
  5 EXEC SQL
        COMMIT;
   EXEC SQL WHENEVER SQLERROR GO TO REPORT_ERROR;

   /* Report the updated statistics for each project supported by one */
   /* of the selected employees.                                       */

   /* Write out the header for Report 1 */
   put file(sysprint)
       edit('REPORT OF PROJECTS AFFECTED BY EMPLOYEE RAISES')
           (col(22),a);
   put file(sysprint)
       edit('PROJECT','EMPID','EMPLOYEE NAME','SALARY')
           (skip(2),col(1),a,col(10),a,col(20),a,col(55),a);

  6 exec sql
        declare c1 cursor for
          select DISTINCT projno, EMPPROJACT.empno,
                         lastname||', '||firstnme, salary
          from CORPDATA/EMPPROJACT, CORPDATA/EMPLOYEE
          where EMPPROJACT.empno = EMPLOYEE.empno and
                comm >= :COMMISSION
          order by projno, empno;
  7 EXEC SQL
        OPEN C1;

```

Figure 5. Sample PL/I Program Using SQL Statements (Part 2 of 6)

```
5722ST1 V5R2M0 020719            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
   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
  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 occured 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
```

Figure 5. Sample PL/I Program Using SQL Statements (Part 3 of 6)

```
5722ST1 V5R2M0 020719            Create SQL PL/I Program          PLIEX                              08/06/02 12:53:36   Page     4
 14  EXEC SQL
         ROLLBACK;
    GO TO FINISHED;

    /* Error occured while generating reports.  Inform user and exit.  */
  REPORT_ERROR:
    PUT FILE(SYSPRINT) EDIT('*** ERROR Occurred while generating '||
     'reports.  SQLCODE=',SQLCODE)(A,F(5));
     GO TO FINISHED;

    /* All done  */
  FINISHED:
    CLOSE FILE(SYSPRINT);
    RETURN;

  END PLIEX;
                      * * * * *  E N D  O F  S O U R C E  * * * * *
```

Figure 5. Sample PL/I Program Using SQL Statements (Part 4 of 6)

```
5722ST1 V5R2M0 020719            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
```

Figure 5. Sample PL/I Program Using SQL Statements (Part 5 of 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  * * * * *
```

Figure 5. Sample PL/I Program Using SQL Statements (Part 6 of 6)

Example: SQL Statements in RPG for iSeries Programs

Note: See "Code disclaimer information" on page viii information for information pertaining to code examples.

```
5722ST1 V5R2M0 020719            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............V5R2M0
INCLUDE file..............*LIBL/*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
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
Source member changed on 07/01/96  17:06:17

H
F*  File declaration for QPRINT
F*
FQPRINT  O   F     132           PRINTER
I*
I* Structure for report 1.
I*
1 IRPT1      E DSPROJECT
I              PROJNAME                        PROJNM
I              RESPEMP                         RESEM
I              PRSTAFF                         STAFF
I              PRSTDATE                        PRSTD
I              PRENDATE                        PREND
I              MAJPROJ                         MAJPRJ
I*
I            DS
I                                         1   6 EMPNO
I                                         7  36 NAME
I                                      P  37  412SALARY
I*
I* Structure for report 2.
I*
IRPT2        DS
I                                         1   6 PRJNUM
I                                         7  42 PNAME
I                                      B  43  440EMPCNT
I                                      P  45  492PRCOST
I*
I            DS
I                                      B   1   20WRKDAY
I                                      P   3   62COMMI
I                                         7  16 RDATE
I                                      P  17  202PERCNT
```

Figure 6. Sample RPG for iSeries Program Using SQL Statements (Part 1 of 6)

```
5722ST1 V5R2M0 020719            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
   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
   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
```

Figure 6. Sample RPG for iSeries Program Using SQL Statements (Part 2 of 6)

```
Record  *...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8  SEQNBR  Last change
    11  C/EXEC SQL                                                                  10400
       C+  DECLARE C2 CURSOR FOR                                                    10500
       C+    SELECT EMPPROJACT.PROJNO, PROJNAME, COUNT(*),                          10600
       C+       SUM((DAYS(EMENDATE) - DAYS(EMSTDATE)) * EMPTIME *                   10700
       C+           DECIMAL((SALARY/:WRKDAY),8,2))                                  10800
       C+    FROM CORPDATA/EMPPROJACT, CORPDATA/PROJECT, CORPDATA/EMPLOYEE          10900
       C+    WHERE EMPPROJACT.PROJNO = PROJECT.PROJNO AND                           11000
       C+          EMPPROJACT.EMPNO = EMPLOYEE.EMPNO AND                            11100
       C+          PRENDATE > :RDATE                                                11200
       C+    GROUP BY EMPPROJACT.PROJNO, PROJNAME                                   11300
       C+    ORDER BY 1                                                             11400
       C/END-EXEC                                                                   11500
       C*                                                                           11600
       C/EXEC SQL OPEN C2                                                           11700
       C/END-EXEC                                                                   11800
       C*                                                                           11900
       C* Fetch and write the rows to QPRINT.                                       12000
       C*                                                                           12100
       C/EXEC SQL WHENEVER NOT FOUND GO TO DONE2                                    12200
       C/END-EXEC                                                                   12300
       C           SQLCOD    DOUNE0                                                 12400
       C/EXEC SQL                                                                   12500
    12  C+   FETCH C2 INTO :RPT2                                                    12600
       C/END-EXEC                                                                   12700
       C                     EXCPTRECD                                              12800
       C                     END                                                    12900
       C           DONE2     TAG                                                    13000
       C/EXEC SQL CLOSE C2                                                          13100
       C/END-EXEC                                                                   13200
       C                     RETRN                                                  13300
       C*                                                                           13400
       C* Error occured while updating table.  Inform user and rollback             13500
       C* changes.                                                                  13600
       C*                                                                           13700
       C           UPDERR    TAG                                                    13800
       C                     EXCPTRECE                                              13900
    13  C/EXEC SQL WHENEVER SQLERROR CONTINUE                                       14000
       C/END-EXEC                                                                   14100
       C*                                                                           14200
    14  C/EXEC SQL                                                                  14300
       C+   ROLLBACK                                                                14400
       C/END-EXEC                                                                   14500
       C                     RETRN                                                  14600
       C*                                                                           14700
       C* Error occured while generating reports.  Inform user and exit.            14800
       C*                                                                           14900
       C           RPTERR    TAG                                                    15000
       C                     EXCPTRECF                                              15100
       C*                                                                           15200
       C* All done.                                                                 15300
       C*                                                                           15400
       C           FINISH    TAG                                                    15500
       OQPRINT  E 0201           RECA                                               15700
       O                                   45 'REPORT OF PROJECTS AFFEC'            15800
       O                                   64 'TED BY EMPLOYEE RAISES'              15900
       O        E 01             RECA                                               16000
       O                                    7 'PROJECT'                             16100
       O                                   17 'EMPLOYEE'                            16200
       O                                   32 'EMPLOYEE NAME'                       16300
       O                                   60 'SALARY'                              16400
       O        E 01             RECB                                               16500
       O                         PROJNO     6                                       16600
       O                         EMPNO     15                                       16700
       O                         NAME      50                                       16800
       O                         SALARYL   61                                       16900
       O        E 22             RECC                                               17000
       O                                   42 'ACCUMULATED STATISTIC'               17100
       O                                   54 'S BY PROJECT'                        17200
       O        E 01             RECC                                               17300
       O                                    7 'PROJECT'                             17400
       O                                   56 'NUMBER OF'                           17500
       O                                   67 'TOTAL'                               17600
       O        E 02             RECC                                               17700
       O                                    6 'NUMBER'                              17800
       O                                   21 'PROJECT NAME'                        17900
       O                                   56 'EMPLOYEES'                           18000
       O                                   66 'COST'                                18100
```

Figure 6. Sample RPG for iSeries Program Using SQL Statements (Part 3 of 6)

```
5722ST1 V5R2M0 020719            Create SQL RPG Program           RPGEX                          08/06/02 12:55:22   Page     4
  182        O        E 01             RECD                                          18200
  195        O                                   57 'CODE='                          19500
  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
  196        O                         SQLCODL   67                                  19600
                                * * * * *  E N D  O F  S O U R C E  * * * * *
```

Figure 6. Sample RPG for iSeries Program Using SQL Statements (Part 4 of 6)

```
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
PRENDATE                         105      DATE(10) COLUMN IN CORPDATA.PROJECT
PRJNUM                           24       CHARACTER(6) IN RPT2
```

Figure 6. Sample RPG for iSeries Program Using SQL Statements (Part 5 of 6)

```
5722ST1 V5R2M0 020719            Create SQL RPG Program          RPGEX                                    08/06/02 12:55:22   Page     6
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  * * * * *
```

Figure 6. Sample RPG for iSeries Program Using SQL Statements (Part 6 of 6)

Example: SQL Statements in ILE RPG for iSeries Programs

Note: See “Code disclaimer information” on page viii information for information pertaining to code examples.

```
5722ST1 V5R2M0 020719           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
Listing option............*PRINT
Target release............V5R2M0
INCLUDE file..............*LIBL/*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
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
Source member changed on 07/01/96  15:55:32

     H                                                                             100
     F*  File declaration for QPRINT                                               200
     F*                                                                            300
     FQPRINT    O    F  132        PRINTER                                         400
     D*                                                                            500
     D* Structure for report 1.                                                    600
     D*                                                                            700
    1 DRPT1           E DS                  EXTNAME(PROJECT)                       800
     D*                                                                            900
     D                DS                                                          1000
     D EMPNO                  1      6                                            1100
     D NAME                   7     36                                            1200
     D SALARY                37     41P 2                                         1300
     D*                                                                           1400
     D* Structure for report 2.                                                   1500
     D*                                                                           1600
     DRPT2            DS                                                          1700
     D PRJNUM                 1      6                                            1800
     D PNAME                  7     42                                            1900
     D EMPCNT                43     44B 0                                         2000
     D PRCOST                45     49P 2                                         2100
     D*                                                                           2200
     D                DS                                                          2300
     D WRKDAY                 1      2B 0                                         2400
     D COMMI                  3      6P 2                                         2500
     D RDATE                  7     16                                            2600
     D PERCNT                17     20P 2                                         2700
      *                                                                           2800
```

Figure 7. Sample ILE RPG for iSeries Program Using SQL Statements (Part 1 of 6)

```
5722ST1 V5R2M0 020719            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
      2 C                   Z-ADD     253           WRKDAY                                   2900
        C                   Z-ADD     2000.00       COMMI                                    3000
        C                   Z-ADD     1.04          PERCNT                                   3100
        C                   MOVEL     '1982-06-'    RDATE                                    3200
        C                   MOVE      '01'          RDATE                                    3300
        C                   SETON                                       LR                   3400
        C*                                                                                   3500
        C* Update the selected projects by the new percentage. If an                         3600
        C* error occurs during the update, ROLLBACK the changes.                             3700
        C*                                                                                   3800
      3 C/EXEC SQL WHENEVER SQLERROR GOTO UPDERR                                             3900
        C/END-EXEC                                                                           4000
        C*                                                                                   4100
        C/EXEC SQL                                                                           4200
      4 C+ UPDATE CORPDATA/EMPLOYEE                                                          4300
        C+    SET SALARY = SALARY * :PERCNT                                                  4400
        C+    WHERE COMM >= :COMMI                                                           4500
        C/END-EXEC                                                                           4600
        C*                                                                                   4700
        C* Commit changes.                                                                   4800
        C*                                                                                   4900
      5 C/EXEC SQL COMMIT                                                                    5000
        C/END-EXEC                                                                           5100
        C*                                                                                   5200
        C/EXEC SQL WHENEVER SQLERROR GO TO RPTERR                                            5300
        C/END-EXEC                                                                           5400
        C*                                                                                   5500
        C* Report the updated statistics for each employee assigned to                       5600
        C* selected projects.                                                                5700
        C*                                                                                   5800
        C* Write out the header for report 1.                                                5900
        C*                                                                                   6000
        C                   EXCEPT    RECA                                                   6100
      6 C/EXEC SQL DECLARE C1 CURSOR FOR                                                     6200
        C+    SELECT DISTINCT PROJNO, EMPPROJACT.EMPNO,                                      6300
        C+           LASTNAME||', '||FIRSTNME, SALARY                                        6400
        C+        FROM CORPDATA/EMPPROJACT, CORPDATA/EMPLOYEE                                6500
        C+        WHERE EMPPROJACT.EMPNO = EMPLOYEE.EMPNO AND                                6600
        C+              COMM >= :COMMI                                                       6700
        C+        ORDER BY PROJNO, EMPNO                                                     6800
        C/END-EXEC                                                                           6900
        C*                                                                                   7000
      7 C/EXEC SQL                                                                           7100
        C+  OPEN C1                                                                          7200
        C/END-EXEC                                                                           7300
        C*                                                                                   7400
        C* Fetch and write the rows to QPRINT.                                               7500
        C*                                                                                   7600
      8 C/EXEC SQL WHENEVER NOT FOUND GO TO DONE1                                            7700
        C/END-EXEC                                                                           7800
        C     SQLCOD        DOUNE     0                                                      7900
        C/EXEC SQL                                                                           8000
      9 C+   FETCH C1 INTO :PROJNO, :EMPNO, :NAME, :SALARY                                   8100
        C/END-EXEC                                                                           8200
        C                   EXCEPT    RECB                                                   8300
        C                   END                                                              8400
        C     DONE1         TAG                                                              8500
        C/EXEC SQL                                                                           8600
     10 C+  CLOSE C1                                                                         8700
        C/END-EXEC                                                                           8800
        C*                                                                                   8900
        C* For all project ending at a date later than the raise date                        9000
        C* (i.e. those projects potentially affected by the salary raises)                   9100
        C* generate a report containing the project number, project name,                    9200
        C* the count of employees participating in the project and the                       9300
        C* total salary cost of the project.                                                 9400
        C*                                                                                   9500
        C* Write out the header for report 2.                                                9600
        C*                                                                                   9700
        C                   EXCEPT    RECC                                                   9800
        C/EXEC SQL                                                                           9900
                                                          12000
```

Figure 7. Sample ILE RPG for iSeries Program Using SQL Statements (Part 2 of 6)

```
5722ST1 V5R2M0 020719           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
  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
  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 occured 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 occured 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
```

Figure 7. Sample ILE RPG for iSeries Program Using SQL Statements (Part 3 of 6)

```
  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  * * * * *
```

Figure 7. Sample ILE RPG for iSeries Program Using SQL Statements (Part 4 of 6)

```
5722ST1 V5R2M0 020719           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
PRENDATE                          ****     COLUMN
                                           99
PRENDATE                          99       DATE(10) COLUMN IN CORPDATA.PROJECT
PRJNUM                            18       CHARACTER(6) DBCS-open IN RPT2
```

Figure 7. Sample ILE RPG for iSeries Program Using SQL Statements (Part 5 of 6)

```
5229ST1 V5R2M0 020719           Create SQL ILE RPG Object         RPGLEEX                          08/06/02 16:03:02   Page     6
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  * * * * *
```

Figure 7. Sample ILE RPG for iSeries Program Using SQL Statements (Part 6 of 6)

Example: SQL Statements in REXX Programs

Note: See "Code disclaimer information" on page viii information for information pertaining to code examples.

```
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.    */
3 SIGNAL ON ERROR
ERRLOC = 'UPDATE_ERROR'
UPDATE_STMT = 'UPDATE CORPDATA/EMPLOYEE ',
              'SET SALARY = SALARY * ?  ',
              'WHERE COMM >= ?          '
EXECSQL,
       'PREPARE S1 FROM :UPDATE_STMT'
4 EXECSQL,
       'EXECUTE S1 USING :PERCENTAGE,',
       '                 :COMMISSION '
/* Commit changes */
5 EXECSQL,
       'COMMIT'
ERRLOC = 'REPORT_ERROR'

/* 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'
6 EXECSQL,
       'DECLARE C1 CURSOR FOR S2'
```

Figure 8. Sample REXX Procedure Using SQL Statements (Part 1 of 3)

```
Record  *...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
 7 EXECSQL,
       'OPEN C1 USING :COMMISSION'

/* Handle the FETCH errors and warnings inline */
SIGNAL OFF ERROR

/* Fetch all of the rows */
DO UNTIL (SQLCODE <> 0)
   9 EXECSQL,
         '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.                                                */
 8 IF SQLCODE = 100 THEN
    LEAVE

  /* Print out the fetched row */
  SAY RPT1.PROJNO '   ' RPT1.EMPNO '   ' RPT1.NAME '     ' RPT1.SALARY
END;

 10 EXECSQL,
        'CLOSE C1'

..+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
/* For all projects ending at a date later than 'raise_date'       */
/* (i.e. 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.                                */

  /* 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'
 11 EXECSQL,
       '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)
```

Figure 8. Sample REXX Procedure Using SQL Statements (Part 2 of 3)

```
*...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
  12 EXECSQL,
         '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:

  13 SIGNAL 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
        14 EXECSQL,
               '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  * * * * *
```

Report produced by sample programs that use SQL

The following report is produced by each of the preceding sample 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
```

Appendix B. DB2 UDB for iSeries CL Command Descriptions for Host Language Precompilers

This appendix contains the syntax diagrams referred to and used in this guide and the SQL Reference book.

For more details, see "SQL precompiler commands".

SQL precompiler commands

DB2 UDB for iSeries provides commands for precompiling programs coded in the following programming languages:

- COBOL
- ILE COBOL
- ILE C
- C++
- PL/I
- RPG
- ILE RPG

CRTSQLCBL (Create Structured Query Language COBOL) Command

Job: B,I   Pgm: B,I   REXX: B,I   Exec

```
                               ┌─*CURLIB/──────┐
►►──CRTSQLCBL──PGM(───────────┤               ├──program-name──)──────────────────────►
                               └─library-name/─┘

►─┬──────────────────────────────────────────────────────────────────────────────┬────►
  └─SRCFILE(──┬─*LIBL/────────┬──┬─QLBLSRC──────────┬──)─┘
              ├─*CURLIB/──────┤  └─source-file-name─┘
              └─library-name/─┘

                                                         (1)
►─┬─────────────────────────────────────────────────────┬─────────────────────────────►
  └─SRCMBR(──┬─*PGM─────────────────────┬──)─┘
             └─source-file-member-name──┘

►─┬─────────────────────────────────┬──┬─────────────────────────────────┬────────────►
  └─OPTION(──┤ OPTION Details ├──)─┘  └─TGTRLS(──┬─*CURRENT─┬──)─┘
                                                 ├─*PRV─────┤
                                                 └─VxRxMx───┘
```

CRTSQLCBL

```
►─┬──────────────────────────────────────────────────────────────────┬─►
  │            ┌─*LIBL/────────┐  ┌─*SRCFILE─────────┐                │
  └─INCFILE(───┼───────────────┼──┴─source-file-name─┴──)─────────────┘
               ├─*CURLIB/──────┤
               └─library-name/─┘

►─┬──────────────────────────────┬─┬─────────────────────────────────┬─►
  │           ┌─*UR───┐          │ │             ┌─*ENDPGM─┐         │
  │           ├─*CHG──┤          │ └─CLOSQLCSR(──┼─*ENDSQL─┼──)──────┘
  └─COMMIT(───┼─*ALL──┼──)───────┘               └─*ENDJOB─┘
              ├─*RS───┤
              ├─*CS───┤
              ├─*NONE─┤
              ├─*NC───┤
              └─*RR───┘

►─┬─────────────────────────────────┬─┬──────────────────────────────┬─►
  │             ┌─*OPTIMIZE─┐       │ │           ┌─*ALLREAD─┐       │
  └─ALWCPYDTA(──┼─*YES──────┼──)────┘ └─ALWBLK(───┼─*NONE────┼──)────┘
                └─*NO───────┘                     └─*READ────┘

►─┬──────────────────────────┬─┬──────────────────────────────────┬─►
  │           ┌─*NO──┐       │ │           ┌─10─────────────┐     │
  └─DLYPRP(───┴─*YES─┴──)────┘ └─GENLVL(───┴─severity-level─┴──)──┘

►─┬──────────────────────────┬─┬───────────────────────────┬─►
  │           ┌─*JOB─┐       │ │           ┌─*JOB───┐      │
  └─DATFMT(───┼─*USA─┼──)────┘ └─DATSEP(───┼─'/'────┼──)───┘
              ├─*ISO─┤                     ├─'.'────┤
              ├─*EUR─┤                     ├─','────┤
              ├─*JIS─┤                     ├─'-'────┤
              ├─*MDY─┤                     ├─' '────┤
              ├─*DMY─┤                     └─*BLANK─┘
              ├─*YMD─┤
              └─*JUL─┘

►─┬──────────────────────────┬─┬───────────────────────────┬─►
  │           ┌─*HMS─┐       │ │           ┌─*JOB───┐      │
  └─TIMFMT(───┼─*USA─┼──)────┘ └─TIMSEP(───┼─':'────┼──)───┘
              ├─*ISO─┤                     ├─'.'────┤
              ├─*EUR─┤                     ├─','────┤
              └─*JIS─┘                     ├─' '────┤
                                           └─*BLANK─┘

►─┬───────────────────────────┬─────────────────────────────►
  │            ┌─*YES─┐       │
  └─REPLACE(───┴─*NO──┴──)────┘

►─┬──────────────────────────────────────────────┬─┬────────────────────────────────┬─►
  │        ┌─*LOCAL─────────────────────┐        │ │         ┌─*CURRENT──┐          │
  └─RDB(───┼─relational-database-name───┼──)─────┘ └─USER(───┴─user-name─┴──)──────┘
           └─*NONE──────────────────────┘
```

```
►─┬──────────────────────────────┬─┬────────────────────────────┬──────────────►
  │            ┌─*NONE─────┐     │ │             ┌─*DUW─┐       │
  └─PASSWORD(──┴─password──┴──)─┘ └─RDBCNNMTH(──┴─*RUW─┴──)────┘

►─┬───────────────────────────────────────┬─┬──────────────────────────┬───────►
  │             ┌─*NONE───────────┐       │ │            ┌─*NO──┐      │
  └─DFTRDBCOL(──┴─collection-name─┴──)───┘ └─DYNDFTCOL(──┴─*YES─┴──)──┘

►─┬──────────────────────────────────────────────────┬─────────────────────────►
  │         ┌─*PGMLIB/────────┐ ┌─*PGM─────────┐     │
  └─SQLPKG(─┼─────────────────┼─┴─package-name─┴──)──┘
            └─library-name/───┘

►─┬──────────────────────────────────────┬─┬────────────────────────────┬──────►
  │           ┌─*NAMING─────────────┐    │ │              ┌─*DB2─┐      │
  └─SQLPATH(──┼─*LIBL───────────────┼──)─┘ └─SQLCURRULE(──┴─*STD─┴──)───┘
              │  ┌───────────────┐  │
              │  ▼               │  │
              └───collection-name┴──┘

►─┬───────────────────────────┬─┬──────────────────────────┬───────────────────►
  │           ┌─*NOFLAG─┐     │ │            ┌─*NONE─┐     │
  └─SAAFLAG(──┴─*FLAG───┴──)─┘ └─FLAGSTD(───┴─*ANS──┴──)──┘

►─┬───────────────────────────────────────────────────────────┬────────────────►
  │           ┌─*LIBL/────────┐ ┌─QSYSPRT────────────┐        │
  └─PRTFILE(──┼───────────────┼─┴─printer-file-name──┴──)─────┘
              ├─*CURLIB/──────┤
              └─library-name/─┘

►─┬─────────────────────────────────────────────┬──────────────────────────────►
  │          ┌─*JOB──────────────────────────┐  │
  └─SRTSEQ(──┼─*JOBRUN───────────────────────┼─)┘
             ├─*LANGIDUNQ────────────────────┤
             ├─*LANGIDSHR────────────────────┤
             ├─*HEX──────────────────────────┤
             │  ┌─*LIBL/────────┐            │
             └──┼───────────────┼─table-name─┘
                ├─*CURLIB/──────┤
                └─library-name/─┘

►─┬─────────────────────────────────┬─┬─────────────────────────────┬──────────►
  │          ┌─*JOB──────────┐      │ │           ┌─*NAMING─┐       │
  └─LANGID(──┼─*JOBRUN───────┼──)───┘ └─USRPRF(──┼─*OWNER──┼──)────┘
             └─language-ID───┘                    └─*USER───┘

►─┬─────────────────────────────┬──────────────────────────────────────────────►
  │             ┌─*USER──┐      │
  └─DYNUSRPRF(──┴─*OWNER─┴──)──┘

►─┬─────────────────────────────────────────────────────────────┬──────────────►
  │             ┌─QTEMP/────────┐ ┌─QSQLTEMP──────────┐         │
  └─TOSRCFILE(──┼───────────────┼─┴─source-file-name──┴──)──────┘
                ├─*LIBL/────────┤
                ├─*CURLIB/──────┤
                └─library-name/─┘
```

```
►─┬──────────────────────────────────┬────────────────────────────────────►◄
  │         ┌─*SRCMBRTXT──┐          │
  └─TEXT(───┼─*BLANK──────┼───)──────┘
            └─'description'┘
```

OPTION Details:

```
   ┌─*NOSRC─────┐
   ├─*NOSOURCE──┤ ┌─*NOXREF─┐ ┌─*GEN───┐ ┌─*JOB─────┐ ┌─*QUOTESQL─┐
├──┼────────────┼─┼─────────┼─┼────────┼─┼──────────┼─┼───────────┼──────►
   ├─*SOURCE────┤ └─*XREF───┘ └─*NOGEN─┘ ├─*PERIOD──┤ └─*APOSTSQL─┘
   └─*SRC───────┘                        ├─*SYSVAL──┤
                                         └─*COMMA───┘

   ┌─*QUOTE─┐ ┌─*SYS─┐ ┌─*NOSECLVL─┐ ┌─*NOLSTDBG─┐
►──┼────────┼─┼──────┼─┼───────────┼─┼───────────┼───────────────────────┤
   └─*APOST─┘ └─*SQL─┘ └─*SECLVL───┘ └─*LSTDBG───┘
```

Notes:

1 All parameters preceding this point can be specified in positional form.

Purpose:

The Create Structured Query Language COBOL (CRTSQLCBL) 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 COBOL compiler to compile the program.

Parameters:

PGM

Specifies the qualified name of the compiled program.

The name of the compiled COBOL program can be qualified by one of the following library values:

***CURLIB** The compiled COBOL program is created in the current library for the job. If no library is specified as the current library for the job, the QGPL library is used.

library name: Specify the name of the library where the compiled COBOL program is created.

program-name: Specify the name of the compiled COBOL program.

SRCFILE

Specifies the qualified name of the source file that contains the COBOL source with SQL statements.

The name of the source file can be qualified by one of the following library values:

***LIBL:** All libraries in the job's library list are searched until the first match is found.

***CURLIB:** The current library for the job is searched. If no library is specified as the current library for the job, the QGPL library is used.

library-name: Specify the name of the library to be searched.

QLBLSRC: If a COBOL source file name is not specified, the IBM-supplied source file QLBLSRC contains the COBOL source.

source-file-name: Specify the name of the source file that contains the COBOL source. This source file should have a record length of 92 bytes. The source file can be a database file, device file, or an inline data file.

SRCMBR

Specifies the name of the source file member that contains the COBOL source. This parameter is specified only if the source file name in the SRCFILE parameter is a database file. If this parameter is not specified, the PGM name specified on the PGM parameter is used.

***PGM:** Specifies that the COBOL source is in the member of the source file that has the same name as that specified on the PGM parameter.

source-file-member-name: Specify the name of the member that contains the COBOL source.

OPTION

Specifies whether one or more of the following options are used when the COBOL source is precompiled. If an option is specified more than once, or if two options conflict, the last option specified is used.

Element 1: Source Listing Options

***NOSOURCE** or ***NOSRC:** A source printout is not produced by the precompiler unless errors are detected during precompile or create package.

***SOURCE or *SRC:** The precompiler produces a source printout consisting of COBOL source input.

Element 2: Cross-Reference Options

***NOXREF:** The precompiler does not cross-reference names.

***XREF:** The precompiler cross-references items in the program to the statement numbers in the program that refer to those items.

Element 3: Program Creation Options

***GEN:** The compiler creates a program that can run after the program is compiled. An SQL package object is created if a relational database name is specified on the RDB parameter.

***NOGEN:** The precompiler does not call the COBOL compiler, and a program and SQL package are not created.

Element 4: Decimal Point Options

***JOB:** The value used as the decimal point for numeric constants in SQL is the representation of decimal point specified for the job at precompile time.

***SYSVAL:** The value used as the decimal point for numeric constants in SQL statements is the QDECFMT system value.

Note: If QDECFMT specifies that the value used as the decimal point is a comma, any numeric constants in lists (such as in the SELECT clause or the VALUES clause) must be separated by a comma followed by a blank. For example, VALUES(1,1, 2,23, 4,1) is equivalent to VALUES(1.1,2.23,4.1) in which the decimal point is a period.

***PERIOD:** The value used as the decimal point for numeric constants in SQL statements is a period.

***COMMA:** The value used as the decimal point for numeric constants in SQL statements is a comma.

Note: Any numeric constants in lists (such as in the SELECT clause or the VALUES clause) must be separated by a comma followed by a blank. For example, VALUES(1,1, 2,23, 4,1) is equivalent to VALUES(1.1,2.23,4.1) where the decimal point is a period.

Element 5: String Delimiter Options

***QUOTESQL:** A double quote (") is the string delimiter in the SQL statements.

***APOSTSQL:** An apostrophe (') is the string delimiter in the SQL statements.

Element 6: Literal Options

***QUOTE:** A double quote (") is used for non-numeric literals and Boolean literals in the COBOL statements.

***APOST:** An apostrophe (') is used for non-numeric literals and Boolean literals in the COBOL statements.

Element 7: Naming Convention Option

***SYS:** The system naming convention (library-name/file-name) is used.

***SQL:** The SQL naming convention (schema-name.table-name) is used. When creating a program on a remote database other than an iSeries system, *SQL must be specified as the naming convention.

Element 8: Second-Level Message Text Option

***NOSECLVL:** Second-level text descriptions are not added to the listing.

***SECLVL:** Second-level text with replacement data is added for all messages on the listing.

Element 9: Debug Listing View

***NOLSTDBG:** Error and debug information is not generated.

***LSTDBG:** The SQL precompiler generates a listing view, and error and debug information required for this view. You can use *LSTDBG only if you are using the CODE/400 product to compile your program.

TGTRLS

Specifies the release of the operating system on which the user intends to use the object being created.

In the examples given for the *CURRENT and *PRV values, and when specifying the release-level value, the format VxRxMx is used to specify the release, where Vx is the version, Rx is the release, and Mx is the modification level. For example, V2R3M0 is version 2, release 3, modification level 0.

***CURRENT:** The object is to be used on the release of the operating system currently running on the user's system. For example, if V2R3M5 is running on the system, *CURRENT means the user intends to use the object on a system with V2R3M5 installed. The user can also use the object on a system with any subsequent release of the operating system installed.

Note: If V2R3M5 is running on the system, and the object is to be used on a system with V2R3M0 installed, specify TGTRLS(V2R3M0) not TGTRLS(*CURRENT).

***PRV:** The object is to be used on the previous release with modification level 0 of the operating system. For example, if V2R3M5 is running on the user's system, *PRV means the user intends to use the object on a system with V2R2M0 installed. The user can also use the object on a system with any subsequent release of the operating system installed.

release-level: Specify the release in the format VxRxMx. The object can be used on a system with the specified release or with any subsequent release of the operating system installed.

Valid values depend on the current version, release, and modification level, and they change with each new release. If you specify a release-level which is earlier than the earliest release level supported by this command, an error message is sent indicating the earliest supported release.

INCFILE

Specifies the qualified name of the source file that contains members included in the program with any SQL INCLUDE statement.

The name of the source file can be qualified by one of the following library values:

***LIBL:** All libraries in the job's library list are searched until the first match is found.

***CURLIB:** The current library for the job is searched. If no library is specified as the current library for the job, the QGPL library is used.

library-name: Specify the name of the library to be searched.

***SRCFILE:** The qualified source file specified in the SRCFILE parameter contains the source file member(s) specified on any SQL INCLUDE statement.

source-file-name: Specify the name of the source file that contains the source file member(s) specified on any SQL INCLUDE statement. The record length of the source file specified here must be no less than the record length of the source file specified for the SRCFILE parameter.

COMMIT

Specifies whether SQL statements in the compiled program are run under commitment control. Files referred to in the host language source are not affected by this option. Only SQL tables, SQL views, and SQL packages referred to in SQL statements are affected.

Note: Files referenced in the COBOL source are not affected by this option.

***CHG or *UR:** Specifies the objects referred to in SQL ALTER, CALL, COMMENT ON, CREATE, DROP, GRANT, LABEL ON, RENAME, and REVOKE statements and the rows updated, deleted, and inserted are locked until the end of the unit of work (transaction). Uncommitted changes in other jobs can be seen.

***ALL or *RS:** Specifies the objects referred to in SQL ALTER, CALL, COMMENT ON, CREATE, DROP, GRANT, LABEL ON, RENAME, and

REVOKE statements and the rows selected, updated, deleted, and inserted are locked until the end of the unit of work (transaction). Uncommitted changes in other jobs cannot be seen.

***CS:** Specifies the objects referred to in SQL ALTER, CALL, COMMENT ON, CREATE, DROP, GRANT, LABEL ON, RENAME, and REVOKE statements and the rows updated, deleted, and inserted are locked until the end of the unit of work (transaction). A row that is selected, but not updated, is locked until the next row is selected. Uncommitted changes in other jobs cannot be seen.

***NONE or *NC:** Specifies that commitment control is not used. Uncommitted changes in other jobs can be seen. If the SQL DROP SCHEMA statement is included in the program, *NONE or *NC must be used. If a relational database is specified on the RDB parameter and the relational database is on a system that is not on an iSeries, *NONE or *NC cannot be specified.

***RR:** Specifies the objects referred to in SQL ALTER, CALL, COMMENT ON, CREATE, DROP, GRANT, LABEL ON, RENAME, and REVOKE statements and the rows selected, updated, deleted, and inserted are locked until the end of the unit of work (transaction). Uncommitted changes in other jobs cannot be seen. All tables referred to in SELECT, UPDATE, DELETE, and INSERT statements are locked exclusively until the end of the unit of work (transaction).

CLOSQLCSR

Specifies when SQL cursors are implicitly closed, SQL prepared statements are implicitly discarded, and LOCK TABLE locks are released. SQL cursors are explicitly closed when you issue the CLOSE, COMMIT, or ROLLBACK (without HOLD) SQL statements.

***ENDPGM:** SQL cursors are closed and SQL prepared statements are discarded when the program ends. LOCK TABLE locks are released when the first SQL program on the call stack ends.

***ENDSQL:** SQL cursors remain open between calls and can be fetched without running another SQL OPEN. One of the programs higher on the call stack must have run at least one SQL statement. SQL cursors are closed, SQL prepared statements are discarded, and LOCK TABLE locks are released when the first SQL program on the call stack ends. If *ENDSQL is specified for a program that is the first SQL program called (the first SQL program on the call stack), the program is treated as if *ENDPGM was specified.

***ENDJOB:** SQL cursors remain open between calls and can be fetched without running another SQL OPEN. The programs higher on the call stack do not need to have run SQL statements. SQL cursors are left open, SQL prepared statements are preserved, and LOCK TABLE locks are held when the first SQL program on the call stack ends. SQL cursors are closed, SQL prepared statements are discarded, and LOCK TABLE locks are released when the job ends.

ALWCPYDTA

Specifies whether a copy of the data can be used in a SELECT statement.

***OPTIMIZE:** The system determines whether to use the data retrieved directly from the database or to use a copy of the data. The decision is based on which method provides the best performance. If COMMIT is *CHG or *CS and ALWBLK is not *ALLREAD, or if COMMIT is *ALL or *RR, then a copy of the data is used only when it is necessary to run a query.

***YES:** A copy of the data is used only when necessary.

***NO:** A copy of the data is not allowed. If a temporary copy of the data is required to perform the query, an error message is returned.

ALWBLK

Specifies whether the database manager can use record blocking, and the extent to which blocking can be used for read-only cursors.

***ALLREAD:** Rows are blocked for read-only cursors if *NONE or *CHG is specified on the COMMIT parameter. All cursors in a program that are not explicitly able to be updated are opened for read-only processing even though EXECUTE or EXECUTE IMMEDIATE statements may be in the program.

Specifying *ALLREAD:

- Allows record blocking under commitment control level *CHG in addition to the blocking allowed for *READ.
- Can improve the performance of almost all read-only cursors in programs, but limits queries in the following ways:
  - The Rollback (ROLLBACK) command, a ROLLBACK statement in host languages, or the ROLLBACK HOLD SQL statement does not reposition a read-only cursor when *ALLREAD is specified.
  - Dynamic running of a positioned UPDATE or DELETE statement (for example, using EXECUTE IMMEDIATE), cannot be used to update a row in a cursor unless the DECLARE statement for the cursor includes the FOR UPDATE clause.

***NONE:** Rows are not blocked for retrieval of data for cursors.

Specifying *NONE:

- Guarantees that the data retrieved is current.
- May reduce the amount of time required to retrieve the first row of data for a query.
- Stops the database manager from retrieving a block of data rows that is not used by the program when only the first few rows of a query are retrieved before the query is closed.
- Can degrade the overall performance of a query that retrieves a large number of rows.

***READ:** Records are blocked for read-only retrieval of data for cursors when:

- *NONE is specified on the COMMIT parameter, which indicates that commitment control is not used.
- The cursor is declared with a FOR READ ONLY clause or there are no dynamic statements that could run a positioned UPDATE or DELETE statement for the cursor.

Specifying *READ can improve the overall performance of queries that meet the above conditions and retrieve a large number of records.

DLYPRP

Specifies whether the dynamic statement validation for a PREPARE statement is delayed until an OPEN, EXECUTE, or DESCRIBE statement is run. Delaying validation improves performance by eliminating redundant validation.

***NO:** Dynamic statement validation is not delayed. When the dynamic statement is prepared, the access plan is validated. When the dynamic statement is used in an OPEN or EXECUTE statement, the access plan is revalidated. Because the authority or the existence of objects referred to by the dynamic statement may change, you must still check the SQLCODE or SQLSTATE after issuing the OPEN or EXECUTE statement to ensure that the dynamic statement is still valid.

***YES:** Dynamic statement validation is delayed until the dynamic statement is used in an OPEN, EXECUTE, or DESCRIBE SQL statement. When the dynamic statement is used, the validation is completed and an access plan is built. If you specify *YES on this parameter, you should check the SQLCODE and SQLSTATE after running an OPEN, EXECUTE, or DESCRIBE statement to ensure that the dynamic statement is valid.

Note: If you specify *YES, performance is not improved if the INTO clause is used on the PREPARE statement or if a DESCRIBE statement uses the dynamic statement before an OPEN is issued for the statement.

GENLVL

Specifies the severity level at which the create operation fails. If errors occur that have a severity level greater than or equal to this value, the operation ends.

10: The default severity level is 10.

severity-level: Specify a value ranging from 0 through 40.

DATFMT

Specifies the format used when accessing date result columns. All output date fields are returned in the specified format. For input date strings, the specified value is used to determine whether the date is specified in a valid format.

Note: An input date string that uses the format *USA, *ISO, *EUR, or *JIS is always valid.

If a relational database is specified on the RDB parameter and the database is on a system that is not an iSeries system, then *USA, *ISO, *EUR, or *JIS must be specified.

***JOB:** The format specified for the job is used. Use the Display Job (DSPJOB) command to determine the current date format for the job.

***USA:** The United States date format (mm/dd/yyyy) is used.

***ISO:** The International Organization for Standardization (ISO) date format (yyyy-mm-dd) is used.

***EUR:** The European date format (dd.mm.yyyy) is used.

***JIS:** The Japanese Industrial Standard date format (yyyy-mm-dd) is used.

***MDY:** The date format (mm/dd/yy) is used.

***DMY:** The date format (dd/mm/yy) is used.

***YMD:** The date format (yy/mm/dd) is used.

***JUL:** The Julian date format (yy/ddd) is used.

DATSEP
Specifies the separator used when accessing date result columns.

Note: This parameter applies only when *JOB, *MDY, *DMY, *YMD, or *JUL is specified on the DATFMT parameter.

***JOB:** The date separator specified for the job at precompile time is used. Use the Display Job (DSPJOB) command to determine the current value for the job.

'/': A slash (/) is used.

'.': A period (.) is used.

',': A comma (,) is used.

'-': A dash (-) is used.

' ': A blank ( ) is used.

***BLANK:** A blank ( ) is used.

TIMFMT
Specifies the format used when accessing time result columns. For input time strings, the specified value is used to determine whether the time is specified in a valid format.

Note: An input date string that uses the format *USA, *ISO, *EUR, or *JIS is always valid.

If a relational database is specified on the RDB parameter and the database is on a system that is not another iSeries system, the time format must be *USA, *ISO, *EUR, *JIS, or *HMS with a time separator of colon or period.

***HMS:** The (hh:mm:ss) format is used.

***USA:** The United States time format (hh:mm *xx*) is used, where *xx* is AM or PM.

***ISO:** The International Organization for Standardization (ISO) time format (hh.mm.ss) is used.

***EUR:** The European time format (hh.mm.ss) is used.

***JIS:** The Japanese Industrial Standard time format (hh:mm:ss) is used.

TIMSEP
Specifies the separator used when accessing time result columns.

Note: This parameter applies only when *HMS is specified on the TIMFMT parameter.

***JOB:** The time separator specified for the job at precompile time is used. Use the Display Job (DSPJOB) command to determine the current value for the job.

':': A colon (:) is used.

'.': A period (.) is used.

',': A comma (,) is used.

' ': A blank ( ) is used.

***BLANK:** A blank ( ) is used.

REPLACE
Specifies whether a new program or SQL package is created when a program or SQL package of the same name exists in the same library. The value of this parameter is passed to the CRTCBLPGM command. More information about this parameter is in REPLACE parameter topic in the CL Reference section of the Information Center.

***YES:** A new program or SQL package is created, and any existing program or SQL package of the same name and type in the specified library is moved to QRPLOBJ.

***NO:** A new program or SQL package is not created if an object of the same name and type already exists in the specified library.

RDB
Specifies the name of the relational database where the SQL package object is created.

***LOCAL:** The program is created as a distributed SQL program. The SQL statements will access the local database. An SQL package object is not created as part of the precompile process. The Create Structured Query Language Package (CRTSQLPKG) command can be used.

relational-database-name: Specify the name of the relational database where the new SQL package object is to be created. When the name of the local relational database is specified, the program created is still a distributed SQL program. The SQL statements will access the local database.

***NONE:** An SQL package object is not created. The program object is not a distributed program and the Create Structured Query Language Package (CRTSQLPKG) command cannot be used.

USER
Specifies the user name sent to the remote system when starting the conversation. This parameter is valid only when RDB is specified.

***CURRENT:** The user profile under which the current job is running is used.

user-name: Specify the user name to be used for the application server job.

PASSWORD
Specifies the password to be used on the remote system. This parameter is valid only if RDB is specified.

***NONE:** No password is sent. If this value is specified, USER(*CURRENT) must also be specified.

password: Specify the password of the user name specified on the USER parameter.

RDBCNNMTH
Specifies the semantics used for CONNECT statements. Refer to the CONNECT (TYPE1) and CONNECT (TYPE2) in the *SQL Reference* book for more information.

***DUW:** CONNECT (Type 2) semantics are used to support distributed unit of work. Consecutive CONNECT statements to additional relational databases do not result in disconnection of previous connections.

***RUW:** CONNECT (Type 1) semantics are used to support remote unit of work. Consecutive CONNECT statements result in the previous connection being disconnected before a new connection is established.

DFTRDBCOL

Specifies the schema name used for the unqualified names of tables, views, indexes, and SQL packages. This parameter applies only to static SQL statements

***NONE:** The naming convention defined on the OPTION parameter is used.

schema-name: Specify the name of the schema identifier. This value is used instead of the naming convention specified on the OPTION parameter.

DYNDFTCOL

Specifies whether the default schema name specified for the DFTRDBCOL parameter is also used for dynamic statements.

***NO:** Do not use the value specified on the DFTRDBCOL parameter for unqualified names of tables, views, indexes, and SQL packages for dynamic SQL statements. The naming convention specified on the OPTION parameter is used.

***YES:** The schema name specified on the DFTRDBCOL parameter will be used for the unqualified names of the tables, views, indexes, and SQL packages in dynamic SQL statements.

SQLPKG

Specifies the qualified name of the SQL package created on the relational database specified on the RDB parameter of this command.

The library values are:

***PGMLIB:** The package is created in the library with the same name as the library containing the program.

library-name: Specify the name of the library where the package is created.

***PGM:** The package name is the same as the program name.

package-name: Specify the name of the package created on the remote database specified on the RDB parameter.

SQLPATH

Specifies the path to be used to find procedures, functions, and user defined types in static SQL statements.

***NAMING:** The path used depends on the naming convention specified on the OPTION parameter.

For *SYS naming, the path used is *LIBL, the current library list at runtime.

For *SQL naming, the path used is ″QSYS″, ″QSYS2″, ″userid″, where ″userid″ is the value of the USER special register. If a schema-name is specified on the DFTRDBCOL parameter, the schema-name takes the place of userid.

***LIBL:** The path used is the library list at runtime.

| *schema-name:* Specify a list of one or more schema names. A maximum of 268
| individual schemas may be specified.

SQLCURRULE
Specifies the semantics used for SQL statements.

***DB2:** The semantics of all SQL statements will default to the rules established for DB2. The following semantics are controlled by this option:

- Hexadecimal constants are treated as character data.

***STD:** The semantics of all SQL statements will default to the rules established by the ISO and ANSI SQL standards. The following semantics are controlled by this option:

- Hexadecimal constants are treated as binary data.

SAAFLAG
Specifies the IBM SQL flagging function. This parameter flags SQL statements to verify whether they conform to IBM SQL syntax. More information about which IBM database products IBM SQL syntax is in the *DRDA IBM SQL Reference*, SC26-3255-00.

***NOFLAG:** The precompiler does not check to see whether SQL statements conform to IBM SQL syntax.

***FLAG:** The precompiler checks to see whether SQL statements conform to IBM SQL syntax.

FLAGSTD
Specifies the American National Standards Institute (ANSI) flagging function. This parameter flags SQL statements to verify whether they conform to the following standards.

```
ANSI X3.135-1992 entry
ISO 9075-1992 entry
FIPS 127.2 entry
```

***NONE:** The precompiler does not check to see whether SQL statements conform to ANSI standards.

***ANS:** The precompiler checks to see whether SQL statements conform to ANSI standards.

PRTFILE
Specifies the qualified name of the printer device file to which the listing is directed. The file must have a minimum record length of 132 bytes or information is lost.

The name of the printer file can be qualified by one of the following library values:

***LIBL:** All libraries in the job's library list are searched until the first match is found.

***CURLIB:** The current library for the job is searched. If no library is specified as the current library for the job, the QGPL library is used.

QSYSPRT: If a file name is not specified, the precompiler printout is directed to the IBM-supplied printer file QSYSPRT.

printer-file-name: Specify the name of the printer device file to which the precompiler printout is directed.

SRTSEQ
Specifies the sort sequence table to be used for string comparisons in SQL statements.

Note: *HEX must be specified for this parameter on distributed applications where the application server is not on an iSeries system or the release level is prior to V2R3M0.

*JOB: The SRTSEQ value for the job is retrieved during the precompile.

***JOBRUN:** The SRTSEQ value for the job is retrieved when the program is run. For distributed applications, SRTSEQ(*JOBRUN) is valid only when LANGID(*JOBRUN) is also specified.

***LANGIDUNQ:** The unique-weight sort table for the language specified on the LANGID parameter is used.

***LANGIDSHR:** The shared-weight sort table for the language specified on the LANGID parameter is used.

***HEX:** A sort sequence table is not used. The hexadecimal values of the characters are used to determine the sort sequence.

The name of the sort sequence table can be qualified by one of the following library values:

*LIBL: All libraries in the job's library list are searched until the first match is found.

***CURLIB:** The current library for the job is searched. If no library is specified as the current library for the job, the QGPL library is used.

library-name: Specify the name of the library to be searched.

table-name: Specify the name of the sort sequence table to be used.

LANGID

Specifies the language identifier to be used when SRTSEQ(*LANGIDUNQ) or SRTSEQ(*LANGIDSHR) is specified.

*JOB: The LANGID value for the job is retrieved during the precompile.

***JOBRUN:** The LANGID value for the job is retrieved when the program is run. For distributed applications, LANGID(*JOBRUN) is valid only when SRTSEQ(*JOBRUN) is also specified.

language-id: Specify a language identifier to be used by the program.

USRPRF

Specifies the user profile that is used when the compiled program object is run, including the authority that the program object has for each object in static SQL statements. The profile of either the program owner or the program user is used to control which objects can be used by the program object.

*NAMING: The user profile is determined by the naming convention. If the naming convention is *SQL, USRPRF(*OWNER) is used. If the naming convention is *SYS, USRPRF(*USER) is used.

***USER:** The profile of the user running the program object is used.

***OWNER:** The user profiles of both the program owner and the program user are used when the program is run.

DYNUSRPRF

Specifies the user profile used for dynamic SQL statements.

***USER:** Local dynamic SQL statements are run under the user profile of the job. Distributed dynamic SQL statements are run under the user profile of the application server job.

***OWNER:** Local dynamic SQL statements are run under the user profile of the program's owner. Distributed dynamic SQL statements are run under the user profile of the SQL package's owner.

TOSRCFILE
Specifies the qualified name of the source file that is to contain the output source member that has been processed by the SQL precompiler. If the specified source file is not found, it will be created. The output member will have the same name as the name that is specified for the SRCMBR parameter.

The possible library values are:

QTEMP: The library QTEMP will be used.

***LIBL:** The job's library list is searched for the specified file. If the file is not found in any library in the library list, the file will be created in the current library.

***CURLIB:** The current library for the job will be used. If no library is specified as the current library for the job, the QGPL library will be used.

library-name: Specify the name of the library that is to contain the output source file.

QSQLTEMP: The source file QSQLTEMP will be used.

source-file-name: Specify the name of the source file to contain the output source member.

TEXT
Specifies the text that briefly describes the program and its function. More information about this parameter is in the TEXT parameter topic in the CL Reference section of the Information Center.

***SRCMBRTXT:** The text is taken from the source file member being used to create the COBOL program. Text for a database source member can be added or changed by using the Start Source Entry Utility (STRSEU) command, or by using either the Add Physical File Member (ADDPFM) or Change Physical File Member (CHGPFM) command. If the source file is an inline file or a device file, the text is blank.

***BLANK:** Text is not specified.

'description': Specify no more than 50 characters of text, enclosed in apostrophes.

Example:

```
CRTSQLCBL   PGM(ACCTS/STATS)  SRCFILE(ACCTS/ACTIVE)
  TEXT('Statistical analysis program for
  active accounts')
```

This command runs the SQL precompiler which precompiles the source and stores the changed source in the member STATS in file QSQLTEMP in library QTEMP. The COBOL compiler is called to create program STATS in library ACCTS using the source member created by the SQL precompiler.

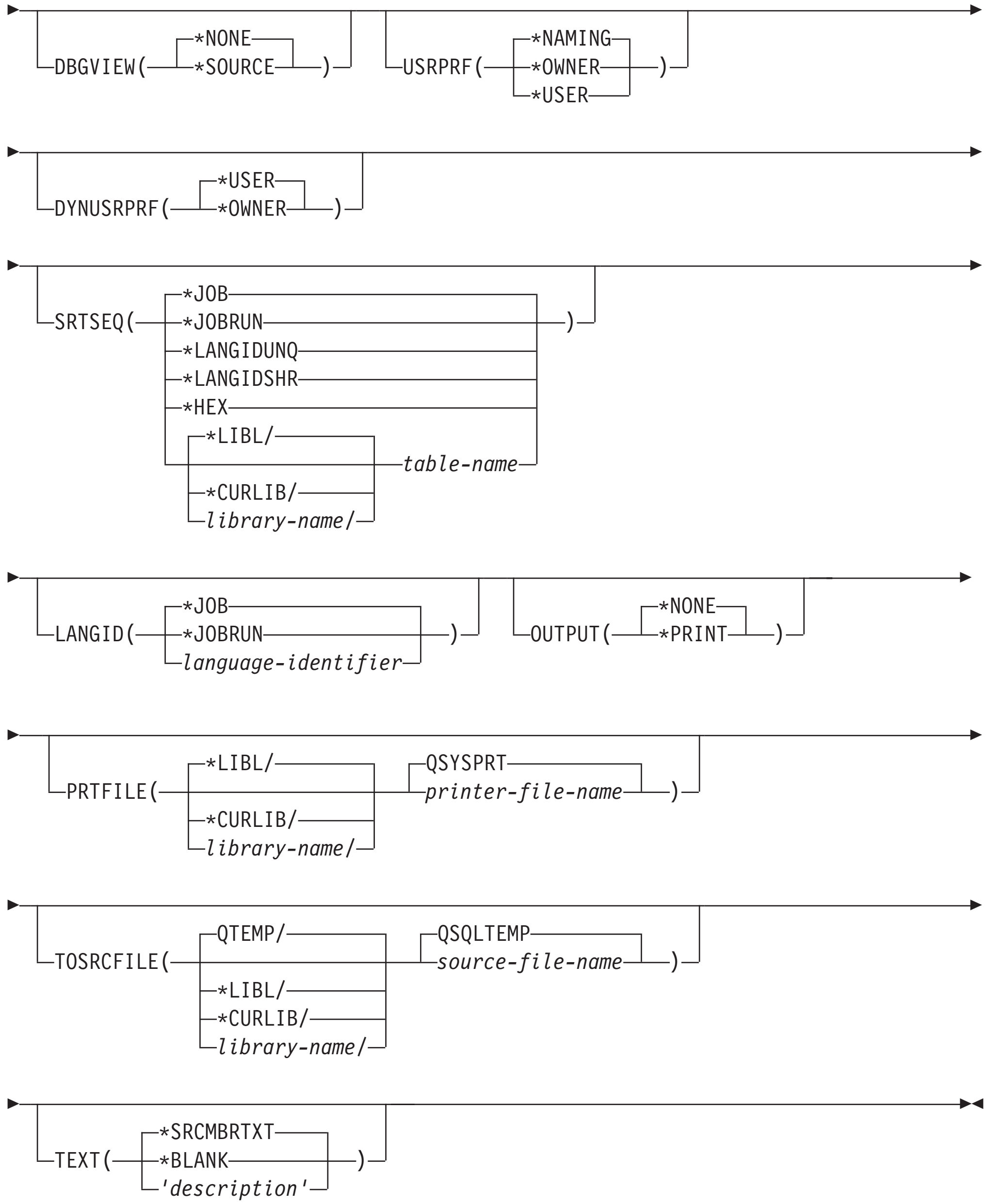
CRTSQLCBLI (Create SQL ILE COBOL Object) Command

Job: B,I   Pgm: B,I   REXX: B,I   Exec

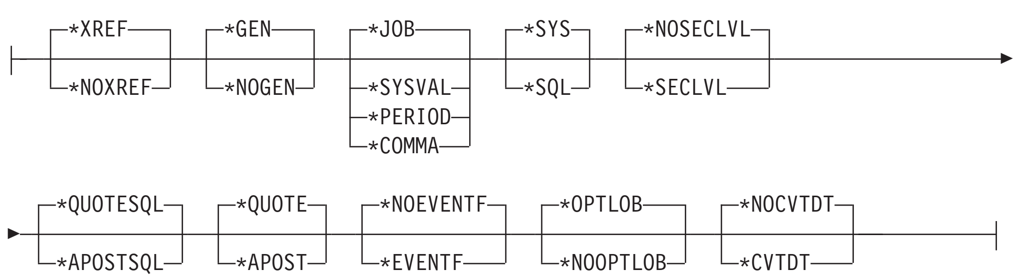
```
                    ┌─*CURLIB/──────┐
►►──CRTSQLCBLI──OBJ(─┼───────────────┼─object-name──)──────────────────────────────►
                    └─library-name/─┘

►──┬───────────────────────────────────────────────────────────┬──────────────────►
   │              ┌─*LIBL/────────┐  ┌─QCBLLESRC────────┐      │
   └─SRCFILE(─────┼───────────────┼──┴─source-file-name─┴──)──┘
                  ├─*CURLIB/──────┤
                  └─library-name/─┘

                                                           (1)
►──┬──────────────────────────────────────────────┬──────────────────────────────►
   │            ┌─*OBJ────────────────────┐        │
   └─SRCMBR(────┴─source-file-member-name─┴──)────┘

►──┬───────────────────────────────────┬──┬──────────────────────────────┬───────►
   └─OPTION(──┤ OPTION Details ├──)──┘  │            ┌─*CURRENT─┐        │
                                        └─TGTRLS(────┼─*PRV─────┼──)────┘
                                                     └─VxRxMx───┘

►──┬──────────────────────────────┬───────────────────────────────────────────────►
   │            ┌─*PGM─────┐      │
   └─OBJTYPE(───┼─*MODULE──┼──)──┘
                └─*SRVPGM──┘

►──┬───────────────────────────────────────────────────────────┬──────────────────►
   │              ┌─*LIBL/────────┐  ┌─*SRCFILE─────────┐      │
   └─INCFILE(─────┼───────────────┼──┴─source-file-name─┴──)──┘
                  ├─*CURLIB/──────┤
                  └─library-name/─┘

►──┬────────────────────────┬──┬──────────────────────────────────────┬──────────►
   │           ┌─*UR───┐    │  │              ┌─*ENDACTGRP─┐           │
   │           ├─*CHG──┤    │  └─CLOSQLCSR(───┴─*ENDMOD────┴──)──────┘
   └─COMMIT(───┼─*ALL──┼──)─┘
               ├─*RS───┤
               ├─*CS───┤
               ├─*NONE─┤
               ├─*NC───┤
               └─*RR───┘

►──┬────────────────────────────────────┬──┬──────────────────────────────┬──────►
   │              ┌─*OPTIMIZE─┐          │  │           ┌─*ALLREAD─┐       │
   └─ALWCPYDTA(───┼─*YES──────┼──)──────┘  └─ALWBLK(────┼─*NONE────┼──)───┘
                  └─*NO───────┘                         └─*READ────┘
```

CRTSQLCBLI

```
►─┬──────────────────────────────┬─┬────────────────────────────────────────┬──►
  │            ┌─*NO──┐          │ │           ┌─10─────────────┐           │
  └─DLYPRP(────┴─*YES─┴──)───────┘ └─GENLVL(───┴─severity-level─┴──)────────┘

►─┬───────────────────────────────┬─┬──────────────────────────────┬──►
  │            ┌─*JOB─┐           │ │           ┌─*JOB───┐         │
  └─DATFMT(────┼─*USA─┼──)────────┘ └─DATSEP(───┼─'/'────┼──)──────┘
               ├─*ISO─┤                         ├─'.'────┤
               ├─*EUR─┤                         ├─','────┤
               ├─*JIS─┤                         ├─'-'────┤
               ├─*MDY─┤                         ├─' '────┤
               ├─*DMY─┤                         └─*BLANK─┘
               ├─*YMD─┤
               └─*JUL─┘

►─┬───────────────────────────────┬─┬──────────────────────────────┬──►
  │            ┌─*HMS─┐           │ │           ┌─*JOB───┐         │
  └─TIMFMT(────┼─*USA─┼──)────────┘ └─TIMSEP(───┼─':'────┼──)──────┘
               ├─*ISO─┤                         ├─'.'────┤
               ├─*EUR─┤                         ├─','────┤
               └─*JIS─┘                         ├─' '────┤
                                                └─*BLANK─┘

►─┬─────────────────────────────┬─┬────────────────────────────────────────────────┬──►
  │             ┌─*YES─┐        │ │        ┌─*LOCAL─────────────────────────┐      │
  └─REPLACE(────┴─*NO──┴──)─────┘ └─RDB(───┼─relational-database-name───────┼──)───┘
                                           └─*NONE──────────────────────────┘

►─┬───────────────────────────────┬─┬─────────────────────────────────┬──►
  │         ┌─*CURRENT──┐         │ │             ┌─*NONE────┐        │
  └─USER(───┴─user-name─┴──)──────┘ └─PASSWORD(───┴─password─┴──)─────┘

►─┬───────────────────────────────┬─┬──────────────────────────────────────┬──►
  │              ┌─*DUW─┐         │ │              ┌─*NONE───────────┐     │
  └─RDBCNNMTH(───┴─*RUW─┴──)──────┘ └─DFTRDBCOL(───┴─collection-name─┴──)──┘

►─┬───────────────────────────────┬──►
  │              ┌─*NO──┐         │
  └─DYNDFTCOL(───┴─*YES─┴──)──────┘

►─┬─────────────────────────────────────────────────────────────┬──►
  │           ┌─*OBJLIB/──────┐  ┌─*OBJ─────────┐             │
  └─SQLPKG(───┼───────────────┼──┴─package-name─┴──)──────────┘
              └─library-name/─┘

►─┬──────────────────────────────────────────────────┬─┬─────────────────────────────┬──►
  │            ┌─*NAMING──────────────────────┐      │ │               ┌─*DB2─┐      │
  └─SQLPATH(───┼─*LIBL────────────────────────┼──)───┘ └─SQLCURRULE(───┴─*STD─┴──)───┘
               │   ┌──────────────────────┐   │
               │   ▼                      │   │
               └───┴─collection-name──────┴───┘

►─┬──────────────────────────────────┬─┬────────────────────────────┬──►
  │             ┌─*NOFLAG─┐          │ │             ┌─*NONE─┐      │
  └─SAAFLAG(────┴─*FLAG───┴──)───────┘ └─FLAGSTD(────┴─*ANS──┴──)───┘
```



OPTION Details:



Notes:

1 All parameters preceding this point can be specified in positional form.

CRTSQLCBLI

Purpose:

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.

Parameters:

OBJ
Specifies the qualified name of the object being created.

***CURLIB:** The new object is created in the current library for the job. If no library is specified as the current library for the job, the QGPL library is used.

library-name: Specify the name of the library where the object is created.

object-name: Specify the name of the object that is being created.

SRCFILE
Specifies the qualified name of the source file that contains the COBOL source with SQL statements.

The name of the source file can be qualified by one of the following library values:

***LIBL** All libraries in the job's library list are searched until the first match is found.

***CURLIB** The current library for the job is searched. If no library is specified as the current library for the job, the QGPL library is used.

library-name: Specify the name of the library to be searched.

QCBLLESRC: If the source file name is not specified, the source file QCBLLESRC contains the COBOL source.

source-file-name: Specify the name of the source file that contains the COBOL source.

SRCMBR
Specifies the name of the source file member that contains the COBOL source. This parameter is specified only if the source file name in the SRCFILE parameter is a database file. If this parameter is not specified, the OBJ name specified on the OBJ parameter is used.

***OBJ:** Specifies that the COBOL source is in the member of the source file that has the same name as that specified on the OBJ parameter.

source-file-member-name: Specify the name of the member that contains the COBOL source.

OPTION
Specifies whether one or more of the following options are used when the COBOL source is precompiled. If an option is specified more than once, or if two options conflict, the last option specified is used.

Element 1: Cross-Reference Options

***XREF:** The precompiler cross-references items in the program to the statement numbers in the program that refer to those items.

***NOXREF:** The precompiler does not cross-reference names.

Element 2: Program Creation Options

***GEN:** The precompiler creates the object that is specified by the OBJTYPE parameter.

***NOGEN:** The precompiler does not call the ILE COBOL compiler, and a module, program, service program, or SQL package are not created.

Element 3: Decimal Point Options

***JOB:** The value used as the decimal point for numeric constants in SQL is the representation of decimal point specified for the job at precompile time.

***SYSVAL:** The value used as the decimal point for numeric constants in SQL statements is the QDECFMT system value.

***PERIOD:** The value used as the decimal point for numeric constants in SQL statements is a period (.).

Note: If QDECFMT specifies that the value used as the decimal point is a comma (,), any numeric constants in lists (such as in the SELECT clause or the VALUES clause) must be separated by a comma (,) followed by a blank ( ). For example, VALUES(1,1, 2,23, 4,1) is equivalent to VALUES(1.1,2.23,4.1) in which the decimal point is a period (.).

***COMMA:** The value used as the decimal point for numeric constants in SQL statements is a comma (,).

Note: Any numeric constants in lists (such as in the SELECT clause or the VALUES clause) must be separated by a comma (,) followed by a blank( ). For example, VALUES(1,1, 2,23, 4,1) is equivalent to VALUES(1.1,2.23,4.1) where the decimal point is a period(.).

Element 4: Naming Convention Options

***SYS:** The system naming convention (library-name/file-name) is used.

***SQL:** The SQL naming convention is used (schema-name.table-name).

When creating a program on a remote database other than an iSeries system, *SQL must be specified as the naming convention.

Element 5: Second-Level Message Text Option

***NOSECLVL:** Second-level text descriptions are not added to the listing.

***SECLVL:** Second-level text with replacement data is added for all messages on the listing.

Element 6: String Delimiter Options

***QUOTESQL:** A double quote (") is the string delimiter in the SQL statements.

***APOSTSQL:** An apostrophe (') is the string delimiter in the SQL statements.

Element 7: Literal Options

***QUOTE:** A double quote (") is used for literals which are not numeric and Boolean literals in the COBOL statements.

***APOST:** An apostrophe (') is used for literals which are not numeric and Boolean literals in the COBOL statements.

Element 8: Event File Creation

***NOEVENTF:** The compiler will not produce an event file for use by CoOperative Development Environment/400 (CODE/400).

***EVENTF:** The compiler produces an event file for use by CoOperative Development Environment/400 (CODE/400). The event file will be created as a member in the file EVFEVENT in your source library. CODE/400 uses this file to offer error feedback integrated with the CODE/400 editor. This option is normally specified by CODE/400 on your behalf.

Element 9: Large Object Optimization for DRDA

***OPTLOB:** The first FETCH for a cursor determines how the cursor will be used for LOBs (Large Objects) on all subsequent FETCHes. This option remains in effect until the cursor is closed.

If the first FETCH uses a LOB locator to access a LOB column, no subsequent FETCH for that cursor can fetch that LOB column into a LOB host variable.

If the first FETCH places the LOB column into a LOB host variable, no subsequent FETCH for that cursor can use a LOB locator for that column.

***NOOPTLOB:** There is no restriction on whether a column is retrieved into a LOB locator or into a LOB host variable. This option can cause performance to degrade.

Element 10: Date conversion

***NOCVTDT:** Specifies that date, time, and timestamp data types that are retrieved from externally-described database files are to be processed using the date, time, and timestamp data types.

***CVTDT:** Specifies that date, time, and timestamp data types that are retrieved from externally-described database files are to be processed as fixed-length character fields.

TGTRLS

Specifies the release of the operating system on which the user intends to use the object being created.

In the examples given for the *CURRENT and *PRV values, and when specifying the *release-level* value, the format VxRxMx is used to specify the release, where Vx is the version, Rx is the release, and Mx is the modification level. For example, V2R3M0 is version 2, release 3, modification level 0.

***CURRENT:** The object is to be used on the release of the operating system currently running on the user's system. For example, if V2R3M5 is running on the system, *CURRENT means the user intends to use the object on a system with V2R3M5 installed. The user can also use the object on a system with any subsequent release of the operating system installed.

Note: If V2R3M5 is running on the system, and the object is to be used on a system with V2R3M0 installed, specify TGTRLS(V2R3M0) not

TGTRLS(*CURRENT).

***PRV:** The object is to be used on the previous release with modification level 0 of the operating system. For example, if V2R3M5 is running on the user's system, *PRV means the user intends to use the object on a system with V2R2M0 installed. The user can also use the object on a system with any subsequent release of the operating system installed.

release-level: Specify the release in the format VxRxMx. The object can be used on a system with the specified release or with any subsequent release of the operating system installed.

Valid values depend on the current version, release, and modification level, and they change with each new release. If you specify a release-level which is earlier than the earliest release level supported by this command, an error message is sent indicating the earliest supported release.

OBJTYPE

Specifies the type of object being created.

***PGM:** The SQL precompiler issues the CRTBNDCBL command to create the bound program.

***MODULE:** The SQL precompiler issues the CRTCBLMOD command to create the module.

***SRVPGM:** The SQL precompiler issues the CRTCBLMOD and CRTSRVPGM commands to create the service program.

Notes:

1. When OBJTYPE(*PGM) or OBJTYPE(*SRVPGM) is specified and the RDB parameter is also specified, the CRTSQLPKG command is issued by the SQL precompiler after the program has been created. When OBJTYPE(*MODULE) is specified, an SQL package is not created and you must issue the CRTSQLPKG command after the CRTPGM or CRTSRVPGM command has created the program.
2. If *NOGEN is specified, only the SQL temporary source member is generated and a module, program, service program, or SQL package are not created.

INCFILE

Specifies the qualified name of the source file that contains members included in the program with any SQL INCLUDE statement.

The name of the source file can be qualified by one of the following library values:

***LIBL:** All libraries in the job's library list are searched until the first match is found.

***CURLIB:** The current library for the job is searched. If no library is specified as the current library for the job, the QGPL library is used.

library-name: Specify the name of the library to be searched.

***SRCFILE:** The qualified source file specified in the SRCFILE parameter contains the source file members specified on any SQL INCLUDE statement.

source-file-name: Specify the name of the source file that contains the source file members specified on any SQL INCLUDE statement. The record length of the source file specified here must be no less than the record length of the source file specified on the SRCFILE parameter.

COMMIT

Specifies whether SQL statements in the compiled unit are run under commitment control. Files referred to in the host language source are not affected by this option. Only SQL tables, SQL views, and SQL packages referred to in SQL statements are affected.

***CHG or *UR:** Specifies the objects referred to in SQL ALTER, CALL, COMMENT ON, CREATE, DROP, GRANT, LABEL ON, RENAME, and REVOKE statements and the rows updated, deleted, and inserted are locked until the end of the unit of work (transaction). Uncommitted changes in other jobs can be seen.

***ALL or *RS:** Specifies the objects referred to in SQL ALTER, CALL, COMMENT ON, CREATE, DROP, GRANT, LABEL ON, RENAME, and REVOKE statements and the rows selected, updated, deleted, and inserted are locked until the end of the unit of work (transaction). Uncommitted changes in other jobs cannot be seen.

***CS:** Specifies the objects referred to in SQL ALTER, CALL, COMMENT ON, CREATE, DROP, GRANT, LABEL ON, RENAME, and REVOKE statements and the rows updated, deleted, and inserted are locked until the end of the unit of work (transaction). A row that is selected, but not updated, is locked until the next row is selected. Uncommitted changes in other jobs cannot be seen.

***NONE or *NC:** Specifies that commitment control is not used. Uncommitted changes in other jobs can be seen. If the SQL DROP SCHEMA statement is included in the program, *NONE or *NC must be used. If a relational database is specified on the RDB parameter and the relational database is on a system that is not on an iSeries, *NONE or *NC cannot be specified.

***RR:** Specifies the objects referred to in SQL ALTER, CALL, COMMENT ON, CREATE, DROP, GRANT, LABEL ON, RENAME, and REVOKE statements and the rows selected, updated, deleted, and inserted are locked until the end of the unit of work (transaction). Uncommitted changes in other jobs cannot be seen. All tables referred to in SELECT, UPDATE, DELETE, and INSERT statements are locked exclusively until the end of the unit of work (transaction).

CLOSQLCSR

Specifies when SQL cursors are implicitly closed, SQL prepared statements are implicitly discarded, and LOCK TABLE locks are released. SQL cursors are explicitly closed when you issue the CLOSE, COMMIT, or ROLLBACK (without HOLD) SQL statements.

***ENDACTGRP:** SQL cursors are closed, SQL prepared statements are implicitly discarded, and LOCK TABLE locks are released when the activation group ends.

***ENDMOD:** SQL cursors are closed and SQL prepared statements are implicitly discarded when the module is exited. LOCK TABLE locks are released when the activation group ends.

ALWCPYDTA

Specifies whether a copy of the data can be used in a SELECT statement.

***OPTIMIZE:** The system determines whether to use the data retrieved directly from the database or to use a copy of the data. The decision is based on which method provides the best performance. If COMMIT is *CHG or *CS and ALWBLK is not *ALLREAD, or if COMMIT is *ALL or *RR, then a copy of the data is used only when it is necessary to run a query.

***YES:** A copy of the data is used only when necessary.

***NO:** A copy of the data is not allowed. If a temporary copy of the data is required to perform the query, an error message is returned.

ALWBLK

Specifies whether the database manager can use record blocking, and the extent to which blocking can be used for read-only cursors.

***ALLREAD:** Rows are blocked for read-only cursors if *NONE or *CHG is specified on the COMMIT parameter. All cursors in a program that are not explicitly able to be updated are opened for read-only processing even though EXECUTE or EXECUTE IMMEDIATE statements may be in the program.

Specifying *ALLREAD:

- Allows record blocking under commitment control level *CHG in addition to the blocking allowed for *READ.
- Can improve the performance of almost all read-only cursors in programs, but limits queries in the following ways:
  - The Rollback (ROLLBACK) command, a ROLLBACK statement in host languages, or the ROLLBACK HOLD SQL statement does not reposition a read-only cursor when *ALLREAD is specified.
  - Dynamic running of a positioned UPDATE or DELETE statement (for example, using EXECUTE IMMEDIATE), cannot be used to update a row in a cursor unless the DECLARE statement for the cursor includes the FOR UPDATE clause.

***NONE:** Rows are not blocked for retrieval of data for cursors.

Specifying *NONE:

- Guarantees that the data retrieved is current.
- May reduce the amount of time required to retrieve the first row of data for a query.
- Stops the database manager from retrieving a block of data rows that is not used by the program when only the first few rows of a query are retrieved before the query is closed.
- Can degrade the overall performance of a query that retrieves a large number of rows.

***READ:** Records are blocked for read-only retrieval of data for cursors when:

- *NONE is specified on the COMMIT parameter, which indicates that commitment control is not used.
- The cursor is declared with a FOR READ ONLY clause or there are no dynamic statements that could run a positioned UPDATE or DELETE statement for the cursor.

Specifying *READ can improve the overall performance of queries that meet the above conditions and retrieve a large number of records.

DLYPRP

Specifies whether the dynamic statement validation for a PREPARE statement is delayed until an OPEN, EXECUTE, or DESCRIBE statement is run. Delaying validation improves performance by eliminating redundant validation.

***NO:** Dynamic statement validation is not delayed. When the dynamic statement is prepared, the access plan is validated. When the dynamic statement is used in an OPEN or EXECUTE statement, the access plan is revalidated. Because the authority or the existence of objects referred to by the dynamic statement may change, you must still check the SQLCODE or SQLSTATE after issuing the OPEN or EXECUTE statement to ensure that the dynamic statement is still valid.

***YES:** Dynamic statement validation is delayed until the dynamic statement is used in an OPEN, EXECUTE, or DESCRIBE SQL statement. When the dynamic statement is used, the validation is completed and an access plan is built. If you specify *YES on this parameter, you should check the SQLCODE and SQLSTATE after running an OPEN, EXECUTE, or DESCRIBE statement to ensure that the dynamic statement is valid.

Note: If you specify *YES, performance is not improved if the INTO clause is used on the PREPARE statement or if a DESCRIBE statement uses the dynamic statement before an OPEN is issued for the statement.

GENLVL

Specifies the severity level at which the create operation fails. If errors occur that have a severity level greater than this value, the operation ends.

10: The default severity level is 10.

severity-level: Specify a value ranging from 0 through 40.

DATFMT

Specifies the format used when accessing date result columns. All output date fields are returned in the specified format. For input date strings, the specified value is used to determine whether the date is specified in a valid format.

Note: An input date string that uses the format *USA, *ISO, *EUR, or *JIS is always valid.

If a relational database is specified on the RDB parameter and the database is on a system that is not an iSeries system, then *USA, *ISO, *EUR, or *JIS must be specified.

***JOB:** The format specified for the job is used. Use the Display Job (DSPJOB) command to determine the current date format for the job.

***USA:** The United States date format (mm/dd/yyyy) is used.

***ISO:** The International Organization for Standardization (ISO) date format (yyyy-mm-dd) is used.

***EUR:** The European date format (dd.mm.yyyy) is used.

***JIS:** The Japanese Industrial Standard date format (yyyy-mm-dd) is used.

***MDY:** The date format (mm/dd/yy) is used.

***DMY:** The date format (dd/mm/yy) is used.

***YMD:** The date format (yy/mm/dd) is used.

***JUL:** The Julian date format (yy/ddd) is used.

DATSEP

Specifies the separator used when accessing date result columns.

Note: This parameter applies only when *JOB, *MDY, *DMY, *YMD, or *JUL is specified on the DATFMT parameter.

***JOB:** The date separator specified for the job at precompile time is used. Use the Display Job (DSPJOB) command to determine the current value for the job.

'/': A slash (/) is used.

'.': A period (.) is used.

',': A comma (,) is used.

'-': A dash (-) is used.

' ': A blank ( ) is used.

***BLANK:** A blank ( ) is used.

TIMFMT
Specifies the format used when accessing time result columns. For input time strings, the specified value is used to determine whether the time is specified in a valid format.

Note: An input date string that uses the format *USA, *ISO, *EUR, or *JIS is always valid.

If a relational database is specified on the RDB parameter and the database is on a system that is not another iSeries system, the time format must be *USA, *ISO, *EUR, *JIS, or *HMS with a time separator of a colon or period.

***HMS:** The **hh:mm:ss** format is used.

***USA:** The United States time format **hh:mm xx** is used, where **xx** is AM or PM.

***ISO:** The International Organization for Standardization (ISO) time format **hh.mm.ss** is used.

***EUR:** The European time format **hh.mm.ss** is used.

***JIS:** The Japanese Industrial Standard time format **hh:mm:ss** is used.

TIMSEP
Specifies the separator used when accessing time result columns.

Note: This parameter applies only when *HMS is specified on the TIMFMT parameter.

***JOB:** The time separator specified for the job at precompile time is used. Use the Display Job (DSPJOB) command to determine the current value for the job.

':': A colon (:) is used.

'.': A period (.) is used.

',': A comma (,) is used.

' ': A blank ( ) is used.

***BLANK:** A blank ( ) is used.

REPLACE
Specifies if a SQL module, program, service program or package is created when there is an existing SQL module, program, service program, or package of the same name and type in the same library. The value of this parameter is passed to the CRTCBLMOD, CRTBNDCBL, CRTSRVPGM, and CRTSQLPKG commands.

***YES:** A new SQL module, program, service program, or package is created, any existing SQL object of the same name and type in the specified library is moved to QRPLOBJ.

***NO:** A new SQL module, program, service program, or package is not created if an SQL object of the same name and type already exists in the specified library.

RDB
Specifies the name of the relational database where the SQL package object is created.

***LOCAL:** The program is created as a distributed SQL program. The SQL statements will access the local database. An SQL package object is not created as part of the precompile process. The Create Structured Query Language Package (CRTSQLPKG) command can be used.

relational-database-name: Specify the name of the relational database where the new SQL package object is to be created. When the name of the local relational database is specified, the program created is still a distributed SQL program. The SQL statements will access the local database.

***NONE:** An SQL package object is not created. The program object is not a distributed program and the Create Structured Query Language Package (CRTSQLPKG) command cannot be used.

USER
Specifies the user name sent to the remote system when starting the conversation. This parameter is valid only when RDB is specified.

***CURRENT:** The user profile under which the current job is running is used.

user-name: Specify the user name being used for the application server job.

PASSWORD
Specifies the password to be used on the remote system. This parameter is valid only if RDB is specified.

***NONE:** No password is sent. If this value is specified, USER(*CURRENT) must also be specified.

password: Specify the password of the user name specified on the USER parameter.

RDBCNNMTH
Specifies the semantics used for CONNECT statements. Refer to the CONNECT (TYPE1) and CONNECT (TYPE2) in the *SQL Reference* book for more information.

***DUW:** CONNECT (Type 2) semantics are used to support distributed unit of work. Consecutive CONNECT statements to additional relational databases do not result in disconnection of previous connections.

***RUW:** CONNECT (Type 1) semantics are used to support remote unit of work. Consecutive CONNECT statements result in the previous connection being disconnected before a new connection is established.

DFTRDBCOL

Specifies the schema name used for the unqualified names of tables, views, indexes, and SQL packages. This parameter applies only to static SQL statements.

***NONE:** The naming convention defined on the OPTION parameter is used.

schema-name: Specify the name of the schema identifier. This value is used instead of the naming convention specified on the OPTION parameter.

DYNDFTCOL

Specifies whether the default schema name specified for the DFTRDBCOL parameter is also used for dynamic statements.

***NO:** Do not use the value specified on the DFTRDBCOL parameter for unqualified names of tables, views, indexes, and SQL packages for dynamic SQL statements. The naming convention specified on the OPTION parameter is used.

***YES:** The schema name specified on the DFTRDBCOL parameter will be used for the unqualified names of the tables, views, indexes, and SQL packages in dynamic SQL statements.

SQLPKG

Specifies the qualified name of the SQL package created on the relational database specified on the RDB parameter of this command.

The possible library values are:

***OBJLIB:** The package is created in the library with the same name as the library specified on the OBJ parameter.

library-name: Specify the name of the library where the package is created.

***OBJ:** The name of the SQL package is the same as the object name specified on the OBJ parameter.

package-name: Specify the name of the SQL package. If the remote system is not an iSeries system, no more than 8 characters can be specified.

SQLPATH

Specifies the path to be used to find procedures, functions, and user defined types in static SQL statements.

***NAMING:** The path used depends on the naming convention specified on the OPTION parameter.

For *SYS naming, the path used is *LIBL, the current library list at runtime.

For *SQL naming, the path used is "QSYS", "QSYS2", "userid", where "userid" is the value of the USER special register. If a schema-name is specified on the DFTRDBCOL parameter, the schema-name takes the place of userid.

***LIBL:** The path used is the library list at runtime.

schema-name: Specify a list of one or more schema names. A maximum of 268 individual schemas may be specified.

SQLCURRULE

Specifies the semantics used for SQL statements.

***DB2:** The semantics of all SQL statements will default to the rules established for DB2. The following semantics are controlled by this option:

- Hexadecimal constants are treated as character data.

***STD:** The semantics of all SQL statements will default to the rules established by the ISO and ANSI SQL standards. The following semantics are controlled by this option:

- Hexadecimal constants are treated as binary data.

SAAFLAG
Specifies the IBM SQL flagging function. This parameter flags SQL statements to verify whether they conform to IBM SQL syntax. More information about which IBM database products IBM SQL syntax is in the *DRDA IBM SQL Reference*, SC26-3255-00.

***NOFLAG:** The precompiler does not check to see whether SQL statements conform to IBM SQL syntax.

***FLAG:** The precompiler checks to see whether SQL statements conform to IBM SQL syntax.

FLAGSTD
Specifies the American National Standards Institute (ANSI) flagging function. This parameter flags SQL statements to verify whether they conform to the following standards.

```
ANSI X3.135-1992 entry
ISO 9075-1992 entry
FIPS 127.2 entry
```

***NONE:** The precompiler does not check to see whether SQL statements conform to ANSI standards.

***ANS:** The precompiler checks to see whether SQL statements conform to ANSI standards.

DBGVIEW
Specifies the type of source debug information to be provided by the SQL precompiler.

***NONE:** The source view is not generated.

***SOURCE:** The SQL precompiler provides the source views for the root and if necessary, SQL INCLUDE statements. A view is provided which contains the statements generated by the precompiler.

USRPRF
Specifies the user profile that is used when the compiled program object is run, including the authority that the program object has for each object in static SQL statements. The profile of either the program owner or the program user is used to control which objects can be used by the program object.

***NAMING:** The user profile is determined by the naming convention. If the naming convention is *SQL, USRPRF(*OWNER) is used. If the naming convention is *SYS, USRPRF(*USER) is used.

***USER:** The profile of the user running the program object is used.

***OWNER:** The user profiles of both the program owner and the program user are used when the program is run.

DYNUSRPRF
Specifies the user profile to be used for dynamic SQL statements.

***USER:** For local programs, dynamic SQL statements run under the profile of the program's user. For distributed programs, dynamic SQL statements run under the profile of the SQL package's user.

***OWNER:** For local programs, dynamic SQL statements run under the profile of the program's owner. For distributed programs, dynamic SQL statements run under the profile of the SQL package's owner.

SRTSEQ
Specifies the sort sequence table to be used for string comparisons in SQL statements.

Note: *HEX must be specified for this parameter on distributed applications where the application server is not on an iSeries system or the release level is prior to V2R3M0.

***JOB:** The SRTSEQ value for the job is retrieved during the precompile.

***JOBRUN:** The SRTSEQ value for the job is retrieved when the program is run. For distributed applications, SRTSEQ(*JOBRUN) is valid only when LANGID(*JOBRUN) is also specified.

***LANGIDUNQ:** The unique-weight sort table for the language specified on the LANGID parameter is used.

The name of the table name can be qualified by one of the following library values:

***LIBL:** All libraries in the job's library list are searched until the first match is found.

***CURLIB:** The current library for the job is searched. If no library is specified as the current library for the job, the QGPL library is used.

library-name: Specify the name of the library to be searched.

***LANGIDSHR:** The sort sequence table uses the same weight for multiple characters, and is the shared-weight sort sequence table associated with the language specified on the LANGID parameter.

***HEX:** A sort sequence is not used. The hexadecimal values of the characters are used to determine the sort sequence.

table-name: Specify the name of the sort sequence table to be used.

LANGID
Specifies the language identifier to be used when SRTSEQ(*LANGIDUNQ) or SRTSEQ(*LANGIDSHR) is specified.

***JOB:** The LANGID value for the job is retrieved during the precompile.

***JOBRUN:** The LANGID value for the job is retrieved when the program is run. For distributed applications, LANGID(*JOBRUN) is valid only when SRTSEQ(*JOBRUN) is also specified.

language-identifier: Specify a language identifier.

OUTPUT
Specifies whether the precompiler listing is generated.

***NONE:** The precompiler listing is not generated.

***PRINT:** The precompiler listing is generated.

PRTFILE
Specifies the qualified name of the printer device file to which the precompiler

printout is directed. The file must have a minimum length of 132 bytes. If a file with a record length of less than 132 bytes is specified, information is lost.

The name of the printer file can be qualified by one of the following library values:

***LIBL** All libraries in the job's library list are searched until the first match is found.

***CURLIB** The current library for the job is searched. If no library is specified as the current library for the job, the QGPL library is used.

library-name: Specify the name of the library to be searched.

QSYSPRT: If a file name is not specified, the precompiler printout is directed to the IBM-supplied printer file QSYSPRT.

printer-file-name: Specify the name of the printer device file to which the precompiler printout is directed.

TOSRCFILE

Specifies the qualified name of the source file that is to contain the output source member that has been processed by the SQL precompiler. If the specified source file is not found, it will be created. The output member will have the same name as the name that is specified for the SRCMBR parameter.

The possible library values are:

QTEMP: The library QTEMP will be used.

***LIBL:** The job's library list is searched for the specified file. If the file is not found in any library in the library list, the file will be created in the current library.

***CURLIB:** The current library for the job will be used. If no library is specified as the current library for the job, the QGPL library will be used.

library-name: Specify the name of the library that is to contain the output source file.

QSQLTEMP: The source file QSQLTEMP will be used.

source-file-name: Specify the name of the source file to contain the output source member.

TEXT

Specifies the text that briefly describes the printer file. More information about this parameter is in the TEXT parameter topic in the CL Reference section of the Information Center.

***SRCMBRTXT:** The text is taken from the source file member being used to create the COBOL program. Text can be added or changed for a database source member by using the Start Source Entry Utility (STRSEU) command, or by using either the Add Physical File Member (ADDPFM) or Change Physical File Member (CHGPFM) command. If the source file is an inline file or a device file, the text is blank.

***BLANK:** Text is not specified.

'description': Specify no more than 50 characters of text, enclosed in apostrophes.

Example:

```
CRTSQLCBLI  PAYROLL OBJTYPE(*MODULE) TEXT('Payroll Program')
```

This command runs the SQL precompiler which precompiles the source and stores the changed source in member PAYROLL in file QSQLTEMP in library QTEMP. The ILE COBOL compiler is called to create module PAYROLL in the current library by using the source member created by the SQL precompiler.

CRTSQLCI (Create Structured Query Language ILE C Object) Command

Job: B,I   Pgm: B,I   REXX: B,I   Exec

```
                   ┌─*CURLIB/──────┐
►►──CRTSQLCI──OBJ(─┼───────────────┼─object-name──)─────────────────────────►
                   └─library-name/─┘

►──┬──────────────────────────────────────────────────────────┬─────────────►
   │                ┌─*LIBL/────────┐ ┌─QCSRC────────────┐   │
   └─SRCFILE(───────┼───────────────┼─┴─source-file-name─┴─)─┘
                    ├─*CURLIB/──────┤
                    └─library-name/─┘

                                                    (1)
►──┬─────────────────────────────────────────────────────┬──────────────────►
   │            ┌─*OBJ────────────────────┐              │
   └─SRCMBR(────┴─source-file-member-name─┴──)───────────┘

►──┬──────────────────────────────────┬─┬──────────────────────────────┬────►
   └─OPTION(──┤ OPTION Details ├──)───┘ │           ┌─*CURRENT─┐       │
                                        └─TGTRLS(───┼─*PRV─────┼──)────┘
                                                    └─VxRxMx───┘

►──┬──────────────────────────────┬─────────────────────────────────────────►
   │            ┌─*MODULE─┐       │
   └─OBJTYPE(───┼─*PGM────┼──)────┘
                └─*SRVPGM─┘

►──┬──────────────────────────────────────────────────────────┬─────────────►
   │                ┌─*LIBL/────────┐ ┌─*SRCFILE─────────┐   │
   └─INCFILE(───────┼───────────────┼─┴─source-file-name─┴─)─┘
                    ├─*CURLIB/──────┤
                    └─library-name/─┘

►──┬─────────────────────────┬─┬──────────────────────────────────────┬─────►
   │           ┌─*UR───┐     │ │              ┌─*ENDACTGRP─┐          │
   └─COMMIT(───┼─*CHG──┼──)──┘ └─CLOSQLCSR(───┴─*ENDMOD────┴──)───────┘
               ├─*ALL──┤
               ├─*RS───┤
               ├─*CS───┤
               ├─*NONE─┤
               ├─*NC───┤
               └─*RR───┘
```

```
►─┬──────────────────────────────────┬─┬───────────────────────────┬──────────►
  │              ┌─*OPTIMIZE─┐       │ │          ┌─*ALLREAD─┐     │
  └─ALWCPYDTA(───┼─*YES──────┼───)───┘ └─ALWBLK(──┼─*NONE────┼──)──┘
                 └─*NO───────┘                    └─*READ────┘

►─┬──────────────────────────┬─┬────────────────────────────────────┬──────────►
  │         ┌─*NO──┐         │ │         ┌─10─────────────┐         │
  └─DLYPRP(─┴─*YES─┴──)──────┘ └─GENLVL(─┴─severity-level─┴──)──────┘

►─┬─────────────────────────────────┬─┬───────────────────────┬────────────────►
  │          ┌─*SRCFILE───┐         │ │         ┌─*JOB─┐      │
  └─MARGINS(─┴─left-right─┴──)──────┘ └─DATFMT(─┼─*USA─┼──)───┘
                                                ├─*ISO─┤
                                                ├─*EUR─┤
                                                ├─*JIS─┤
                                                ├─*MDY─┤
                                                ├─*DMY─┤
                                                ├─*YMD─┤
                                                └─*JUL─┘

►─┬─────────────────────────────┬─┬────────────────────────┬───────────────────►
  │         ┌─*JOB────┐         │ │         ┌─*HMS─┐       │
  └─DATSEP(─┼─'/'─────┼──)──────┘ └─TIMFMT(─┼─*USA─┼──)────┘
            ├─'.'─────┤                     ├─*ISO─┤
            ├─','─────┤                     ├─*EUR─┤
            ├─'-'─────┤                     └─*JIS─┘
            ├─' '─────┤
            └─*BLANK──┘

►─┬─────────────────────────────┬─┬────────────────────────────┬───────────────►
  │         ┌─*JOB────┐         │ │          ┌─*YES─┐          │
  └─TIMSEP(─┼─':'─────┼──)──────┘ └─REPLACE(─┴─*NO──┴──)───────┘
            ├─'.'─────┤
            ├─','─────┤
            ├─' '─────┤
            └─*BLANK──┘

►─┬─────────────────────────────────────────┬─┬───────────────────────────┬────►
  │      ┌─*LOCAL─────────────────┐         │ │       ┌─*CURRENT──┐       │
  └─RDB(─┼─relational-database-name─┼──)────┘ └─USER(─┴─user-name─┴──)────┘
         └─*NONE──────────────────┘

►─┬──────────────────────────────┬─┬────────────────────────────┬──────────────►
  │           ┌─*NONE────┐       │ │            ┌─*DUW─┐        │
  └─PASSWORD(─┴─password─┴──)────┘ └─RDBCNNMTH(─┴─*RUW─┴──)─────┘

►─┬─────────────────────────────────────┬─┬───────────────────────────┬────────►
  │             ┌─*NONE───────────┐     │ │            ┌─*NO──┐       │
  └─DFTRDBCOL(──┴─collection-name─┴──)──┘ └─DYNDFTCOL(─┴─*YES─┴──)────┘
```

```
►─┬──────────────────────────────────────┬─┬─────────────────────────────────────────────────┬──►
  │            ┌─*OBJ──────────┐         │ │            ┌─*NAMING─────────────────────┐    │
  └─SQLPKG(────┴─package-name──┴──)──────┘ └─SQLPATH(───┼─*LIBL───────────────────────┼─)──┘
                                                        │   ┌──────────────────────┐  │
                                                        │   ▼                      │  │
                                                        └───collection-name────────┴──┘

►─┬──────────────────────────────┬─┬─────────────────────────────┬──►
  │              ┌─*DB2─┐        │ │          ┌─*NOFLAG─┐        │
  └─SQLCURRULE(──┴─*STD─┴──)─────┘ └─SAAFLAG(─┴─*FLAG───┴──)─────┘

►─┬──────────────────────────┬─┬───────────────────────────────┬──►
  │          ┌─*NONE─┐       │ │          ┌─*NONE───┐          │
  └─FLAGSTD(─┴─*ANS──┴──)────┘ └─DBGVIEW(─┴─*SOURCE─┴──)───────┘

►─┬──────────────────────────┬─┬──────────────────────────────┬──►
  │         ┌─*NAMING─┐      │ │            ┌─*USER──┐        │
  └─USRPRF(─┼─*OWNER──┼──)───┘ └─DYNUSRPRF(─┴─*OWNER─┴──)─────┘
            └─*USER───┘

►─┬───────────────────────────────────────────────┬──►
  │         ┌─*JOB──────────────────────────────┐ │
  └─SRTSEQ(─┼─*JOBRUN───────────────────────────┼─)┘
            ├─*LANGIDUNQ────────────────────────┤
            ├─*LANGIDSHR────────────────────────┤
            ├─*HEX──────────────────────────────┤
            │  ┌─*LIBL/──────────┐              │
            └──┼─────────────────┼──table-name──┘
               ├─*CURLIB/────────┤
               └─library-name/───┘

►─┬──────────────────────────────────────┬──►
  │         ┌─*JOB────────────────────┐  │
  └─LANGID(─┼─*JOBRUN─────────────────┼─)┘
            └─language-identifier─────┘

►─┬───────────────────────────┬──►
  │         ┌─*NONE──┐        │
  └─OUTPUT(─┴─*PRINT─┴──)─────┘

►─┬──────────────────────────────────────────────────────────┬──►
  │          ┌─*LIBL/──────────┐  ┌─QSYSPRT───────────┐      │
  └─PRTFILE(─┼─────────────────┼──┴─printer-file-name─┴──)───┘
             ├─*CURLIB/────────┤
             └─library-name/───┘

►─┬─────────────────────────────────────────────────────────────┬──►
  │            ┌─QTEMP/──────────┐  ┌─*CALC─────────────┐       │
  └─TOSRCFILE(─┼─────────────────┼──┼─QSQLTEMP──────────┼──)────┘
               ├─*LIBL/──────────┤  └─source-file-name──┘
               ├─*CURLIB/────────┤
               └─library-name/───┘

►─┬────────────────────────────────┬──►◄
  │       ┌─*SRCMBRTXT──────┐      │
  └─TEXT(─┼─*BLANK──────────┼──)───┘
          └─'description'───┘
```

OPTION Details:

```
   ┌─*XREF───┐  ┌─*GEN───┐  ┌─*PERIOD─┐  ┌─*SYS─┐  ┌─*NOSECLVL─┐
├──┼─────────┼──┼────────┼──┼─────────┼──┼──────┼──┼───────────┼──────────────►
   └─*NOXREF─┘  └─*NOGEN─┘  ├─*JOB────┤  └─*SQL─┘  └─*SECLVL───┘
                            ├─*SYSVAL─┤
                            └─*COMMA──┘

   ┌─*NOCNULRQD─┐  ┌─*NOEVENTF─┐  ┌─*OPTLOB───┐
►──┼────────────┼──┼───────────┼──┼───────────┼───────────────────────────────┤
   └─*CNULRQD───┘  └─*EVENTF───┘  └─*NOOPTLOB─┘
```

Notes:

1 All parameters preceding this point can be specified in positional form.

Purpose:

The Create Structured Query Language ILE C Object (CRTSQLCI) command calls the Structured Query Language (SQL) precompiler that 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.

Parameters:

OBJ
Specifies the qualified name of the object being created.

The name of the object can be qualified by one of the following library values:

***CURLIB:** The object is created in the current library for the job. If no library is specified as the current library for the job, the QGPL library is used.

library-name: Specify the name of the library where the object is created.

object-name: Specify the name of the object that is being created.

SRCFILE
Specifies the qualified name of the source file that contains the C source with SQL statements.

The name of the source file can be qualified by one of the following library values:

***LIBL:** All libraries in the job's library list are searched until the first match is found.

***CURLIB:** The current library for the job is searched. If no library is specified as the current library for the job, the QGPL library is used.

library-name: Specify the name of the library to be searched.

QCSRC: If the source file name is not specified, the IBM-supplied source file QCSRC contains the C source.

source-file-name: Specify the name of the source file that contains the C source.

SRCMBR
Specifies the name of the source file member that contains the C source. This

parameter is only specified if the source file name in the SRCFILE parameter is a database file. If this parameter is not specified, the OBJ name specified on the OBJ parameter is used.

***OBJ:** Specifies that the C source is in the member of the source file that has the same name as that specified on the OBJ parameter.

source-file-member-name: Specify the name of the member that contains the C source.

OPTION

Specifies whether one or more of the following options are used when the C source is precompiled. If an option is specified more than once, or if two options conflict, the last option specified is used.

Element 1: Cross-Reference Options

***XREF:** The precompiler cross-references items in the program to the statement numbers in the program that refer to those items.

***NOXREF:** The precompiler does not cross-reference names.

Element 2: Program Creation Options

***GEN:** The precompiler creates the object that is specified by the OBJTYPE parameter.

***NOGEN:** The precompiler does not call the C compiler, and a module, program, service program, or SQL package is not created.

Element 3: Decimal Point Options

***PERIOD:** The value used as the decimal point for numeric constants in SQL statements is a period.

***JOB:** The value used as the decimal point for numeric constants in SQL is the representation of decimal point specified for the job at precompile time.

***SYSVAL:** The value used as the decimal point for numeric constants in SQL statements is the QDECFMT system value.

Note: If QDECFMT specifies that the value used as the decimal point is a comma, any numeric constants in lists (such as in the SELECT clause or the VALUES clause) must be separated by a comma followed by a blank. For example, VALUES(1,1, 2,23, 4,1) is equivalent to VALUES(1.1,2.23,4.1) in which the decimal point is a period.

***COMMA:** The value used as the decimal point for numeric constants in SQL statements is a comma.

Note: Any numeric constants in lists (such as in the SELECT clause or the VALUES clause) must be separated by a comma followed by a blank. For example, VALUES(1,1, 2,23, 4,1) is equivalent to VALUES(1.1,2.23,4.1) where the decimal point is a period.

Element 4: Naming Convention Options

***SYS:** The system naming convention (library-name/file-name) is used.

***SQL:** The SQL naming convention is used (schema-name.table-name). When creating a package on a remote database other than an iSeries system, *SQL must be specified as the naming convention.

Element 5: Second-Level Message Text Option

***NOSECLVL:** Second-level text descriptions are not added to the listing.

***SECLVL:** Second-level text with replacement data is added for all messages on the listing.

Element 6: NUL Required Options

***NOCNULRQD:** For output character and graphic host variables, the NUL-terminator is not returned when the host variable is exactly the same length as the data. Input character and graphic host variables do not require a NUL-terminator.

***CNULRQD:** Output character and graphic host variables always contain the NUL-terminator. If there is not enough space for the NUL-terminator, the data is truncated and the NUL-terminator is added. Input character and graphic host variables require a NUL-terminator.

Element 7: Event File Creation

***NOEVENTF:** The compiler will not produce an event file for use by CoOperative Development Environment/400 (CODE/400).

***EVENTF:** The compiler produces an event file for use by CoOperative Development Environment/400 (CODE/400). The event file will be created as a member in the file EVFEVENT in your source library. CODE/400 uses this file to offer error feedback integrated with the CODE/400 editor. This option is normally specified by CODE/400 on your behalf.

Element 8: Large Object Optimization for DRDA

***OPTLOB:** The first FETCH for a cursor derermines how the cursor will be used for LOBs (Large Objects) on all subsequent FETCHes. This option remains in effect until the cursor is closed.

If the first FETCH uses a LOB locator to access a LOB column, no subsequent FETCH for that cursor can fetch that LOB column into a LOB host variable.

If the first FETCH places the LOB column into a LOB host variable, no subsequent FETCH for that cursor can use a LOB locator for that column.

***NOOPTLOB:** There is no restriction on whether a column is retrieved into a LOB locator or into a LOB host variable. This option can cause performance to degrade.

TGTRLS

Specifies the release of the operating system on which the user intends to use the object being created.

In the examples given for the *CURRENT and *PRV values, and when specifying the *release-level* value, the format VxRxMx is used to specify the release, where Vx is the version, Rx is the release, and Mx is the modification level. For example, V2R3M0 is version 2, release 3, modification level 0.

***CURRENT:** The object is to be used on the release of the operating system currently running on the user's system. For example, if V2R3M5 is running on the system, *CURRENT means the user intends to use the object on a system with V2R3M5 installed. The user can also use the object on a system with any subsequent release of the operating system installed.

Note: If V2R3M5 is running on the system, and the object is to be used on a system with V2R3M0 installed, specify TGTRLS(V2R3M0) not TGTRLS(*CURRENT).

***PRV:** The object is to be used on the previous release with modification level 0 of the operating system. For example, if V2R3M5 is running on the user's system, *PRV means the user intends to use the object on a system with V2R2M0 installed. The user can also use the object on a system with any subsequent release of the operating system installed.

release-level: Specify the release in the format VxRxMx. The object can be used on a system with the specified release or with any subsequent release of the operating system installed.

Valid values depend on the current version, release, and modification level, and they change with each new release. If you specify a release-level which is earlier than the earliest release level supported by this command, an error message is sent indicating the earliest supported release.

OBJTYPE

Specifies the type of object being created.

***MODULE:** The SQL precompiler issues the CRTCMOD command to create the module.

***PGM:** The SQL precompiler issues the CRTBNDC command to create the bound program.

***SRVPGM:** The SQL precompiler issues the CRTCMOD and CRTSRVPGM commands to create the service program.

The user must create a source member in QSRVSRC that has the same name as the name specified on the OBJ parameter. The source member must contain the export information for the module. More information about the export file is in the Integrated Language Environment*C/400 Programmers Guide.

Notes:

1. When OBJTYPE(*PGM) or OBJTYPE(*SRVPGM) is specified and the RDB parameter is also specified, the CRTSQLPKG command is issued by the SQL precompiler after the program has been created. When OBJTYPE(*MODULE) is specified, an SQL package is not created and the user must issue the CRTSQLPKG command after the CRTPGM or CRTSRVPGM command has created the program.
2. If *NOGEN is specified, only the SQL temporary source member is generated and a module, program, service program, or SQL package is not created.

INCFILE

Specifies the qualified name of the source file that contains members included in the program with any SQL INCLUDE statement.

The name of the source file can be qualified by one of the following library values:

***LIBL:** All libraries in the job's library list are searched until the first match is found.

***CURLIB:** The current library for the job is searched. If no library is specified as the current library for the job, the QGPL library is used.

library-name: Specify the name of the library to be searched.

***SRCFILE:** The qualified source file specified in the SRCFILE parameter contains the source file members specified on any SQL INCLUDE statement.

source-file-name: Specify the name of the source file that contains the source file members specified on any SQL INCLUDE statement. The record length of the source file specified here must be no less than the record length of the source file specified on the SRCFILE parameter.

COMMIT

Specifies whether SQL statements in the compiled unit are run under commitment control. Files referred to in the host language source are not affected by this option. Only SQL tables, SQL views, and SQL packages referred to in SQL statements are affected.

***CHG or *UR:** Specifies the objects referred to in SQL ALTER, CALL, COMMENT ON, CREATE, DROP, GRANT, LABEL ON, RENAME, and REVOKE statements and the rows updated, deleted, and inserted are locked until the end of the unit of work (transaction). Uncommitted changes in other jobs can be seen.

***ALL or *RS:** Specifies the objects referred to in SQL ALTER, CALL, COMMENT ON, CREATE, DROP, GRANT, LABEL ON, RENAME, and REVOKE statements and the rows selected, updated, deleted, and inserted are locked until the end of the unit of work (transaction). Uncommitted changes in other jobs cannot be seen.

***CS:** Specifies the objects referred to in SQL ALTER, CALL, COMMENT ON, CREATE, DROP, GRANT, LABEL ON, RENAME, and REVOKE statements and the rows updated, deleted, and inserted are locked until the end of the unit of work (transaction). A row that is selected, but not updated, is locked until the next row is selected. Uncommitted changes in other jobs cannot be seen.

***NONE or *NC:** Specifies that commitment control is not used. Uncommitted changes in other jobs can be seen. If the SQL DROP SCHEMA statement is included in the program, *NONE or *NC must be used. If a relational database is specified on the RDB parameter and the relational database is on a system that is not on an iSeries, *NONE or *NC cannot be specified.

***RR:** Specifies the objects referred to in SQL ALTER, CALL, COMMENT ON, CREATE, DROP, GRANT, LABEL ON, RENAME, and REVOKE statements and the rows selected, updated, deleted, and inserted are locked until the end of the unit of work (transaction). Uncommitted changes in other jobs cannot be seen. All tables referred to in SELECT, UPDATE, DELETE, and INSERT statements are locked exclusively until the end of the unit of work (transaction).

CLOSQLCSR

Specifies when SQL cursors are implicitly closed, SQL prepared statements are implicitly discarded, and LOCK TABLE locks are released. SQL cursors are explicitly closed when you issue the CLOSE, COMMIT, or ROLLBACK (without HOLD) SQL statements.

***ENDACTGRP:** SQL cursors are closed, SQL prepared statements are implicitly discarded, and LOCK TABLE locks are released when the activation group ends.

***ENDMOD:** SQL cursors are closed and SQL prepared statements are implicitly discarded when the module is exited. LOCK TABLE locks are released when the first SQL program on the call stack ends.

ALWCPYDTA

Specifies whether a copy of the data can be used in a SELECT statement.

***OPTIMIZE:** The system determines whether to use the data retrieved directly from the database or to use a copy of the data. The decision is based on which method provides the best performance. If COMMIT is *CHG or *CS and ALWBLK is not *ALLREAD, or if COMMIT is *ALL or *RR, then a copy of the data is used only when it is necessary to run a query.

***YES:** A copy of the data is used only when necessary.

***NO:** A copy of the data is not allowed. If a temporary copy of the data is required to perform the query, an error message is returned.

ALWBLK

Specifies whether the database manager can use record blocking, and the extent to which blocking can be used for read-only cursors.

***ALLREAD:** Rows are blocked for read-only cursors if *NONE or *CHG is specified on the COMMIT parameter. All cursors in a program that are not explicitly able to be updated are opened for read-only processing even though EXECUTE or EXECUTE IMMEDIATE statements may be in the program.

Specifying *ALLREAD:

- Allows record blocking under commitment control level *CHG in addition to the blocking allowed for *READ.
- Can improve the performance of almost all read-only cursors in programs, but limits queries in the following ways:
  - The Rollback (ROLLBACK) command, a ROLLBACK statement in host languages, or the ROLLBACK HOLD SQL statement does not reposition a read-only cursor when *ALLREAD is specified.
  - Dynamic running of a positioned UPDATE or DELETE statement (for example, using EXECUTE IMMEDIATE), cannot be used to update a row in a cursor unless the DECLARE statement for the cursor includes the FOR UPDATE clause.

***NONE:** Rows are not blocked for retrieval of data for cursors.

Specifying *NONE:

- Guarantees that the data retrieved is current.
- May reduce the amount of time required to retrieve the first row of data for a query.
- Stops the database manager from retrieving a block of data rows that is not used by the program when only the first few rows of a query are retrieved before the query is closed.
- Can degrade the overall performance of a query that retrieves a large number of rows.

***READ:** Records are blocked for read-only retrieval of data for cursors when:

- *NONE is specified on the COMMIT parameter, which indicates that commitment control is not used.

- The cursor is declared with a FOR READ ONLY clause or there are no dynamic statements that could run a positioned UPDATE or DELETE statement for the cursor.

Specifying *READ can improve the overall performance of queries that meet the above conditions and retrieve a large number of records.

DLYPRP

Specifies whether the dynamic statement validation for a PREPARE statement is delayed until an OPEN, EXECUTE, or DESCRIBE statement is run. Delaying validation improves performance by eliminating redundant validation.

***NO:** Dynamic statement validation is not delayed. When the dynamic statement is prepared, the access plan is validated. When the dynamic statement is used in an OPEN or EXECUTE statement, the access plan is revalidated. Because the authority or the existence of objects referred to by the dynamic statement may change, you must still check the SQLCODE or SQLSTATE after issuing the OPEN or EXECUTE statement to ensure that the dynamic statement is still valid.

***YES:** Dynamic statement validation is delayed until the dynamic statement is used in an OPEN, EXECUTE, or DESCRIBE SQL statement. When the dynamic statement is used, the validation is completed and an access plan is built. If you specify *YES on this parameter, you should check the SQLCODE and SQLSTATE after running an OPEN, EXECUTE, or DESCRIBE statement to ensure that the dynamic statement is valid.

Note: If you specify *YES, performance is not improved if the INTO clause is used on the PREPARE statement or if a DESCRIBE statement uses the dynamic statement before an OPEN is issued for the statement.

GENLVL

Specifies the severity level at which the create operation fails. If errors occur that have a severity level greater than this value, the operation ends.

10: The default severity level is 10.

severity-level: Specify a value ranging from 0 through 40.

MARGINS

Specifies the part of the precompiler input record that contains source text.

***SRCFILE:** The precompiler uses file member margin values that are specified by the user on the SRCMBR parameter.

Element 1: Left Margin

| *left:* Specify the beginning position for the statements. Valid values range from
| 1 through 32754.

Element 2: Right Margin

| *right:* Specify the ending position for the statements. Valid values range from 1
| through 32754.

DATFMT

Specifies the format used when accessing date result columns. All output date fields are returned in the specified format. For input date strings, the specified value is used to determine whether the date is specified in a valid format.

Note: An input date string that uses the format *USA, *ISO, *EUR, or *JIS is always valid.

If a relational database is specified on the RDB parameter and the database is on a system that is not an iSeries system, then *USA, *ISO, *EUR, or *JIS must be specified.

<u>*JOB:</u> The format specified for the job is used. Use the Display Job (DSPJOB) command to determine the current date format for the job.

***USA:** The United States date format (mm/dd/yyyy) is used.

***ISO:** The International Organization for Standardization (ISO) date format (yyyy-mm-dd) is used.

***EUR:** The European date format (dd.mm.yyyy) is used.

***JIS:** The Japanese Industrial Standard date format (yyyy-mm-dd) is used.

***MDY:** The date format (mm/dd/yy) is used.

***DMY:** The date format (dd/mm/yy) is used.

***YMD:** The date format (yy/mm/dd) is used.

***JUL:** The Julian date format (yy/ddd) is used.

DATSEP

Specifies the separator used when accessing date result columns.

Note: This parameter applies only when *JOB, *MDY, *DMY, *YMD, or *JUL is specified on the DATFMT parameter.

<u>*JOB:</u> The date separator specified for the job at precompile time is used. Use the Display Job (DSPJOB) command to determine the current value for the job.

'/': A slash (/) is used.

'.': A period (.) is used.

',': A comma (,) is used.

'-': A dash (-) is used.

' ': A blank ( ) is used.

***BLANK:** A blank ( ) is used.

TIMFMT

Specifies the format used when accessing time result columns. For input time strings, the specified value is used to determine whether the time is specified in a valid format.

Note: An input time string that uses the format *USA, *ISO, *EUR, or *JIS is always valid.

If a relational database is specified on the RDB parameter and the database is on a system that is not another iSeries system, the time format must be *USA, *ISO, *EUR, *JIS, or *HMS with a time separator of colon or period.

***HMS:** The **hh:mm:ss** format is used.

***USA:** The United States time format **hh:mm xx** is used, where **xx** is AM or PM.

***ISO:** The International Organization for Standardization (ISO) time format **hh.mm.ss** is used.

***EUR:** The European time format **hh.mm.ss** is used.

***JIS:** The Japanese Industrial Standard time format **hh:mm:ss** is used.

TIMSEP

Specifies the separator used when accessing time result columns.

Note: This parameter applies only when *HMS is specified on the TIMFMT parameter.

***JOB:** The time separator specified for the job at precompile time is used. Use the Display Job (DSPJOB) command to determine the current value for the job.

':': A colon (:) is used.

'.': A period (.) is used.

',': A comma (,) is used.

' ': A blank ( ) is used.

***BLANK:** A blank ( ) is used.

REPLACE

Specifies if a SQL module, program, service program or package is created when there is an existing SQL module, program, service program, or package of the same name and type in the same library. The value of this parameter is passed to the CRTCMOD, CRTBNDC, CRTSRVPGM, and CRTSQLPKG commands.

***YES:** A new SQL module, program, service program, or package is created, and any existing object of the same name and type in the specified library is moved to QRPLOBJ.

***NO:** A new SQL module, program, service program, or package is not created if an object of the same name and type already exists in the specified library.

RDB

Specifies the name of the relational database where the SQL package object is created.

***LOCAL:** The program is created as a distributed SQL program. The SQL statements will access the local database. An SQL package object is not created as part of the precompile process. The Create Structured Query Language Package (CRTSQLPKG) command can be used.

relational-database-name: Specify the name of the relational database where the new SQL package object is to be created. When the name of the local relational database is specified, the program created is still a distributed SQL program. The SQL statements will access the local database.

***NONE:** An SQL package object is not created. The program object is not a distributed program and the Create Structured Query Language Package (CRTSQLPKG) command cannot be used.

USER
Specifies the user name sent to the remote system when starting the conversation. This parameter is valid only when RDB is specified.

***CURRENT:** The user profile under which the current job is running is used.

user-name: Specify the user name being used for the application server job.

PASSWORD
Specifies the password to be used on the remote system. This parameter is valid only if RDB is specified.

***NONE:** No password is sent. If this value is specified, USER(*CURRENT) must also be specified.

password: Specify the password of the user name specified on the USER parameter.

RDBCNNMTH
Specifies the semantics used for CONNECT statements. Refer to the SQL Reference book for more information.

***DUW:** CONNECT (Type 2) semantics are used to support distributed unit of work. Consecutive CONNECT statements to additional relational databases do not result in disconnection of previous connections.

***RUW:** CONNECT (Type 1) semantics are used to support remote unit of work. Consecutive CONNECT statements result in the previous connection being disconnected before a new connection is established.

DFTRDBCOL
Specifies the schema name used for the unqualified names of tables, views, indexes, and SQL packages. This parameter applies only to static SQL statements.

***NONE:** The naming convention defined on the OPTION parameter is used.

schema-name: Specify the name of the schema identifier. This value is used instead of the naming convention specified on the OPTION parameter.

DYNDFTCOL
Specifies whether the default schema name specified for the DFTRDBCOL parameter is also used for dynamic statements.

***NO:** Do not use the value specified on the DFTRDBCOL parameter for unqualified names of tables, views, indexes, and SQL packages for dynamic SQL statements. The naming convention specified on the OPTION parameter is used.

***YES:** The schema name specified on the DFTRDBCOL parameter will be used for the unqualified names of the tables, views, indexes, and SQL packages in dynamic SQL statements.

SQLPKG
Specifies the qualified name of the SQL package created on the relational database specified on the RDB parameter of this command.

The possible library values are:

***OBJLIB:** The package is created in the library with the same name as the library specified on the OBJ parameter.

library-name: Specify the name of the library where the package is created.

***OBJ:** The name of the SQL package is the same as the object name specified on the OBJ parameter.

package-name: Specify the name of the SQL package. If the remote system is not an iSeries system, no more than 8 characters can be specified.

SQLPATH
Specifies the path to be used to find procedures, functions, and user defined types in static SQL statements.

***NAMING:** The path used depends on the naming convention specified on the OPTION parameter.

For *SYS naming, the path used is *LIBL, the current library list at runtime.

For *SQL naming, the path used is "QSYS", "QSYS2", "userid", where "userid" is the value of the USER special register. If a schema-name is specified on the DFTRDBCOL parameter, the schema-name takes the place of userid.

***LIBL:** The path used is the library list at runtime.

schema-name: Specify a list of one or more schema names. A maximum of 268 individual schemas may be specified.

SQLCURRULE
Specifies the semantics used for SQL statements.

***DB2:** The semantics of all SQL statements will default to the rules established for DB2. The following semantics are controlled by this option:

- Hexadecimal constants are treated as character data.

***STD:** The semantics of all SQL statements will default to the rules established by the ISO and ANSI SQL standards. The following semantics are controlled by this option:

- Hexadecimal constants are treated as binary data.

SAAFLAG
Specifies the IBM SQL flagging function. This parameter flags SQL statements to verify whether they conform to IBM SQL syntax. More information about which IBM database products IBM SQL syntax is in the *DRDA IBM SQL Reference*, SC26-3255-00.

***NOFLAG:** The precompiler does not check to see whether SQL statements conform to IBM SQL syntax.

***FLAG:** The precompiler checks to see whether SQL statements conform to IBM SQL syntax

FLAGSTD
Specifies the American National Standards Institute (ANSI) flagging function. This parameter flags SQL statements to verify whether they conform to the following standards.

```
ANSI X3.135-1992 entry
ISO 9075-1992 entry
FIPS 127.2 entry
```

***NONE:** The precompiler does not check to see whether SQL statements conform to ANSI standards.

***ANS:** The precompiler checks to see whether SQL statements conform to ANSI standards.

DBGVIEW
This parameter specifies the type of source debug information to be provided by the SQL precompiler.

***NONE:** The source view will not be generated.

***SOURCE:** The SQL precompiler provides the source views for the root and if necessary, SQL INCLUDE statements. A view is provided that contains the statements generated by the precompiler.

USRPRF
Specifies the user profile that is used when the compiled program object is run, including the authority that the program object has for each object in static SQL statements. The profile of either the program owner or the program user is used to control which objects can be used by the program object.

***NAMING:** The user profile is determined by the naming convention. If the naming convention is *SQL, USRPRF(*OWNER) is used. If the naming convention is *SYS, USRPRF(*USER) is used.

***USER:** The profile of the user running the program object is used.

***OWNER:** The user profiles of both the program owner and the program user are used when the program is run.

DYNUSRPRF
Specifies the user profile to be used for dynamic SQL statements.

***USER:** Local dynamic SQL statements are run under the profile of the program's user. Distributed dynamic SQL statements are run under the profile of the SQL package's user.

***OWNER:** Local dynamic SQL statements are run under the profile of the program's owner. Distributed dynamic SQL statements are run under the profile of the SQL package's owner.

SRTSEQ
Specifies the sort sequence table to be used for string comparisons in SQL statements.

Note: *HEX must be specified for this parameter on distributed applications where the application server is not on an iSeries system or the release level is prior to V2R3M0.

***JOB:** The SRTSEQ value for the job is retrieved during the precompile.

***JOBRUN:** The LANGID value for the job is retrieved when the program is run. For distributed applications, LANGID(*JOBRUN) is valid only when SRTSEQ(*JOBRUN) is also specified.

***HEX:** A sort sequence table is not used. The hexadecimal values of the characters are used to determine the sort sequence.

***LANGIDSHR:** The sort sequence table uses the same weight for multiple characters, and is the shared-weight sort sequence table associated with the language specified on the LANGID parameter.

***LANGIDUNQ:** The unique-weight sort table for the language specified on the LANGID parameter is used.

The name of the table name can be qualified by one of the following library values:

***LIBL:** All libraries in the job's library list are searched until the first match is found.

***CURLIB:** The current library for the job is searched. If no library is specified as the current library for the job, the QGPL library is used.

library-name: Specify the name of hte library to be searched.

table-name: Specify the name of the sort sequence table to be used.

LANGID

Specifies the language identifier to be used when SRTSEQ(*LANGIDUNQ) or SRTSEQ(*LANGIDSHR) is specified.

***JOB:** The LANGID value for the job is retrieved during the precompile.

***JOBRUN:** The LANGID value for the job is retrieved when the program is run. For distributed applications, LANGID(*JOBRUN) is valid only when SRTSEQ(*JOBRUN) is also specified.

language-identifier: Specify a language identifier.

OUTPUT

Specifies whether the precompiler listing is generated.

***NONE:** The precompiler listing is not generated.

***PRINT:** The precompiler listing is generated.

PRTFILE

Specifies the qualified name of the printer device file to which the precompiler printout is directed. The file must have a minimum length of 132 bytes. If a file with a record length of less than 132 bytes is specified, information is lost.

The name of the printer file can be qualified by one of the following library values:

***LIBL:** All libraries in the job's library list are searched until the first match is found.

***CURLIB:** The current library for the job is searched. If no library is specified as the current library for the job, the QGPL library is used.

library-name: Specify the name of the library to be searched.

QSYSPRT: If a file name is not specified, the precompiler printout is directed to the IBM-supplied printer file QSYSPRT.

printer-file-name: Specify the name of the printer device file to which the precompiler printout is directed.

TOSRCFILE

Specifies the qualified name of the source file that is to contain the output source member that the SQL precompiler has processed. If the precompiler cannot find the specified source file, it creates the file. The output member will have the same name as the name that is specified for the SRCMBR parameter.

The possible library values are:

QTEMP: The library QTEMP will be used.

***LIBL:** The job's library list is searched for the specified file. If the file is not found in any library in the library list, the file will be created in the current library.

***CURLIB:** The current library for the job will be used. If no library is specified as the current library for the job, the QGPL library will be used.

library-name: Specify the name of the library that is to contain the output source file.

***CALC:** The output source file name will be generated based on the margins of the source file. The name will be QSQLTxxxxx, where xxxxx is the width of the source file. If the source file record length is less than or equal to 92, the name will be QSQLTEMP.

QSQLTEMP: The source file QSQLTEMP will be used.

source-file-name: Specify the name of the source file to contain the output source member.

TEXT

Specifies the text that briefly describes the program and the function. more information about this parameter is in the TEXT parameter topic in the CL Reference section of the Information Center.

***SRCMBRTXT:** The text is taken from the source file member being used to create the C program. Text can be added or changed for a database source member by using the Start Source Entry Utility (STRSEU) command, or by using either the Add Physical File Member (ADDPFM) command or the Change Physical File Member (CHGPFM) command. If the source file is an inline file or a device file, the text is blank.

***BLANK:** Text is not specified.

'description': Specify no more than 50 characters of text, enclosed in apostrophes.

Example:

```
CRTSQLCI  PAYROLL  OBJTYPE(*MODULE)
  TEXT('Payroll Program')
```

This command runs the SQL precompiler which precompiles the source and stores the changed source in member PAYROLL in file QSQLTEMP in library QTEMP. The ILE C for iSeries compiler is called to create module PAYROLL in the current library by using the source member created by the SQL precompiler.

CRTSQLCPPI (Create Structured Query Language C++ Object) Command

Job: B,I   Pgm: B,I   REXX: B,I   Exec

```
                         ┌─*CURLIB/──────┐
►►──CRTSQLCPPI──OBJ(─────┼───────────────┼──object-name──)────────────────►
                         └─library-name/─┘
```

CRTSQLCPPI

```
►─┬──────────────────────────────────────────────────────────────────┬──────────►
  │            ┌─*LIBL/───────────┐ ┌─QCSRC─────────────┐              │
  └─SRCFILE(───┼──────────────────┼─┴─source-file-name──┴──)───────────┘
               ├─*CURLIB/─────────┤
               └─library-name/────┘

                                           (1)
►─┬──────────────────────────────────────┬────────────────────────────────────►
  │           ┌─*OBJ─────────────────────┐ │
  └─SRCMBR(───┴─source-file-member-name──┴──)─┘

►─┬──────────────────────────────┬─┬─────────────────────────────┬────────────►
  └─OPTION(─┤ OPTION Details ├─)─┘ │           ┌─*CURRENT─┐      │
                                   └─TGTRLS(───┴─VxRxMx───┴──)───┘

►─┬──────────────────────────────────────────────────────────────────┬──────────►
  │            ┌─*LIBL/───────────┐ ┌─*SRCFILE──────────┐            │
  └─INCFILE(───┼──────────────────┼─┴─source-file-name──┴──)─────────┘
               ├─*CURLIB/─────────┤
               └─library-name/────┘

►─┬────────────────────────┬─┬──────────────────────────────────────┬─────────►
  │           ┌─*UR────┐   │ │              ┌─*ENDACTGRP─┐          │
  │           ├─*CHG───┤   │ └─CLOSQLCSR(───┴─*ENDMOD────┴──)───────┘
  └─COMMIT(───┼─*ALL───┼─)─┘
              ├─*RS────┤
              ├─*CS────┤
              ├─*NONE──┤
              ├─*NC────┤
              └─*RR────┘

►─┬───────────────────────────────────┬─┬──────────────────────────────┬───────►
  │              ┌─*OPTIMIZE─┐        │ │          ┌─*ALLREAD─┐        │
  └─ALWCPYDTA(───┼─*YES──────┼──)─────┘ └─ALWBLK(──┼─*NONE────┼──)─────┘
                 └─*NO───────┘                     └─*READ────┘

►─┬───────────────────────┬─┬─────────────────────────────────┬───────────────►
  │          ┌─*NO──┐     │ │          ┌─10─────────────┐     │
  └─DLYPRP(──┴─*YES─┴──)──┘ └─GENLVL(──┴─severity-level─┴──)──┘

►─┬───────────────────────────────┬─┬──────────────────────┬──────────────────►
  │            ┌─*SRCFILE───┐     │ │          ┌─*JOB─┐    │
  └─MARGINS(───┴─left-right─┴──)──┘ └─DATFMT(──┼─*USA─┼─)──┘
                                               ├─*ISO─┤
                                               ├─*EUR─┤
                                               ├─*JIS─┤
                                               ├─*MDY─┤
                                               ├─*DMY─┤
                                               ├─*YMD─┤
                                               └─*JUL─┘
```

```
►─┬─────────────────────────────┬─┬─────────────────────────────┬────────────►
  │            ┌─*JOB───┐       │ │            ┌─*HMS─┐        │
  └─DATSEP(────┼─'/'────┼────)──┘ └─TIMFMT(────┼─*USA─┼──)─────┘
               ├─'.'────┤                      ├─*ISO─┤
               ├─','────┤                      ├─*EUR─┤
               ├─'-'────┤                      └─*JIS─┘
               ├─' '────┤
               └─*BLANK─┘

►─┬─────────────────────────────┬─────────────────────────────────────────────►
  │            ┌─*JOB───┐       │
  └─TIMSEP(────┼─':'────┼────)──┘
               ├─'.'────┤
               ├─','────┤
               ├─' '────┤
               └─*BLANK─┘

►─┬─────────────────────────────┬─┬─────────────────────────────────────────┬─►
  │             ┌─*YES─┐        │ │       ┌─*LOCAL─────────────────────┐    │
  └─REPLACE(────┴─*NO──┴─────)──┘ └─RDB(──┼─relational-database-name───┼──)─┘
                                          └─*NONE──────────────────────┘

►─┬──────────────────────────────┬─┬─────────────────────────────────┬────────►
  │        ┌─*CURRENT──┐         │ │            ┌─*NONE─────┐        │
  └─USER(──┴─user-name─┴──────)──┘ └─PASSWORD(──┴─password──┴─────)──┘

►─┬──────────────────────────────┬─┬─────────────────────────────────────┬────►
  │              ┌─*DUW─┐        │ │               ┌─*NONE───────────┐   │
  └─RDBCNNMTH(───┴─*RUW─┴─────)──┘ └─DFTRDBCOL(────┴─collection-name─┴─)─┘

►─┬──────────────────────────────┬────────────────────────────────────────────►
  │              ┌─*NO──┐        │
  └─DYNDFTCOL(───┴─*YES─┴─────)──┘

►─┬───────────────────────────────────────────────────────┬───────────────────►
  │          ┌─*OBJLIB/──────┐      ┌─*OBJ─────────┐      │
  └─SQLPKG(──┼───────────────┼──────┴─package-name─┴──)───┘
             └─library-name/─┘

►─┬───────────────────────────────────────────┬─┬──────────────────────────┬──►
  │           ┌─*NAMING─────────────┐         │ │              ┌─*DB2─┐    │
  └─SQLPATH(──┼─*LIBL───────────────┼──────)──┘ └─SQLCURRULE(──┴─*STD─┴──)─┘
              │  ┌────────────────┐ │
              │  ▼                │ │
              └───collection-name─┴─┘

►─┬────────────────────────────────┬─┬───────────────────────────┬────────────►
  │            ┌─*NOFLAG─┐         │ │            ┌─*NONE─┐      │
  └─SAAFLAG(───┴─*FLAG───┴──────)──┘ └─FLAGSTD(───┴─*ANS──┴───)──┘

►─┬────────────────────────────────┬─┬──────────────────────────────┬─────────►
  │            ┌─*NONE───┐         │ │           ┌─*NAMING─┐        │
  └─DBGVIEW(───┴─*SOURCE─┴──────)──┘ └─USRPRF(───┼─*OWNER──┼────)───┘
                                                 └─*USER───┘
```

CRTSQLCPPI

```
►─┬──────────────────────────────────┬──────────────────────────────────────────►
  │               ┌─*USER──┐         │
  └─DYNUSRPRF(────┴─*OWNER─┴──)──────┘

►─┬──────────────────────────────────────────────────┬──────────────────────────►
  │          ┌─*JOB─────────────────────────┐        │
  └─SRTSEQ(──┼─*JOBRUN──────────────────────┼──)─────┘
             ├─*LANGIDUNQ───────────────────┤
             ├─*LANGIDSHR───────────────────┤
             ├─*HEX─────────────────────────┤
             │  ┌─*LIBL/────────┐           │
             └──┼───────────────┼─table-name┘
                ├─*CURLIB/──────┤
                └─library-name/─┘

►─┬─────────────────────────────────────────────┬─┬────────────────────────────┬►
  │          ┌─*JOB─────────────────┐           │ │           ┌─*NONE──┐       │
  └─LANGID(──┼─*JOBRUN──────────────┼──)────────┘ └─OUTPUT(───┴─*PRINT─┴──)───┘
             └─language-identifier──┘

►─┬──────────────────────────────────────────────────────────────┬──────────────►
  │              ┌─*LIBL/────────┐   ┌─QSYSPRT────────────┐      │
  └─PRTFILE(─────┼───────────────┼───┴─printer-file-name──┴──)───┘
                 ├─*CURLIB/──────┤
                 └─library-name/─┘

►─┬──────────────────────────────────────────────────────────────┬──────────────►
  │               ┌─QTEMP/────────┐   ┌─*CALC─────────────┐      │
  └─TOSRCFILE(────┼───────────────┼───┼─QSQLTEMP──────────┼──)───┘
                  ├─*LIBL/────────┤   └─source-file-name──┘
                  ├─*CURLIB/──────┤
                  └─library-name/─┘

►─┬───────────────────────────────────┬────────────────────────────────────────►◄
  │        ┌─*SRCMBRTXT──────┐        │
  └─TEXT(──┼─*BLANK──────────┼──)─────┘
           └─'description'───┘
```

OPTION Details:

```
   ┌─*XREF────┐  ┌─*GEN────┐  ┌─*JOB─────┐  ┌─*SYS─┐  ┌─*NOSECLVL─┐
├──┴─*NOXREF──┴──┴─*NOGEN──┴──┼─*PERIOD──┼──┴─*SQL─┴──┴─*SECLVL───┴──────────►
                              ├─*SYSVAL──┤
                              └─*COMMA───┘

   ┌─*NOCNULRQD─┐  ┌─*NOEVENTF─┐  ┌─*OPTLOB───┐
►──┴─*CNULRQD───┴──┴─*EVENTF───┴──┴─*NOOPTLOB─┴──────────────────────────────┤
```

Notes:

1 All parameters preceding this point can be specified in positional form.

Purpose:

The Create Structured Query Language C++ Object (CRTSQLCPPI) command calls the Structured Query Language (SQL) precompiler. The SQL precompiler precompiles C++ source containing SQL statements, produces a temporary source member, and then optionally calls the C++ compiler to create a module.

Parameters:

OBJ

Specifies the qualified name of the object that the precompiler creates.

One of the following library values can qualify the name of the object:

***CURLIB** The object is created in the current library for the job. If you do not specify a library as the current library for the job, the precompiler uses QGPL library.

library-name: Specify the name of the library where the object is created.

object-name: Specify the name of the object that the precompiler creates.

SRCFILE

Specifies the qualified name of the source file that contains the C++ source with SQL statements.

One of the following library values can qualify the name of the source file:

***LIBL:** The precompiler searches all libraries in the job's library list until it finds the first match.

***CURLIB:** The precompiler searches the current library for the job. If you do not specify a library as the current library for the job, it uses the QGPL library.

library-name: Specify the name of the library that the precompiler searches.

QCSRC: If you do not specify the source file name, the IBM-supplied source file QCSRC contains the C++ source.

source-file-name: Specify the name of the source file that contains the C++ source.

SRCMBR

Specifies the name of the source file member that contains the C++ source. Specify this parameter only if the source file name in the SRCFILE parameter is a database file. If you do not specify this parameter, the precompiler uses the OBJ name that is specified on the OBJ parameter.

***OBJ:** Specifies that the C++ source is in the member of the source file that has the same name as the file specified on the OBJ parameter.

source-file-member-name: Specify the name of the member that contains the C++ source.

OPTION

Specifies whether one or more of the following options are used when the C++ source is precompiled. If an option is specified more than once, or if two options conflict, the last option specified is used.

Element 1: Cross-Reference Options

***XREF:** The precompiler cross-references items in the program to the statement numbers in the program that refer to those items.

***NOXREF:** The precompiler does not cross-reference names.

Element 2: Program Creation Options

***GEN:** The precompiler creates the module object.

***NOGEN:** The precompiler does not call the C++ compiler, and does not create a module.

Element 3: Decimal Point Options

***JOB:** The value used as the decimal point for numeric constants in SQL is the representation of decimal point that is specified for the job at precompile time.

Note: If the job specifies that the value used as the decimal point is a comma, any numeric constants in lists (such as in the SELECT clause or the VALUES clause) must be separated by a comma followed by a blank. For example, VALUES(1,1, 2,23, 4,1) is equivalent to VALUES(1.1,2.23,4.1) in which the decimal point is a period.

***PERIOD:** The value used as the decimal point for numeric constants in SQL statements is a period.

***COMMA:** The value used as the decimal point for numeric constants in SQL statements is a comma.

Note: Any numeric constants in lists (such as in the SELECT clause or the VALUES clause) must be separated by a comma followed by a blank. For example, VALUES(1,1, 2,23, 4,1) is equivalent to VALUES(1.1,2.23,4.1) where the decimal point is a period.

Element 4: Naming Convention Options

***SYS:** The system naming convention (library-name/file-name) is used.

***SQL:** The SQL naming convention is used (schema-name.table-name). When creating a package on a remote database other than an iSeries system, you must specify *SQL as the naming convention.

Element 5: Second-Level Message Text Option

***NOSECLVL:** Second-level text descriptions are not added to the listing.

***SECLVL:** Second-level text with replacement data is added for all messages on the listing.

Element 6: NUL Required Options

***NOCNULRQD:** For output character and graphic host variables, the NUL-terminator is not returned when the host variable is exactly the same length as the data. Input character and graphic host variables do not require a NUL-terminator.

***CNULRQD:** Output character and graphic host variables always contain the NUL-terminator. If there is not enough space for the NUL-terminator, the data is truncated, and the NUL-terminator is added. Input character and graphic host variables require a NUL-terminator.

Element 7: Event File Creation

***NOEVENTF:** The compiler will not produce an event file for use by CoOperative Development Environment/400 (CODE/400).

***EVENTF:** The compiler produces an event file for use by CoOperative Development Environment/400 (CODE/400). It creates the event file as a member in the file EVFEVENT in your source library. CODE/400 uses this file to offer error feedback that is integrated with the CODE/400 editor. CODE/400 normally specifies this option on your behalf.

Element 8: Large Object Optimization for DRDA

***OPTLOB:** The first FETCH for a cursor derermines how the cursor will be used for LOBs (Large Objects) on all subsequent FETCHes. This option remains in effect until the cursor is closed.

If the first FETCH uses a LOB locator to access a LOB column, no subsequent FETCH for that cursor can fetch that LOB column into a LOB host variable.

If the first FETCH places the LOB column into a LOB host variable, no subsequent FETCH for that cursor can use a LOB locator for that column.

***NOOPTLOB:** There is no restriction on whether a column is retrieved into a LOB locator or into a LOB host variable. This option can cause performance to degrade.

TGTRLS

Specifies the release of the operating system on which the user intends to use the object that is being created.

The examples given for the *CURRENT value, as well as the *release-level* value, use the format VxRxMx to specify the release. In this format, Vx is the version, Rx is the release, and Mx is the modification level. For example, V2R3M0 is version 2, release 3, modification level 0.

***CURRENT:** The object is to be used on the release of the operating system that is currently running on the user's system. For example, if V2R3M5 is running on the system, *CURRENT means that the user intends to use the object on a system with V2R3M5 installed. The user can also use the object on a system with any subsequent release of the operating system installed.

Note: If V2R3M5 is running on the system, and the object is to be used on a system with V2R3M0 installed, specify TGTRLS(V2R3M0) not TGTRLS(*CURRENT).

release-level: Specify the release in the format VxRxMx. The object can be used on a system with the specified release or with any subsequent release of the operating system installed.

Valid values depend on the current version, release, and modification level, and they change with each new release. If you specify a release-level which is earlier than the earliest release level that is supported by this command, an error message is sent indicating the earliest supported release.

INCFILE

Specifies the qualified name of the source file that contains members that are included in the program with any SQL INCLUDE statement.

One of the following library values can qualify the name of the source file:

***LIBL:** All libraries in the job's library list are searched until the first match is found.

***CURLIB:** The current library for the job is searched. If no library is specified as the current library for the job, the QGPL library is used.

library-name: Specify the name of the library to be searched.

***SRCFILE:** The qualified source file specified in the SRCFILE parameter contains the source file members that are specified on any SQL INCLUDE statement.

source-file-name: Specify the name of the source file that contains the source file members that are specified on any SQL INCLUDE statement. The record length of the source file that is specified here must be no less than the record length of the source file specified on the SRCFILE parameter.

COMMIT

Specifies whether SQL statements in the compiled unit are run under commitment control. Files referred to in the host language source are not affected by this option. Only SQL tables, SQL views, and SQL packages referred to in SQL statements are affected.

***CHG or *UR:** Specifies the objects referred to in SQL ALTER, CALL, COMMENT ON, CREATE, DROP, GRANT, LABEL ON, RENAME, and REVOKE statements and the rows updated, deleted, and inserted are locked until the end of the unit of work (transaction). Uncommitted changes in other jobs can be seen.

***ALL or *RS:** Specifies the objects referred to in SQL ALTER, CALL, COMMENT ON, CREATE, DROP, GRANT, LABEL ON, RENAME, and REVOKE statements and the rows selected, updated, deleted, and inserted are locked until the end of the unit of work (transaction). Uncommitted changes in other jobs cannot be seen.

***CS:** Specifies the objects referred to in SQL ALTER, CALL, COMMENT ON, CREATE, DROP, GRANT, LABEL ON, RENAME, and REVOKE statements and the rows updated, deleted, and inserted are locked until the end of the unit of work (transaction). A row that is selected, but not updated, is locked until the next row is selected. Uncommitted changes in other jobs cannot be seen.

***NONE or *NC:** Specifies that commitment control is not used. Uncommitted changes in other jobs can be seen. If the SQL DROP SCHEMA statement is included in the program, *NONE or *NC must be used. If a relational database is specified on the RDB parameter and the relational database is on a system that is not on an iSeries, *NONE or *NC cannot be specified.

***RR:** Specifies the objects referred to in SQL ALTER, CALL, COMMENT ON, CREATE, DROP, GRANT, LABEL ON, RENAME, and REVOKE statements and the rows selected, updated, deleted, and inserted are locked until the end of the unit of work (transaction). Uncommitted changes in other jobs cannot be seen. All tables referred to in SELECT, UPDATE, DELETE, and INSERT statements are locked exclusively until the end of the unit of work (transaction).

CLOSQLCSR

Specifies when SQL cursors are implicitly closed, SQL prepared statements are implicitly discarded, and LOCK TABLE locks are released. SQL cursors are explicitly closed when you issue the CLOSE, COMMIT, or ROLLBACK (without HOLD) SQL statements.

***ENDACTGRP:** SQL cursors are closed, SQL prepared statements are implicitly discarded, and LOCK TABLE locks are released when the activation group ends.

***ENDMOD:** SQL cursors are closed, and SQL prepared statements are implicitly discarded when the module is exited. LOCK TABLE locks are released when the first SQL program on the call stack ends.

ALWCPYDTA

Specifies whether a copy of the data can be used in a SELECT statement.

***OPTIMIZE:** The system determines whether to use the data retrieved directly from the database or to use a copy of the data. The decision is based on which method provides the best performance. If COMMIT is *CHG or *CS and ALWBLK is not *ALLREAD, or if COMMIT is *ALL or *RR, then a copy of the data is used only when it is necessary to run a query.

***YES:** A copy of the data is used only when necessary.

***NO:** A copy of the data is not allowed. If a temporary copy of the data is required to perform the query, an error message is returned.

ALWBLK

Specifies whether the database manager can use record blocking, and the extent to which blocking can be used for read-only cursors.

***ALLREAD:** Rows are blocked for read-only cursors if *NONE or *CHG is specified on the COMMIT parameter. All cursors in a program that are not explicitly able to be updated are opened for read-only processing even though EXECUTE or EXECUTE IMMEDIATE statements may be in the program.

Specifying *ALLREAD:

- Allows record blocking under commitment control level *CHG in addition to the blocking allowed for *READ.
- Can improve the performance of almost all read-only cursors in programs, but limits queries in the following ways:
  - The Rollback (ROLLBACK) command, a ROLLBACK statement in host languages, or the ROLLBACK HOLD SQL statement does not reposition a read-only cursor when *ALLREAD is specified.
  - Dynamic running of a positioned UPDATE or DELETE statement (for example, using EXECUTE IMMEDIATE), cannot be used to update a row in a cursor unless the DECLARE statement for the cursor includes the FOR UPDATE clause.

***NONE:** Rows are not blocked for retrieval of data for cursors.

Specifying *NONE:

- Guarantees that the data retrieved is current.
- May reduce the amount of time required to retrieve the first row of data for a query.
- Stops the database manager from retrieving a block of data rows that is not used by the program when only the first few rows of a query are retrieved before the query is closed.
- Can degrade the overall performance of a query that retrieves a large number of rows.

***READ:** Records are blocked for read-only retrieval of data for cursors when:

- *NONE is specified on the COMMIT parameter, which indicates that commitment control is not used.

- The cursor is declared with a FOR READ ONLY clause or there are no dynamic statements that could run a positioned UPDATE or DELETE statement for the cursor.

Specifying *READ can improve the overall performance of queries that meet the above conditions and retrieve a large number of records.

DLYPRP
Specifies whether the dynamic statement validation for a PREPARE statement is delayed until an OPEN, EXECUTE, or DESCRIBE statement is run. Delaying validation improves performance by eliminating redundant validation.

***NO:** Dynamic statement validation is not delayed. When the dynamic statement is prepared, the access plan is validated. When the dynamic statement is used in an OPEN or EXECUTE statement, the access plan is revalidated. Because the authority or the existence of objects referred to by the dynamic statement may change, you must still check the SQLCODE or SQLSTATE after issuing the OPEN or EXECUTE statement to ensure that the dynamic statement is still valid.

***YES:** Dynamic statement validation is delayed until the dynamic statement is used in an OPEN, EXECUTE, or DESCRIBE SQL statement. When the dynamic statement is used, the validation is completed, and an access plan is built. If you specify *YES on this parameter, you should check the SQLCODE and SQLSTATE after running an OPEN, EXECUTE, or DESCRIBE statement to ensure that the dynamic statement is valid.

Note: If you specify *YES, performance is not improved if the INTO clause is used on the PREPARE statement or if a DESCRIBE statement uses the dynamic statement before an OPEN is issued for the statement.

GENLVL
Specifies the severity level at which the create operation fails. If errors occur that have a severity level greater than this value, the operation ends.

10: The default severity level is 10.

severity-level: Specify a value ranging from 0 through 40.

MARGINS
Specifies the part of the precompiler input record that contains source text.

***SRCFILE:** The file member margin values specified by the user on the SRCMBR parameter are used.

Element 1: Left Margin

left: Specify the beginning position for the statements. Valid values range from 1 through 32754.

Element 2: Right Margin

right: Specify the ending position for the statements. Valid values range from 1 through 32754.

DATFMT
Specifies the format used when accessing date result columns. All output date fields are returned in the specified format. For input date strings, the specified value is used to determine whether the date is specified in a valid format.

Note: An input date string that uses the format *USA, *ISO, *EUR, or *JIS is always valid.

If a relational database is specified on the RDB parameter and the database is on a system that is not an iSeries system, then *USA, *ISO, *EUR, or *JIS must be specified.

***JOB:** The format specified for the job is used. Use the Display Job (DSPJOB) command to determine the current date format for the job.

***USA:** The United States date format (mm/dd/yyyy) is used.

***ISO:** The International Organization for Standardization (ISO) date format (yyyy-mm-dd) is used.

***EUR:** The European date format (dd.mm.yyyy) is used.

***JIS:** The Japanese Industrial Standard date format (yyyy-mm-dd) is used.

***MDY:** The date format (mm/dd/yy) is used.

***DMY:** The date format (dd/mm/yy) is used.

***YMD:** The date format (yy/mm/dd) is used.

***JUL:** The Julian date format (yy/ddd) is used.

DATSEP

Specifies the separator used when accessing date result columns.

Note: This parameter applies only when *JOB, *MDY, *DMY, *YMD, or *JUL is specified on the DATFMT parameter.

***JOB:** The date separator specified for the job at precompile time is used. Use the Display Job (DSPJOB) command to determine the current value for the job.

'/': A slash (/) is used.

'.': A period (.) is used.

',': A comma (,) is used.

'-': A dash (-) is used.

' ': A blank ( ) is used.

***BLANK:** A blank ( ) is used.

TIMFMT

Specifies the format used when accessing time result columns. For input time strings, the specified value is used to determine whether the time is specified in a valid format.

Note: An input time string that uses the format *USA, *ISO, *EUR, or *JIS is always valid.

If a relational database is specified on the RDB parameter and the database is on a system that is not another iSeries system, the time format must be *USA, *ISO, *EUR, *JIS, or *HMS with a time separator of colon or period.

***HMS:** The **hh:mm:ss** format is used.

***USA:** The United States time format **hh:mm xx** is used, where **xx** is AM or PM.

***ISO:** The International Organization for Standardization (ISO) time format **hh.mm.ss** is used.

***EUR:** The European time format **hh.mm.ss** is used.

***JIS:** The Japanese Industrial Standard time format **hh:mm:ss** is used.

TIMSEP
Specifies the separator used when accessing time result columns.

Note: This parameter applies only when *HMS is specified on the TIMFMT parameter.

***JOB:** The time separator specified for the job at precompile time is used. Use the Display Job (DSPJOB) command to determine the current value for the job.

':': A colon (:) is used.

'.': A period (.) is used.

',': A comma (,) is used.

' ': A blank ( ) is used.

***BLANK:** A blank ( ) is used.

REPLACE
Specifies if an SQL module is created when there is an existing SQL module of the same name in the same library. The value of this parameter is passed to the CRTCPPMOD command.

***YES:** A new SQL module is created, and any existing object of the same name in the specified library is moved to QRPLOBJ.

***NO:** A new SQL module is not created if an object of the same name already exists in the specified library.

RDB
Specifies the name of the relational database where the SQL package object is created.

***LOCAL:** The program is created as a distributed SQL program. The SQL statements will access the local database. An SQL package object is not created as part of the precompile process. The Create Structured Query Language Package (CRTSQLPKG) command can be used.

relational-database-name: Specify the name of the relational database where the new SQL package object is to be created. When the name of the local relational database is specified, the program created is still a distributed SQL program. The SQL statements will access the local database.

***NONE:** An SQL package object is not created. The program object is not a distributed program and the Create Structured Query Language Package (CRTSQLPKG) command cannot be used.

USER
Specifies the user name sent to the remote system when starting the conversation. This parameter is valid only when RDB is specified.

***CURRENT:** The user profile under which the current job is running is used.

user-name: Specify the user name being used for the application server job.

PASSWORD
Specifies the password to be used on the remote system. This parameter is valid only if RDB is specified.

***NONE:** No password is sent. If this value is specified, USER(*CURRENT) must also be specified.

password: Specify the password of the user name that is specified on the USER parameter.

RDBCNNMTH
Specifies the name of the relational database where the SQL package object is created.

***DUW:** CONNECT (Type 2) semantics are used to support distributed unit of work. Consecutive CONNECT statements to additional relational databases do not result in disconnection of previous connections.

***RUW:** CONNECT (Type 1) semantics are used to support remote unit of work. Consecutive CONNECT statements result in the previous connection being disconnected before a new connection is established

DFTRDBCOL
Specifies the schema name used for the unqualified names of tables, views, indexes, and SQL packages. This parameter applies only to static SQL statements.

***NONE:** The naming convention defined on the OPTION parameter is used.

schema-name: Specify the name of the schema identifier. This value is used instead of the naming convention that is specified on the OPTION parameter.

DYNDFTCOL
Specifies whether the default schema name specified for the DFTRDBCOL parameter is also used for dynamic statements.

***NO:** Do not use the value specified on the DFTRDBCOL parameter for unqualified names of tables, views, indexes, and SQL packages for dynamic SQL statements. The naming convention specified on the OPTION parameter is used.

***YES:** The schema name specified on the DFTRDBCOL parameter will be used for the unqualified names of the tables, views, indexes, and SQL packages in dynamic SQL statements.

SQLPKG
Specifies the qualified name of the SQL package created on the relational database specified on the RDB parameter of this command.

The possible library values are:

***OBJLIB:** The package is created in the library with the same name as the library specified on the OBJ parameter.

library-name: Specify the name of the library where the package is created.

***OBJ:** The name of the SQL package is the same as the object name specified on the OBJ parameter.

package-name: Specify the name of the SQL package. If the remote system is not an iSeries system, no more than 8 characters can be specified.

SQLPATH
Specifies the path to be used to find procedures, functions, and user defined types in static SQL statements.

***NAMING:** The path used depends on the naming convention specified on the OPTION parameter.

For *SYS naming, the path used is *LIBL, the current library list at runtime.

For *SQL naming, the path used is ″QSYS″, ″QSYS2″, ″userid″, where ″userid″ is the value of the USER special register. If a schema-name is specified on the DFTRDBCOL parameter, the schema-name takes the place of userid.

***LIBL:** The path used is the library list at runtime.

schema-name: Specify a list of one or more schema names. A maximum of 268 individual schemas may be specified.

SQLCURRULE
Specifies the semantics used for SQL statements.

***DB2:** The semantics of all SQL statements will default to the rules established for DB2. The following semantics are controlled by this option:

- Hexadecimal constants are treated as character data.

***STD:** The semantics of all SQL statements will default to the rules established by the ISO and ANSI SQL standards. The following semantics are controlled by this option:

- Hexadecimal constants are treated as binary data.

SAAFLAG
Specifies the IBM SQL flagging function. This parameter flags SQL statements to verify whether they conform to IBM SQL syntax. More information about which IBM database products IBM SQL syntax is in the *DRDA IBM SQL Reference*, SC26-3255-00.

***NOFLAG:** The precompiler does not check to see whether SQL statements conform to IBM SQL syntax.

***FLAG:** The precompiler checks to see whether SQL statements conform to IBM SQL syntax.

FLAGSTD
Specifies the American National Standards Institute (ANSI) flagging function. This parameter flags SQL statements to verify whether they conform to the following standards.

```
ANSI X3.135-1992 entry
ISO 9075-1992 entry
FIPS 127.2 entry
```

***NONE:** The precompiler does not check to see whether SQL statements conform to ANSI standards.

***ANS:** The precompiler checks to see whether SQL statements conform to ANSI standards.

DBGVIEW

This parameter specifies the type of source debug information to be provided by the SQL precompiler.

***NONE:** The source view will not be generated.

***SOURCE:** The SQL precompiler provides the source views for the root and if necessary, SQL INCLUDE statements. A view is provided that contains the statements generated by the precompiler.

USRPRF

Specifies the user profile that is used when the compiled program object is run, including the authority that the program object has for each object in static SQL statements. The profile of either the program owner or the program user is used to control which objects can be used by the program object.

***NAMING:** The user profile is determined by the naming convention. If the naming convention is *SQL, USRPRF(*OWNER) is used. If the naming convention is *SYS, USRPRF(*USER) is used.

***USER:** The profile of the user running the program object is used.

***OWNER:** The user profiles of both the program owner and the program user are used when the program is run.

DYNUSRPRF

Specifies the user profile to be used for dynamic SQL statements.

***USER:** Local dynamic SQL statements are run under the profile of the program's user. Distributed dynamic SQL statements are run under the profile of the SQL package's user.

***OWNER:** Local dynamic SQL statements are run under the profile of the program's owner. Distributed dynamic SQL statements are run under the profile of the SQL package's owner.

SRTSEQ

Specifies the sort sequence table to be used for string comparisons in SQL statements.

Note: *HEX must be specified for this parameter on distributed applications where the application server is not on an iSeries system or the release level is prior to V2R3M0.

***JOB:** The SRTSEQ value for the job is retrieved during the precompile.

***JOBRUN:** The SRTSEQ value for the job is retrieved when the program is run. For distributed applications, SRTSEQ(*JOBRUN) is valid only when LANGID(*JOBRUN) is also specified.

***HEX:** A sort sequence table is not used. The hexadecimal values of the characters are used to determine the sort sequence.

***LANGIDSHR:** The sort sequence table uses a unique weight for each character, and is the unique-weight sort table for the language specified on the LANGID parameter.

***LANGIDUNQ:** The unique-weight sort table for the language that is specified on the LANGID parameter is used.

The name of the table name can be qualified by one of the following library values:

***LIBL:** All libraries in the job's library list are searched until the first match is found.

***CURLIB:** The current library for the job is searched. If no library is specified as the current library for the job, the QGPL library is used.

library-name: Specify the name of hte library to be searched.

table-name: Specify a table name.

LANGID

Specifies the language identifier to be used when SRTSEQ(*LANGIDUNQ) or SRTSEQ(*LANGIDSHR) is specified.

***JOB:** The LANGID value for the job is retrieved during the precompile.

***JOBRUN:**The LANGID value for the job is retrieved when the program is run. For distributed applications, LANGID(*JOBRUN) is valid only when SRTSEQ(*JOBRUN) is also specified.

language-identifier: Specify a language identifier.

OUTPUT

Specifies whether the precompiler listing is generated.

***NONE:** The precompiler listing is not generated.

***PRINT:** The precompiler listing is generated.

PRTFILE

Specifies the qualified name of the printer device file to which the precompiler printout is directed. The file must have a minimum length of 132 bytes. If a file with a record length of less than 132 bytes is specified, information is lost.

The name of the printer file can be qualified by one of the following library values:

***LIBL:** All libraries in the job's library list are searched until the first match is found.

***CURLIB:** The current library for the job is searched. If no library is specified as the current library for the job, the QGPL library is used.

library-name: Specify the name of the library to be searched.

QSYSPRT: If a file name is not specified, the precompiler printout is directed to the IBM-supplied printer file QSYSPRT.

printer-file-name: Specify the name of the printer device file to which the precompiler printout is directed.

TOSRCFILE

Specifies the qualified name of the source file that is to contain the output source member that has been processed by the SQL precompiler. If the specified source file is not found, it will be created. The output member will have the same name as the name that is specified for the SRCMBR parameter.

The possible library values are:

QTEMP: The library QTEMP will be used.

***LIBL:** The job's library list is searched for the specified file. If the file is not found in any library in the library list, the file will be created in the current library.

***CURLIB:** The current library for the job will be used. If no library is specified as the current library for the job, the QGPL library will be used.

library-name: Specify the name of the library that is to contain the output source file.

***CALC:** The output source file name will be generated based on the margins of the source file. The name will be QSQLTxxxxx, where xxxxx is the width of the source file. If the source file record length is less than or equal to 92, the name will be QSQLTEMP.

QSQLTEMP: The source file QSQLTEMP will be used.

source-file-name: Specify the name of the source file to contain the output source member.

TEXT

Specifies the text that briefly describes the program and the function. More information about this parameter is in the TEXT parameter topic in the CL Reference section of the Information Center.

***SRCMBRTXT:** The text is taken from the source file member being used to create the C++ program. You can add or change text for a database source member by using the Start Source Entry Utility (STRSEU) command. You can also use either the Add Physical File Member (ADDPFM) command or the Change Physical File Member (CHGPFM) command. If the source file is an inline file or a device file, the text is blank.

***BLANK:** Text is not specified.

'description': Specify no more than 50 characters of text, enclosed in apostrophes.

Example:

```
CRTSQLCPPI  PAYROLL  OBJTYPE(*MODULE)
  TEXT('Payroll Program')
```

This command runs the SQL precompiler which precompiles the source and stores the changed source in member PAYROLL in file QSQLTEMP in library QTEMP. The command calls the ILE C++ compiler to create module PAYROLL in the current library by using the source member that is created by the SQL precompiler.

CRTSQLPLI (Create Structured Query Language PL/I) Command

Job: B,I   Pgm: B,I   REXX: B,I   Exec

```
                        ┌─*CURLIB/───────┐
►►──CRTSQLPLI──PGM(─────┼────────────────┼──program-name──)────────────────►
                        └─library-name/──┘

►─┬──────────────────────────────────────────────────────────────┬─────────►
  │                                                              │
  └─SRCFILE(──┬─*LIBL/────────┬──┬─QPLISRC──────────┬──)─────────┘
              ├─*CURLIB/──────┤  └─source-file-name─┘
              └─library-name/─┘
```

CRTSQLPLI

```
                                                       (1)
►─┬──────────────────────────────────────────────┬──────────────────────────────────────►
  │             ┌─*PGM──────────────────────┐    │
  └─SRCMBR(─────┴─source-file-member-name───┴─)──┘

►─┬───────────────────────────────────┬──┬─────────────────────────────────┬────────────►
  └─OPTION(──┤ Option Details ├───────┘  │             ┌─*CURRENT─┐       │
                                         └─TGTRLS(─────┼─*PRV─────┼──)────┘
                                                       └─VxRxMx───┘

►─┬──────────────────────────────────────────────────────────────┬─────────────────────►
  │           ┌─*LIBL/──────────┐  ┌─*SRCFILE──────────┐          │
  └─INCFILE(──┼─────────────────┼──┴─source-file-name──┴──)───────┘
              ├─*CURLIB/────────┤
              └─library-name/───┘

►─┬─────────────────────────────┬──┬───────────────────────────────────┬───────────────►
  │            ┌─*UR───┐        │  │               ┌─*ENDPGM─┐         │
  │            ├─*CHG──┤        │  └─CLOSQLCSR(────┼─*ENDSQL─┼──)──────┘
  └─COMMIT(────┼─*ALL──┼──)─────┘                  └─*ENDJOB─┘
               ├─*RS───┤
               ├─*CS───┤
               ├─*NONE─┤
               ├─*NC───┤
               └─*RR───┘

►─┬──────────────────────────────────────┬──┬──────────────────────────────────┬───────►
  │              ┌─*OPTIMIZE─┐           │  │             ┌─*ALLREAD─┐         │
  └─ALWCPYDTA(───┼─*YES──────┼──)────────┘  └─ALWBLK(─────┼─*NONE────┼──)──────┘
                 └─*NO───────┘                            └─*READ────┘

►─┬────────────────────────────┬──┬─────────────────────────────────────┬──────────────►
  │           ┌─*NO──┐         │  │            ┌─10─────────────┐       │
  └─DLYPRP(───┴─*YES─┴──)──────┘  └─GENLVL(────┴─severity-level─┴──)────┘

►─┬────────────────────────────────────┬──┬──────────────────────────┬─────────────────►
  │            ┌─*SRCFILE───┐          │  │             ┌─*JOB─┐     │
  └─MARGINS(───┴─left-right─┴──)───────┘  └─DATFMT(─────┼─*USA─┼──)──┘
                                                        ├─*ISO─┤
                                                        ├─*EUR─┤
                                                        ├─*JIS─┤
                                                        ├─*MDY─┤
                                                        ├─*DMY─┤
                                                        ├─*YMD─┤
                                                        └─*JUL─┘
```

```
►──┬─────────────────────────────┬──┬──────────────────────────┬──────────────►
   │            ┌─*JOB───┐       │  │           ┌─*HMS─┐       │
   └─DATSEP(────┼─'/'────┼──)────┘  └─TIMFMT(───┼─*USA─┼──)────┘
                ├─'.'────┤                      ├─*ISO─┤
                ├─','────┤                      ├─*EUR─┤
                ├─'-'────┤                      └─*JIS─┘
                ├─' '────┤
                └─*BLANK─┘

►──┬─────────────────────────────┬────────────────────────────────────────────►
   │            ┌─*JOB───┐       │
   └─TIMSEP(────┼─':'────┼──)────┘
                ├─'.'────┤
                ├─','────┤
                ├─' '────┤
                └─*BLANK─┘

►──┬────────────────────────────┬──┬──────────────────────────────────────────┬──►
   │              ┌─*YES─┐      │  │      ┌─*LOCAL──────────────────────┐     │
   └─REPLACE(─────┴─*NO──┴──)───┘  └─RDB(─┼─relational-database-name────┼──)──┘
                                          └─*NONE───────────────────────┘

►──┬───────────────────────────────┬──┬──────────────────────────────────┬─────►
   │        ┌─*CURRENT──┐          │  │              ┌─*NONE─────┐       │
   └─USER(──┴─user-name─┴──)───────┘  └─PASSWORD(────┴─password──┴──)────┘

►──┬───────────────────────────────┬──┬──────────────────────────────────────┬─►
   │              ┌─*DUW─┐         │  │               ┌─*NONE───────────┐    │
   └─RDBCNNMTH(───┴─*RUW─┴──)──────┘  └─DFTRDBCOL(────┴─collection-name─┴──)─┘

►──┬───────────────────────────────┬───────────────────────────────────────────►
   │              ┌─*NO──┐         │
   └─DYNDFTCOL(───┴─*YES─┴──)──────┘

►──┬────────────────────────────────────────────────────────────┬──────────────►
   │          ┌─*PGMLIB/───────┐    ┌─*PGM─────────┐            │
   └─SQLPKG(──┼────────────────┼────┴─package-name─┴──)─────────┘
              └─library-name/──┘

►──┬───────────────────────────────────────────────┬──┬──────────────────────────┬─►
   │           ┌─*NAMING─────────────────┐         │  │              ┌─*DB2─┐    │
   └─SQLPATH(──┼─*LIBL───────────────────┼──)──────┘  └─SQLCURRULE(──┴─*STD─┴──)─┘
               │  ┌──────────────────┐   │
               │  ▼                  │   │
               └───collection-name───┴───┘

►──┬──────────────────────────────┬──┬──────────────────────────┬──────────────►
   │           ┌─*NOFLAG─┐        │  │           ┌─*NONE─┐      │
   └─SAAFLAG(──┴─*FLAG───┴──)─────┘  └─FLAGSTD(──┴─*ANS──┴──)───┘

►──┬──────────────────────────────────────────────────────────────┬────────────►
   │           ┌─*LIBL/─────────┐    ┌─QSYSPRT────────────┐       │
   └─PRTFILE(──┼────────────────┼────┴─printer-file-name──┴──)────┘
               ├─*CURLIB/───────┤
               └─library-name/──┘
```

```
►─┬──────────────────────────────────────────────────────┬──────────────────────►
  │              ┌─*JOB──────────────────────┐           │
  └─SRTSEQ(──────┼─*JOBRUN───────────────────┼──)────────┘
                 ├─*LANGIDUNQ────────────────┤
                 ├─*LANGIDSHR────────────────┤
                 ├─*HEX──────────────────────┤
                 │  ┌─*LIBL/──────────┐       │
                 └──┼─────────────────┼─table-name─┘
                    ├─*CURLIB/────────┤
                    └─library-name/───┘

►─┬──────────────────────────────────┬─┬────────────────────────────────┬───────►
  │          ┌─*JOB───────────┐      │ │          ┌─*NAMING─┐           │
  └─LANGID(──┼─*JOBRUN────────┼──)───┘ └─USRPRF(──┼─*OWNER──┼──)────────┘
             └─language-ID────┘                   └─*USER───┘

►─┬──────────────────────────────────┬──────────────────────────────────────────►
  │            ┌─*USER──┐            │
  └─DYNUSRPRF(─┴─*OWNER─┴──)─────────┘

►─┬─────────────────────────────────────────────────────────────────┬───────────►
  │             ┌─QTEMP/─────────┐ ┌─QSQLTEMP─────────┐             │
  └─TOSRCFILE(──┼────────────────┼─┴─source-file-name─┴──)──────────┘
                ├─*LIBL/─────────┤
                ├─*CURLIB/───────┤
                └─library-name/──┘

►─┬───────────────────────────────────┬─────────────────────────────────────────►◄
  │        ┌─*SCRMBRTXT────┐          │
  └─TEXT(──┼─*BLANK────────┼──)───────┘
           └─'description'─┘
```

Option Details:

```
   ┌─*NOSRC────┐
   ├─*NOSOURCE─┤   ┌─*NOXREF─┐   ┌─*GEN───┐   ┌─*JOB─────┐   ┌─*SYS─┐
├──┼───────────┼───┴─────────┴───┴────────┴───┼──────────┼───┴──────┴──────────►
   ├─*SRC──────┤   └─*XREF───┘   └─*NOGEN─┘   ├─*PERIOD──┤   └─*SQL─┘
   └─*SOURCE───┘                              ├─*SYSVAL──┤
                                              └─*COMMA───┘

   ┌─*NOSECLVL─┐   ┌─*OPTLOB───┐
►──┴───────────┴───┴───────────┴──)─────────────────────────────────────────────┤
   └─*SECLVL───┘   └─*NOOPTLOB─┘
```

Notes:

1 All parameters preceding this point can be specified in positional form.

Purpose:

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.

Parameters:

PGM

Specifies the qualified name of the compiled program.

The name of the compiled PL/I program can be qualified by one of the following library values:

***CURLIB:** The current library for the job is searched. If no library is specified as the current library for the job, the QGPL library is used.

library-name: Specify the name of the library where the compiled PL/I program is created.

program-name: Specify the name of the compiled program.

SRCFILE

Specifies the qualified name of the source file that contains the PL/I source with SQL statements.

The name of the source file can be qualified by one of the following library values:

***LIBL:** All libraries in the job's library list are searched until the first match is found.

***CURLIB:** The current library for the job is searched. If no library is specified as the current library for the job, the QGPL library is used.

library-name: Specify the name of the library to be searched.

QPLISRC: If the source file name is not specified, the IBM-supplied source file QPLISRC contains the PL/I source.

source-file-name: Specify the name of the source file that contains the PL/I source.

SRCMBR

Specifies the name of the source file member that contains the PL/I source. This parameter is specified only if the source file name in the SRCFILE parameter is a database file. If this parameter is not specified, the PGM name specified on the PGM parameter is used.

***PGM:** Specifies that the PL/I source is in the member of the source file that has the same name as that specified on the PGM parameter.

source-file-member-name: Specify the name of the member that contains the PL/I source.

OPTION

Specifies whether one or more of the following options are used when the PL/I source is precompiled. If an option is specified more than once, or if two options conflict, the last option specified is used.

Element 1: Source Listing Options

***NOSOURCE:** or ***NOSRC:** A source printout is not produced by the precompiler unless errors are detected during precompile or create package.

***SOURCE or *SRC:** The precompiler produces a source printout consisting of PL/I source input.

Element 2: Cross-Reference Options

***NOXREF:** The precompiler does not cross-reference names.

***XREF:** The precompiler cross-references items in the program to the statement numbers in the program that refer to those items.

Element 3: Program Creation Options

***GEN:** The compiler creates a program that can run after the program is compiled. An SQL package object is created if a relational database name is specified on the RDB parameter.

***NOGEN:** The precompiler does not call the C compiler, and a program and SQL package are not created.

Element 4: Decimal Point Options

***JOB:** The value used as the decimal point for numeric constants in SQL is the representation of decimal point specified for the job at precompile time.

***PERIOD:** The value used as the decimal point for numeric constants used in SQL statements is a period.

***SYSVAL:** The value used as the decimal point for numeric constants in SQL statements is the QDECFMT system value.

Note: If QDECFMT specifies that the value used as the decimal point is a comma, any numeric constants in lists (such as in the SELECT clause or the VALUES clause) must be separated by a comma followed by a blank. For example, VALUES(1,1, 2,23, 4,1) is equivalent to VALUES(1.1,2.23,4.1) in which the decimal point is a period.

***COMMA:** The value used as the decimal point for numeric constants in SQL statements is a comma.

Note: Any numeric constants in lists (such as in the SELECT clause or the VALUES clause) must be separated by a comma followed by a blank. For example, VALUES(1,1, 2,23, 4,1) is equivalent to VALUES(1.1,2.23,4.1) where the decimal point is a period.

Element 5: Naming Convention Options

***SYS:** The system naming convention (library-name/file-name) is used.

***SQL:** The SQL naming convention is used (schema-name.table-name). When creating a program on a remote database other than an iSeries system, *SQL must be specified as the naming convention.

Element 6: Second-Level Message Text Option

***NOSECLVL:** Second-level text descriptions are not added to the listing.

***SECLVL:** Second-level text with replacement data is added to the printout for all messages on the listing.

Element 7: Large Object Optimization for DRDA Option

***OPTLOB:** The first FETCH for a cursor determines how the cursor will be used for LOBs (Large Objects) on all subsequent FETCHes. This option remains in effect until the cursor is closed.

If the first FETCH uses a LOB locator to access a LOB column, no subsequent FETCH for that cursor can fetch that LOB column into a LOB host variable.

If the first FETCH places the LOB column into a LOB host variable, no subsequent FETCH for that cursor can use a LOB locator for that column.

***NOOPTLOB:** There is no restriction on whether a column is retrieved into a LOB locator or into a LOB host variable. This option can cause performance to degrade.

TGTRLS

Specifies the release of the operating system on which the user intends to use the object being created.

In the examples given for the *CURRENT and *PRV values, and when specifying the *release-level* value, the format VxRxMx is used to specify the release, where Vx is the version, Rx is the release, and Mx is the modification level. For example, V2R3M0 is version 2, release 3, modification level 0.

***CURRENT:** The object is to be used on the release of the operating system currently running on the user's system. For example, if V2R3M5 is running on the system, *CURRENT means the user intends to use the object on a system with V2R3M5 installed. The user can also use the object on a system with any subsequent release of the operating system installed.

Note: If V2R3M5 is running on the system, and the object is to be used on a system with V2R3M0 installed, specify TGTRLS(V2R3M0) not TGTRLS(*CURRENT).

***PRV:** The object is to be used on the previous release with modification level 0 of the operating system. For example, if V2R3M5 is running on the user's system, *PRV means the user intends to use the object on a system with V2R2M0 installed. The user can also use the object on a system with any subsequent release of the operating system installed.

release-level: Specify the release in the format VxRxMx. The object can be used on a system with the specified release or with any subsequent release of the operating system installed.

Valid values depend on the current version, release, and modification level, and they change with each new release. If you specify a release-level which is earlier than the earliest release level supported by this command, an error message is sent indicating the earliest supported release.

INCFILE

Specifies the qualified name of the source file that contains members included in the program with any SQL INCLUDE statement.

The name of the source file can be qualified by one of the following library values:

***LIBL:** All libraries in the job's library list are searched until the first match is found.

***CURLIB:** The current library for the job is searched. If no library is specified as the current library for the job, the QGPL library is used.

library-name: Specify the name of the library to be searched.

***SRCFILE:** The qualified source file specified in the SRCFILE parameter contains the source file members specified on any SQL INCLUDE statement.

source-file-name: Specify the name of the source file that contains the source file members specified on any SQL INCLUDE statement. The record length of the source file specified must be no less than the record length of the source file specified for the SRCFILE parameter.

COMMIT

Specifies whether SQL statements in the compiled program are run under commitment control. Files referred to in the host language source are not affected by this option. Only SQL tables, SQL views, and SQL packages referred to in SQL statements are affected.

***CHG or *UR:** Specifies the objects referred to in SQL ALTER, CALL, COMMENT ON, CREATE, DROP, GRANT, LABEL ON, RENAME, and REVOKE statements and the rows updated, deleted, and inserted are locked until the end of the unit of work (transaction). Uncommitted changes in other jobs can be seen.

***ALL or *RS:** Specifies the objects referred to in SQL ALTER, CALL, COMMENT ON, CREATE, DROP, GRANT, LABEL ON, RENAME, and REVOKE statements and the rows selected, updated, deleted, and inserted are locked until the end of the unit of work (transaction). Uncommitted changes in other jobs cannot be seen.

***CS:** Specifies the objects referred to in SQL ALTER, CALL, COMMENT ON, CREATE, DROP, GRANT, LABEL ON, RENAME, and REVOKE statements and the rows updated, deleted, and inserted are locked until the end of the unit of work (transaction). A row that is selected, but not updated, is locked until the next row is selected. Uncommitted changes in other jobs cannot be seen.

***NONE or *NC:** Specifies that commitment control is not used. Uncommitted changes in other jobs can be seen. If the SQL DROP SCHEMA statement is included in the program, *NONE or *NC must be used. If a relational database is specified on the RDB parameter and the relational database is on a system that is not on an iSeries, *NONE or *NC cannot be specified.

***RR:** Specifies the objects referred to in SQL ALTER, CALL, COMMENT ON, CREATE, DROP, GRANT, LABEL ON, RENAME, and REVOKE statements and the rows selected, updated, deleted, and inserted are locked until the end of the unit of work (transaction). Uncommitted changes in other jobs cannot be seen. All tables referred to in SELECT, UPDATE, DELETE, and INSERT statements are locked exclusively until the end of the unit of work (transaction).

CLOSQLCSR

Specifies when SQL cursors are implicitly closed, SQL prepared statements are implicitly discarded, and LOCK TABLE locks are released. SQL cursors are explicitly closed when you issue the CLOSE, COMMIT, or ROLLBACK (without HOLD) SQL statements.

***ENDPGM:** SQL cursors are closed and SQL prepared statements are discarded when the program ends. LOCK TABLE locks are released when the first SQL program on the call stack ends.

***ENDSQL:** SQL cursors remain open between calls and can be fetched without running another SQL OPEN. One of the programs higher on the call stack must have run at least one SQL statement. SQL cursors are closed, SQL prepared statements are discarded, and LOCK TABLE locks are released when the first SQL program on the call stack ends. If *ENDSQL is specified for a program that is the first SQL program called (the first SQL program on the call stack), the program is treated as if *ENDPGM was specified.

***ENDJOB:** SQL cursors remain open between calls and can be fetched without running another SQL OPEN. The programs higher on the call stack do not need to have run SQL statements. SQL cursors are left open, SQL prepared statements are preserved, and LOCK TABLE locks are held when the first SQL program on the call stack ends. SQL cursors are closed, SQL prepared statements are discarded, and LOCK TABLE locks are released when the job ends.

ALWCPYDTA

Specifies whether a copy of the data can be used in a SELECT statement.

***OPTIMIZE:** The system determines whether to use the data retrieved directly from the database or to use a copy of the data. The decision is based on which method provides the best performance. If COMMIT is *CHG or *CS and ALWBLK is not *ALLREAD, or if COMMIT is *ALL or *RR, then a copy of the data is used only when it is necessary to run a query.

***YES:** A copy of the data is used only when necessary.

***NO:** A copy of the data is not allowed. If a temporary copy of the data is required to perform the query, an error message is returned.

ALWBLK

Specifies whether the database manager can use record blocking, and the extent to which blocking can be used for read-only cursors.

***ALLREAD:** Rows are blocked for read-only cursors if *NONE or *CHG is specified on the COMMIT parameter. All cursors in a program that are not explicitly able to be updated are opened for read-only processing even though EXECUTE or EXECUTE IMMEDIATE statements may be in the program.

Specifying *ALLREAD:

- Allows record blocking under commitment control level *CHG in addition to the blocking allowed for *READ.
- Can improve the performance of almost all read-only cursors in programs, but limits queries in the following ways:
  - The Rollback (ROLLBACK) command, a ROLLBACK statement in host languages, or the ROLLBACK HOLD SQL statement does not reposition a read-only cursor when *ALLREAD is specified.
  - Dynamic running of a positioned UPDATE or DELETE statement (for example, using EXECUTE IMMEDIATE), cannot be used to update a row in a cursor unless the DECLARE statement for the cursor includes the FOR UPDATE clause.

***NONE:** Rows are not blocked for retrieval of data for cursors.

Specifying *NONE:

- Guarantees that the data retrieved is current.
- May reduce the amount of time required to retrieve the first row of data for a query.
- Stops the database manager from retrieving a block of data rows that is not used by the program when only the first few rows of a query are retrieved before the query is closed.
- Can degrade the overall performance of a query that retrieves a large number of rows.

***READ:** Records are blocked for read-only retrieval of data for cursors when:

- *NONE is specified on the COMMIT parameter, which indicates that commitment control is not used.
- The cursor is declared with a FOR READ ONLY clause or there are no dynamic statements that could run a positioned UPDATE or DELETE statement for the cursor.

Specifying *READ can improve the overall performance of queries that meet the above conditions and retrieve a large number of records.

DLYPRP

Specifies whether the dynamic statement validation for a PREPARE statement is delayed until an OPEN, EXECUTE, or DESCRIBE statement is run. Delaying validation improves performance by eliminating redundant validation.

***NO:** Dynamic statement validation is not delayed. When the dynamic statement is prepared, the access plan is validated. When the dynamic statement is used in an OPEN or EXECUTE statement, the access plan is revalidated. Because the authority or the existence of objects referred to by the dynamic statement may change, you must still check the SQLCODE or SQLSTATE after issuing the OPEN or EXECUTE statement to ensure that the dynamic statement is still valid.

***YES:** Dynamic statement validation is delayed until the dynamic statement is used in an OPEN, EXECUTE, or DESCRIBE SQL statement. When the dynamic statement is used, the validation is completed and an access plan is built. If you specify *YES on this parameter, you should check the SQLCODE and SQLSTATE after running an OPEN, EXECUTE, or DESCRIBE statement to ensure that the dynamic statement is valid.

Note: If you specify *YES, performance is not improved if the INTO clause is used on the PREPARE statement or if a DESCRIBE statement uses the dynamic statement before an OPEN is issued for the statement.

GENLVL

Specifies the severity level of errors at which the program is not changed. If, while preparing SQL statements in the program, errors occur with a severity level equal to or greater than the value specified on this parameter, the program is not changed.

10: The default severity level is 10.

severity-level: Specify a value ranging from 0 through 40.

MARGINS

Specifies the part of the precompiler input record that contains source text.

***SRCFILE:** The file member margin values specified by the user on the SRCMBR parameter are used. If the member is a SQLPLI source type, the margin values are the values specified on the SEU services display. If the member is a different source type, the margin values are the default values of 2 and 72.

Element 1: Left Margin

left: Specify the beginning position for the statements. Valid values range from 1 through 80.

Element 2: Right Margin

right: Specify the ending position for the statements. Valid values range from 1 through 80.

DATFMT

Specifies the format used when accessing date result columns. All output date fields are returned in the specified format. For input date strings, the specified value is used to determine whether the date is specified in a valid format.

Note: An input date string that uses the format *USA, *ISO, *EUR, or *JIS is always valid.

If a relational database is specified on the RDB parameter and the database is on a system that is not an iSeries system, then *USA, *ISO, *EUR, or *JIS must be specified.

***JOB:** The format specified for the job is used. Use the Display Job (DSPJOB) command to determine the current date format for the job.

***USA:** The United States date format (mm/dd/yyyy) is used.

***ISO:** The International Organization for Standardization (ISO) date format (yyyy-mm-dd) is used.

***EUR:** The European date format (dd.mm.yyyy) is used.

***JIS:** The Japanese Industrial Standard date format (yyyy-mm-dd) is used.

***MDY:** The date format (mm/dd/yy) is used.

***DMY:** The date format (dd/mm/yy) is used.

***YMD:** The date format (yy/mm/dd) is used.

***JUL:** The Julian date format (yy/ddd) is used.

DATSEP

Specifies the separator used when accessing date result columns.

Note: This parameter applies only when *JOB, *MDY, *DMY, *YMD, or *JUL is specified on the DATFMT parameter.

***JOB:** The date separator specified for the job at precompile time is used. Use the Display Job (DSPJOB) command to determine the current value for the job.

'/': A slash (/) is used.

'.': A period (.) is used.

',': A comma (,) is used.

'-': A dash (-) is used.

' ': A blank ( ) is used.

***BLANK:** A blank ( ) is used.

TIMFMT

Specifies the format used when accessing time result columns. For input time strings, the specified value is used to determine whether the time is specified in a valid format.

Note: An input date string that uses the format *USA, *ISO, *EUR, or *JIS is always valid.

If a relational database is specified on the RDB parameter and the database is on a system that is not another iSeries system, the time format must be *USA, *ISO, *EUR, *JIS, or *HMS with a time separator of colon or period.

***HMS:** The (hh:mm:ss) format is used.

***USA:** The United States time format (hh:mm *xx*) is used, where *xx* is AM or PM.

***ISO:** The International Organization for Standardization (ISO) time format (hh.mm.ss) is used.

***EUR:** The European time format (hh.mm.ss) is used.

***JIS:** The Japanese Industrial Standard time format (hh:mm:ss) is used.

TIMSEP
Specifies the separator used when accessing time result columns.

Note: This parameter applies only when *HMS is specified on the TIMFMT parameter.

***JOB:** The time separator specified for the job at precompile time is used. Use the Display Job (DSPJOB) command to determine the current value for the job.

':': A colon (:) is used.

'.': A period (.) is used.

',': A comma (,) is used.

' ': A blank ( ) is used.

***BLANK:** A blank ( ) is used.

REPLACE
Specifies whether a new program or SQL package is created when a program or SQL package of the same name exists in the same library. The value of this parameter is passed to the CRTPLIPGM command. More information about this parameter is in Appendix A, "Expanded Parameter Descriptions" in the CL Reference book.

***YES:** A new program or SQL package is created, and any existing program or SQL package of the same name and type in the specified library is moved to QRPLOBJ.

***NO:** A new program or SQL package is not created if an object of the same name and type already exists in the specified library.

RDB
Specifies the name of the relational database where the SQL package object is created.

***LOCAL:** The program is created as a distributed SQL program. The SQL statements will access the local database. An SQL package object is not created as part of the precompile process. The Create Structured Query Language Package (CRTSQLPKG) command can be used.

relational-database-name: Specify the name of the relational database where the new SQL package object is to be created. When the name of the local relational database is specified, the program created is still a distributed SQL program. The SQL statements will access the local database.

***NONE:** An SQL package object is not created. The program object is not a distributed program and the Create Structured Query Language Package (CRTSQLPKG) command cannot be used.

USER

Specifies the user name sent to the remote system when starting the conversation. This parameter is valid only when RDB is specified.

***CURRENT:** The user profile under which the current job is running is used.

user-name: Specify the user name being used for the application server job.

PASSWORD

Specifies the password to be used on the remote system. This parameter is valid only if RDB is specified.

***NONE:** No password is sent. If this value is specified, USER(*CURRENT) must also be specified.

password: Specify the password of the user name specified on the USER parameter.

RDBCNNMTH

Specifies the semantics used for CONNECT statements. Refer to the CONNECT (TYPE1) and CONNECT (TYPE2) in the *SQL Reference* book for more information.

***DUW:** CONNECT (Type 2) semantics are used to support distributed unit of work. Consecutive CONNECT statements to additional relational databases do not result in disconnection of previous connections.

***RUW:** Only one connection to a relational database is allowed. Consecutive CONNECT statements result in the previous connections being disconnected before a new connection is established.

DFTRDBCOL

Specifies the schema name used for the unqualified names of tables, views, indexes, and SQL packages. This parameter applies only to static SQL statements.

***NONE:** The naming convention defined on the OPTION parameter is used.

schema-name: Specify the name of the schema identifier. This value is used instead of the naming convention specified on the OPTION parameter.

DYNDFTCOL

Specifies whether the default schema name specified for the DFTRDBCOL parameter is also used for dynamic statements.

***NO:** Do not use the value specified on the DFTRDBCOL parameter for unqualified names of tables, views, indexes, and SQL packages for dynamic SQL statements. The naming convention specified on the OPTION parameter is used.

***YES:** The schema name specified on the DFTRDBCOL parameter will be used for the unqualified names of the tables, views, indexes, and SQL packages in dynamic SQL statements.

SQLPKG
Specifies the qualified name of the SQL package created on the relational database specified on the RDB parameter of this command.

The possible library values are:

***PGMLIB:** The package is created in the library with the same name as the library containing the program.

library-name: Specify the name of the library where the package is created.

***PGM:** The package name is the same as the program name.

package-name: Specify the name of the package created on the remote database specified on the RDBNAME parameter.

SQLPATH
Specifies the path to be used to find procedures, functions, and user defined types in static SQL statements.

***NAMING:** The path used depends on the naming convention specified on the OPTION parameter.

For *SYS naming, the path used is *LIBL, the current library list at runtime.

For *SQL naming, the path used is ″QSYS″, ″QSYS2″, ″userid″, where ″userid″ is the value of the USER special register. If a schema-name is specified on the DFTRDBCOL parameter, the schema-name takes the place of userid.

***LIBL:** The path used is the library list at runtime.

schema-name: Specify a list of one or more schema names. A maximum of 268 individual schemas may be specified.

SQLCURRULE
Specifies the semantics used for SQL statements.

***DB2:** The semantics of all SQL statements will default to the rules established for DB2. The following semantics are controlled by this option:

- Hexadecimal constants are treated as character data.

***STD:** The semantics of all SQL statements will default to the rules established by the ISO and ANSI SQL standards. The following semantics are controlled by this option:

- Hexadecimal constants are treated as binary data.

SAAFLAG
Specifies the IBM SQL flagging function. This parameter flags SQL statements to verify whether they conform to IBM SQL syntax. More information about which IBM database products IBM SQL syntax is in the *DRDA IBM SQL Reference*, SC26-3255-00.

***NOFLAG:** The precompiler does not check to see whether SQL statements conform to IBM SQL syntax.

***FLAG:** The precompiler checks to see whether SQL statements conform to IBM SQL syntax.

FLAGSTD

Specifies the American National Standards Institute (ANSI) flagging function. This parameter flags SQL statements to verify whether they conform to the following standards.

```
ANSI X3.135-1992 entry
ISO 9075-1992 entry
FIPS 127.2 entry
```

Specifies the American National Standards Institute (ANSI) flagging function. This parameter flags SQL statements to verify whether they conform to the following standards.

```
ANSI X3.135-1992 entry
ISO 9075-1992 entry
FIPS 127.2 entry
```


***NONE:** The precompiler does not check to see whether SQL statements conform to ANSI standards.

***ANS:** The precompiler checks to see whether SQL statements conform to ANSI standards.

PRTFILE

Specifies the qualified name of the printer device file to which the listing is directed. The file must have a minimum record length of 132 bytes or information is lost.

The name of the printer file can be qualified by one of the following library values:

***LIBL:** All libraries in the job's library list are searched until the first match is found.

***CURLIB:** The current library for the job is searched. If no library is specified as the current library for the job, the QGPL library is used.

library-name: Specify the name of the library to be searched.

QSYSPRT: If a file name is not specified, the precompiler printout is directed to the IBM-supplied printer file QSYSPRT.

printer-file-name: Specify the name of the printer device file to which the precompiler printout is directed.

SRTSEQ

Specifies the sort sequence table to be used for string comparisons in SQL statements.

Note: *HEX must be specified for this parameter on distributed applications where the application server is not on an iSeries system or the release level is prior to V2R3M0.

***JOB:** The SRTSEQ value for the job is retrieved during the precompile.

***JOBRUN:** The SRTSEQ value for the job is retrieved when the program is run. For distributed applications, SRTSEQ(*JOBRUN) is valid only when LANGID(*JOBRUN) is also specified.

***LANGIDUNQ:** The sort sequence table must contain a unique weight for each character in the code page.

***LANGIDSHR:** The shared-weight sort table for the language specified on the LANGID parameter is used.

***HEX:** A sort sequence table is not used. The hexadecimal values of the characters are used to determine the sort sequence.

The name of the sort sequence table can be qualified by one of thefollowing library values:

***LIBL:** All libraries in the job's library list are searched until the first match is found.

***CURLIB:** The current library for the job is searched. If no library is specified as the current library for the job, the QGPL library is used.

library-name: Specify the name of the library to be searched.

table-name: Specify the name of the sort sequence table to be used.

LANGID
Specifies the language identifier to be used when SRTSEQ(*LANGIDUNQ) or SRTSEQ(*LANGIDSHR) is specified.

***JOB:** The LANGID value for the job is retrieved during the precompile.

***JOBRUN:** The LANGID value for the job is retrieved when the program is run. For distributed applications, LANGID(*JOBRUN) is valid only when SRTSEQ(*JOBRUN) is also specified.

language-id: Specify a language identifier to be used by the program.

USRPRF
Specifies the user profile that is used when the compiled program object is run, including the authority that the program object has for each object in static SQL statements. The profile of either the program owner or the program user is used to control which objects can be used by the program object.

***NAMING:** The user profile is determined by the naming convention. If the naming convention is *SQL, USRPRF(*OWNER) is used. If the naming convention is *SYS, USRPRF(*USER) is used.

***USER:** The profile of the user running the program object is used.

***OWNER:** The user profiles of both the program owner and the program user are used when the program is run.

DYNUSRPRF
Specifies the user profile used for dynamic SQL statements.

***USER:** The program runs under the user profile of the program's user.

***OWNER:** Local dynamic SQL statements are run under the user profile of the program's owner. Distributed dynamic SQL statements are run under the user profile of the SQL package's owner.

TOSRCFILE
Specifies the qualified name of the source file that is to contain the output source member that has been processed by the SQL precompiler. If the specified source file is not found, it will be created. The output member will have the same name as the name that is specified for the SRCMBR parameter.

The possible library values are:

QTEMP: The library QTEMP will be used.

***LIBL:** The job's library list is searched for the specified file. If the file is not found in any library in the library list, the file will be created in the current library.

***CURLIB:** The current library for the job will be used. If no library is specified as the current library for the job, the QGPL library will be used.

library-name: Specify the name of the library that is to contain the output source file.

QSQLTEMP: The source file QSQLTEMP will be used.

source-file-name: Specify the name of the source file to contain the output source member.

TEXT

Specifies the text that briefly describes the program and its function. More information about this parameter is in the TEXT parameter topic in the CL Reference section of the Information Center.

***SCRMBRTXT:** The text is taken from the source file member being used to create the PL/I program. The user can add or change text for a database source member by using the Start Source Entry Utility (STRSEU) command, or by using either the Add Physical File Member (ADDPFM) or Change Physical File Member (CHGPFM) command. If the source file is an inline file or a device file, the text is blank.

***BLANK:** Text is not specified.

'description': Specify no more than 50 characters of text, enclosed in apostrophes.

Example:

```
CRTSQLPLI   PAYROLL  TEXT('Payroll Program')
```

This command runs the SQL precompiler, which precompiles the source and stores the changed source in member PAYROLL in file QSQLTEMP in library QTEMP. The PL/I compiler is called to create program PAYROLL in the current library using the source member created by the SQL precompiler.

CRTSQLRPG (Create Structured Query Language RPG) Command

Job: B,I   Pgm: B,I   REXX: B,I   Exec

```
              ┌─*CURLIB/──────┐
►►──CRTSQLRPG──PGM(─┴─library-name/─┴──program-name──)──────────────►

►─┬─────────────────────────────────────────────────────┬──────────►
  │          ┌─*LIBL/────────┐ ┌─QRPGSRC──────────┐     │
  └─SRCFILE(─┼───────────────┼─┴─source-file-name─┴──)─┘
             ├─*CURLIB/──────┤
             └─library-name/─┘
```

CRTSQLRPG

```
                                                 (1)
►─┬────────────────────────────────────────────┬───────────────────────────►
  │            ┌─*PGM────────────────────┐      │
  └─SRCMBR(────┴─source-file-member-name─┴──)─┘

►─┬──────────────────────────────────┬─┬──────────────────────────────┬────►
  └─OPTION(──┤ OPTION DETAILS ├──)─┘ │             ┌─*CURRENT─┐       │
                                     └─TGTRLS(────┼─*PRV─────┼──)───┘
                                                  └─VxRxMx───┘

►─┬────────────────────────────────────────────────────────────────┬───────►
  │             ┌─*LIBL/─────────┐ ┌─*SRCFILE─────────┐          │
  └─INCFILE(───┼────────────────┼─┴─source-file-name─┴──)──────┘
                ├─*CURLIB/───────┤
                └─library-name/──┘

►─┬──────────────────────────┬─┬──────────────────────────────────┬────────►
  │          ┌─*UR────┐      │ │                ┌─*ENDPGM─┐       │
  │          ├─*CHG───┤      │ └─CLOSQLCSR(────┼─*ENDSQL─┼──)───┘
  └─COMMIT(──┼─*ALL───┼──)─┘                  └─*ENDJOB─┘
             ├─*RS────┤
             ├─*CS────┤
             ├─*NONE──┤
             ├─*NC────┤
             └─*RR────┘

►─┬──────────────────────────────────┬─┬─────────────────────────────┬─────►
  │             ┌─*OPTIMIZE─┐        │ │          ┌─*ALLREAD─┐       │
  └─ALWCPYDTA(──┼─*YES──────┼──)───┘ └─ALWBLK(──┼─*NONE────┼──)───┘
                └─*NO───────┘                    └─*READ────┘

►─┬────────────────────────┬─┬────────────────────────────────┬────────────►
  │          ┌─*NO──┐      │ │          ┌─10─────────────┐    │
  └─DLYPRP(──┴─*YES─┴──)─┘ └─GENLVL(──┴─severity-level─┴──)─┘

►─┬────────────────────────┬─┬────────────────────────────┬────────────────►
  │          ┌─*JOB─┐      │ │          ┌─*JOB────┐       │
  └─DATFMT(──┼─*USA─┼──)─┘ └─DATSEP(──┼─'/'─────┼──)───┘
             ├─*ISO─┤                   ├─'.'─────┤
             ├─*EUR─┤                   ├─','─────┤
             ├─*JIS─┤                   ├─'-'─────┤
             ├─*MDY─┤                   ├─' '─────┤
             ├─*DMY─┤                   └─*BLANK──┘
             ├─*YMD─┤
             └─*JUL─┘
```

```
►─┬────────────────────────────┬─┬───────────────────────────┬──────────►
  │            ┌─*HMS─┐        │ │            ┌─*JOB───┐     │
  └─TIMFMT(────┼─*USA─┼──)─────┘ └─TIMSEP(────┼─':'────┼──)──┘
               ├─*ISO─┤                       ├─'.'────┤
               ├─*EUR─┤                       ├─','────┤
               └─*JIS─┘                       ├─' '────┤
                                              └─*BLANK─┘

►─┬────────────────────────┬────────────────────────────────────────────►
  │           ┌─*YES─┐     │
  └─REPLACE(──┴─*NO──┴──)──┘

►─┬─────────────────────────────────────────────┬─┬─────────────────────────────┬──►
  │      ┌─*LOCAL──────────────────────┐        │ │       ┌─*CURRENT──┐         │
  └─RDB(─┼─relational-database-name────┼──)─────┘ └─USER(─┴─user-name─┴──)──────┘
         └─*NONE───────────────────────┘

►─┬──────────────────────────────────┬─┬──────────────────────────────┬──►
  │            ┌─*NONE─────┐         │ │             ┌─*DUW─┐         │
  └─PASSWORD(──┴─password──┴──)──────┘ └─RDBCNNMTH(──┴─*RUW─┴──)──────┘

►─┬─────────────────────────────────────────┬─┬────────────────────────────┬──►
  │              ┌─*NONE────────────┐       │ │             ┌─*NO──┐       │
  └─DFTRDBCOL(───┴─collection-name──┴──)────┘ └─DYNDFTCOL(──┴─*YES─┴──)────┘

►─┬────────────────────────────────────────────────────────┬────────────►
  │          ┌─*PGMLIB/──────┐ ┌─*PGM──────────┐           │
  └─SQLPKG(──┼───────────────┼─┴─package-name──┴──)────────┘
             └─library-name/─┘

►─┬─────────────────────────────────────────────┬─┬──────────────────────────────┬──►
  │           ┌─*NAMING───────────────┐         │ │              ┌─*DB2─┐        │
  └─SQLPATH(──┼─*LIBL─────────────────┼──)──────┘ └─SQLCURRULE(──┴─*STD─┴──)─────┘
              │  ┌─────────────────┐  │
              │  ▼                 │  │
              └───collection-name──┴──┘

►─┬─────────────────────────────────┬─┬─────────────────────────────┬──►
  │           ┌─*NOFLAG─┐           │ │            ┌─*NONE─┐        │
  └─SAAFLAG(──┴─*FLAG───┴──)────────┘ └─FLAGSTD(───┴─*ANS──┴──)─────┘

►─┬─────────────────────────────────────────────────────────────┬───────►
  │           ┌─*LIBL/────────┐ ┌─QSYSPRT────────────┐          │
  └─PRTFILE(──┼───────────────┼─┴─printer-file-name──┴──)───────┘
              ├─*CURLIB/──────┤
              └─library-name/─┘
```

CRTSQLRPG

```
►─┬─────────────────────────────────────────────────────────┬───────►
  │          ┌─*JOB──────────────────────────┐              │
  └─SRTSEQ(──┼─*JOBRUN───────────────────────┼──)───────────┘
             ├─*LANGIDUNQ────────────────────┤
             ├─*LANGIDSHR────────────────────┤
             ├─*HEX──────────────────────────┤
             │ ┌─*LIBL/────────┐             │
             └─┼───────────────┼─table-name──┘
               ├─*CURLIB/──────┤
               └─library-name/─┘

►─┬──────────────────────────────┬─┬─────────────────────────────┬───►
  │          ┌─*JOB─────────┐    │ │          ┌─*NAMING─┐        │
  └─LANGID(──┼─*JOBRUN──────┼──)─┘ └─USRPRF(──┼─*OWNER──┼──)─────┘
             └─language-ID──┘                 └─*USER───┘

►─┬─────────────────────────────┬────────────────────────────────────►
  │             ┌─*USER──┐      │
  └─DYNUSRPRF(──┴─*OWNER─┴──)───┘

►─┬──────────────────────────────────────────────────────────┬───────►
  │             ┌─QTEMP/────────┐ ┌─QSQLTEMP─────────┐       │
  └─TOSRCFILE(──┼───────────────┼─┴─source-file-name─┴──)────┘
                ├─*LIBL/────────┤
                ├─*CURLIB/──────┤
                └─library-name/─┘

►─┬─────────────────────────────┬───────────────────────────────────►◄
  │        ┌─*SRCMBRTXT────┐    │
  └─TEXT(──┼─*BLANK────────┼──)─┘
           └─'description'─┘
```

OPTION Details:

```
  ┌─*NOSRC────┐
  ├─*NOSOURCE─┤ ┌─*NOXREF─┐ ┌─*GEN───┐ ┌─*JOB────┐ ┌─*SYS─┐
├─┼───────────┼─┼─────────┼─┼────────┼─┼─────────┼─┼──────┼────────►
  ├─*SOURCE───┤ └─*XREF───┘ └─*NOGEN─┘ ├─*SYSVAL─┤ └─*SQL─┘
  └─*SRC──────┘                        ├─*PERIOD─┤
                                       └─*COMMA──┘

  ┌─*NOSECLVL─┐ ┌─*NOSEQSRC─┐ ┌─*NOLSTDBG─┐
►─┼───────────┼─┼───────────┼─┼───────────┼───────────────────────┤
  └─*SECLVL───┘ └─*SEQSRC───┘ └─*LSTDBG───┘
```

Notes:

1 All parameters preceding this point can be specified in positional form.

Purpose:

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.

Parameters:

PGM

Specifies the qualified name of the compiled program.

The name of the compiled RPG can be qualified by one of the following library values:

***CURLIB:** The compiled RPG program is created in the current library for the job. If no library is specified as the current library for the job, the QGPL library is used.

library-name: Specify the name of hte library where the compiled RPG program is created.

program-name: Specify the name of the compiled program.

SRCFILE

Specifies the qualified name of the source file that contains the RPG source with SQL statements.

The name of the source file can be qualified by one of the following library values:

***LIBL:** All libraries in the job's library list are searched until the first match is found.

***CURLIB:** The current library for the job is searched. If no library is specified as the current library for the job, the QGPL library is used.

library-name: Specify the name of the library to be searched.

QRPGSRC: If the source file name is not specified, the IBM-supplied source file QRPGSRC contains the RPG source.

source-file-name: Specify the name of the source file that contains the RPG source.

SRCMBR

Specifies the name of hte source file member that contains the RPG source. This parameter is specified only if the source file name in the SRCFILE parameter is a database file. If this parameter is not specified, the PGM name specified on the PGM parameter is used.

***PGM:** Specifies that the RPG source is in the member of the source file that has the same name as that specified on the PGM parameter.

source-file-member-name: Specify the name of the member that contains the RPG source.

OPTION

Specifies whether one or more of the following options are used when the RPG source is precompiled. If an option is specified more than once, or if two options conflict, the last option specified is used.

Element 1: Source Listing Options

***NOSOURCE** or ***NOSRC:** A source printout is not produced by the precompiler unless errors are detected during precompile or create package.

***SOURCE or *SRC:** The precompiler produces a source printout, consisting of RPG source input.

Element 2: Cross-Reference Options

***NOXREF:** The precompiler does not cross-reference names.

***XREF:** The precompiler cross-references items in the program to the statement numbers in the program that refer to those items.

Element 3: Program Creation Options

***GEN:** The compiler creates a program that can run after the program is compiled. An SQL package object is created if a relational database name is specified on the RDB parameter.

***NOGEN:** The precompiler does not call the RPG compiler, and a program and SQL package are not created.

Element 4: Decimal Point Options

***JOB:** The value used as the decimal point for numeric constants in SQL is the representation of decimal point specified for the job at precompile time.

***SYSVAL:** The value used as the decimal point for numeric constants in SQL statements is the QDECFMT system value.

Note: If QDECFMT specifies that the value used as the decimal point is a comma, any numeric constants in lists (such as in the SELECT clause, VALUES clause, and so on.) must be separated by a comma followed by a blank. For example, VALUES(1,1, 2,23, 4,1) is equivalent to VALUES(1.1,2.23,4.1) where the decimal point is a period.

***PERIOD:** The value used as the decimal point for numeric constants used in SQL statements is a period.

***COMMA:** The value used as the decimal point for numeric constants in SQL statements is a comma.

Note: Any numeric constants in lists (such as in the SELECT clause, VALUES clause, and so on.) must be separated by a comma followed by a blank. For example, VALUES(1,1, 2,23, 4,1) is equivalent to VALUES(1.1,2.23,4.1) where the decimal point is a period.

Element 5: Naming Convention Options

***SYS:** The system naming convention (library-name/file-name) is used.

***SQL:** The SQL naming convention is used (schema-name.table-name). When creating a program on a remote database other than an iSeries system, *SQL must be specified as the naming convention.

Element 6: Second-Level Message Text Option

***NOSECLVL:** Second-level text descriptions are not added to the listing.

***SECLVL:** Second-level text with replacement data is added for all messages on the listing.

Element 7: Source Sequence Number Option

***NOSEQSRC:** Source sequence numbers from the input source files are used when creating the new source member in QSQLTEMP.

***SEQSRC:** Source records written to the new source member in QSQLTEMP are numbered starting at 000001.

Element 8: Debug Listing View Option

***NOLSTDBG:** Error and debug information is not generated.

***LSTDBG:** The SQL precompiler generates a listing view and error and debug information required for this view. You can use *LSTDBG only if you are using the CODE/400 product to compile your program.

TGTRLS

Specifies the release of the operating system on which the user intends to use the object being created.

In the examples given for the *CURRENT and *PRV values, and when specifying the *release-level* value, the format VxRxMx is used to specify the release, where Vx is the version, Rx is the release, and Mx is the modification level. For example, V2R3M0 is version 2, release 3, modification level 0.

***CURRENT:** The object is to be used on the release of the operating system currently running on the user's system. For example, if V2R3M5 is running on the system, *CURRENT means the user intends to use the object on a system with V2R3M5 installed. The user can also use the object on a system with any subsequent release of the operating system installed.

Note: If V2R3M5 is running on the system, and the object is to be used on a system with V2R3M0 installed, specify TGTRLS(V2R3M0) not TGTRLS(*CURRENT).

***PRV:** The object is to be used on the previous release with modification level 0 of the operating system. For example, if V2R3M5 is running on the user's system, *PRV means the user intends to use the object on a system with V2R2M0 installed. The user can also use the object on a system with any subsequent release of the operating system installed.

release-level: Specify the release in the format VxRxMx. The object can be used on a system with the specified release or with any subsequent release of the operating system installed.

Valid values depend on the current version, release, and modification level, and they change with each new release. If you specify a release-level which is earlier than the earliest release level supported by this command, an error message is sent indicating the earliest supported release.

INCFILE

Specifies the qualified name of the source file that contains members included in the program with any SQL INCLUDE statement.

The name of the source file can be qualified by one of the following library values:

***LIBL:** All libraries in the job's library list are searched until the first match is found.

***CURLIB:** The current library for the job is searched. If no library is specified as the current library for the job, the QGPL library is used.

library-name: Specify the name of the library to be searched.

***SRCFILE:** The qualified source file specified in the SRCFILE parameter contains the source file members specified on any SQL INCLUDE statement.

source-file-name: Specify the name of the source file that contains the source file members specified on any SQL INCLUDE statement. The record length of the source file specified here must be no less than the record length of the source file specified for the SRCFILE parameter.

COMMIT
Specifies whether SQL statements in the compiled program are run under commitment control. Files referred to in the host language source are not affected by this option. Only SQL tables, SQL views, and SQL packages referred to in SQL statements are affected.

Note: Files referenced in the RPG source are not affected by this option.

***CHG or *UR:** Specifies the objects referred to in SQL ALTER, CALL, COMMENT ON, CREATE, DROP, GRANT, LABEL ON, RENAME, and REVOKE statements and the rows updated, deleted, and inserted are locked until the end of the unit of work (transaction). Uncommitted changes in other jobs can be seen.

***ALL or *RS:** Specifies the objects referred to in SQL ALTER, CALL, COMMENT ON, CREATE, DROP, GRANT, LABEL ON, RENAME, and REVOKE statements and the rows selected, updated, deleted, and inserted are locked until the end of the unit of work (transaction). Uncommitted changes in other jobs cannot be seen.

***CS:** Specifies the objects referred to in SQL ALTER, CALL, COMMENT ON, CREATE, DROP, GRANT, LABEL ON, RENAME, and REVOKE statements and the rows updated, deleted, and inserted are locked until the end of the unit of work (transaction). A row that is selected, but not updated, is locked until the next row is selected. Uncommitted changes in other jobs cannot be seen.

***NONE or *NC:** Specifies that commitment control is not used. Uncommitted changes in other jobs can be seen. If the SQL DROP SCHEMA statement is included in the program, *NONE or *NC must be used. If a relational database is specified on the RDB parameter and the relational database is on a system that is not on an iSeries, *NONE or *NC cannot be specified.

***RR:** Specifies the objects referred to in SQL ALTER, CALL, COMMENT ON, CREATE, DROP, GRANT, LABEL ON, RENAME, and REVOKE statements and the rows selected, updated, deleted, and inserted are locked until the end of the unit of work (transaction). Uncommitted changes in other jobs cannot be seen. All tables referred to in SELECT, UPDATE, DELETE, and INSERT statements are locked exclusively until the end of the unit of work (transaction).

CLOSQLCSR
Specifies when SQL cursors are implicitly closed, SQL prepared statements are implicitly discarded, and LOCK TABLE locks are released. SQL cursors are explicitly closed when you issue the CLOSE, COMMIT, or ROLLBACK (without HOLD) SQL statements.

***ENDPGM:** SQL cursors are closed and SQL prepared statements are discarded when the program ends. LOCK TABLE locks are released when the first SQL program on the call stack ends.

***ENDSQL:** SQL cursors remain open between calls and can be fetched without running another SQL OPEN. One of the programs higher on the call stack must have run at least one SQL statement. SQL cursors are closed, SQL prepared statements are discarded, and LOCK TABLE locks are released when the first SQL program on the call stack ends. If *ENDSQL is specified for a program that is the first SQL program called (the first SQL program on the call stack), the program is treated as if *ENDPGM was specified.

***ENDJOB:** SQL cursors remain open between calls and can be fetched without running another SQL OPEN. One of the programs higher on the call stack must have run at least one SQL statement. SQL cursors are closed, SQL prepared statements are discarded, and LOCK TABLE locks are released when the first SQL program on the call stack ends. If *ENDSQL is specified for a program that is the first SQL program called (the first SQL program on the call stack), the program is treated as if *ENDPGM was specified.

ALWCPYDTA

Specifies whether a copy of the data can be used in a SELECT statement.

***OPTIMIZE:** The system determines whether to use the data retrieved directly from the database or to use a copy of the data. The decision is based on which method provides the best performance. If COMMIT is *CHG or *CS and ALWBLK is not *ALLREAD, or if COMMIT is *ALL or *RR, then a copy of the data is used only when it is necessary to run a query.

***YES:** A copy of the data is used only when necessary.

***NO:** A copy of the data is not allowed. If a temporary copy of the data is required to perform the query, an error message is returned.

ALWBLK

Specifies whether the database manager can use record blocking, and the extent to which blocking can be used for read-only cursors.

***ALLREAD:** Rows are blocked for read-only cursors if *NONE or *CHG is specified on the COMMIT parameter. All cursors in a program that are not explicitly able to be updated are opened for read-only processing even though EXECUTE or EXECUTE IMMEDIATE statements may be in the program.

Specifying *ALLREAD:

- Allows record blocking under commitment control level *CHG in addition to the blocking allowed for *READ.
- Can improve the performance of almost all read-only cursors in programs, but limits queries in the following ways:
  - The Rollback (ROLLBACK) command, a ROLLBACK statement in host languages, or the ROLLBACK HOLD SQL statement does not reposition a read-only cursor when *ALLREAD is specified.
  - Dynamic running of a positioned UPDATE or DELETE statement (for example, using EXECUTE IMMEDIATE), cannot be used to update a row in a cursor unless the DECLARE statement for the cursor includes the FOR UPDATE clause.

***NONE:** Rows are not blocked for retrieval of data for cursors.

Specifying *NONE:

- Guarantees that the data retrieved is current.
- May reduce the amount of time required to retrieve the first row of data for a query.
- Stops the database manager from retrieving a block of data rows that is not used by the program when only the first few rows of a query are retrieved before the query is closed.
- Can degrade the overall performance of a query that retrieves a large number of rows.

***READ:** Records are blocked for read-only retrieval of data for cursors when:

- *NONE is specified on the COMMIT parameter, which indicates that commitment control is not used.
- The cursor is declared with a FOR READ ONLY clause or there are no dynamic statements that could run a positioned UPDATE or DELETE statement for the cursor.

Specifying *READ can improve the overall performance of queries that meet the above conditions and retrieve a large number of records.

DLYPRP

Specifies whether the dynamic statement validation for a PREPARE statement is delayed until an OPEN, EXECUTE, or DESCRIBE statement is run. Delaying validation improves performance by eliminating redundant validation.

***NO:** Dynamic statement validation is not delayed. When the dynamic statement is prepared, the access plan is validated. When the dynamic statement is used in an OPEN or EXECUTE statement, the access plan is revalidated. Because the authority or the existence of objects referred to by the dynamic statement may change, you must still check the SQLCODE or SQLSTATE after issuing the OPEN or EXECUTE statement to ensure that the dynamic statement is still valid.

***YES:** Dynamic statement validation is delayed until the dynamic statement is used in an OPEN, EXECUTE, or DESCRIBE SQL statement. When the dynamic statement is used, the validation is completed and an access plan is built. If you specify *YES on this parameter, you should check the SQLCODE and SQLSTATE after running an OPEN, EXECUTE, or DESCRIBE statement to ensure that the dynamic statement is valid.

Note: If you specify *YES, performance is not improved if the INTO clause is used on the PREPARE statement or if a DESCRIBE statement uses the dynamic statement before an OPEN is issued for the statement.

GENLVL

Specifies the severity level at which the create operation fails. If errors occur that have a severity level greater than or equal to this value, the operation ends.

10: The default severity level is 10.

severity-level: Specify a value ranging from 0 through 40.

DATFMT

Specifies the format used when accessing date result columns. All output date fields are returned in the specified format. For input date strings, the specified value is used to determine whether the date is specified in a valid format.

Note: An input date string that uses the format *USA, *ISO, *EUR, or *JIS is always valid.

If a relational database is specified on the RDB parameter and the database is on a system that is not an iSeries system, then *USA, *ISO, *EUR, or *JIS must be specified.

***JOB:** The format specified for the job is used. Use the Display Job (DSPJOB) command to determine the current date format for the job.

***USA:** The United States date format (mm/dd/yyyy) is used.

***ISO:** The International Organization for Standardization (ISO) date format (yyyy-mm-dd) is used.

***EUR:** The European date format (dd.mm.yyyy) is used.

***JIS:** The Japanese Industrial Standard date format (yyyy-mm-dd) is used.

***MDY:** The date format (mm/dd/yy) is used.

***DMY:** The date format (dd/mm/yy) is used.

***YMD:** The date format (yy/mm/dd) is used.

***JUL:** The Julian date format (yy/ddd) is used.

DATSEP

Specifies the separator used when accessing date result columns.

Note: This parameter applies only when *JOB, *MDY, *DMY, *YMD, or *JUL is specified on the DATFMT parameter.

***JOB:** The date separator specified for the job at precompile time is used. Use the Display Job (DSPJOB) command to determine the current value for the job.

'/': A slash (/) is used.

'.': A period (.) is used.

',': A comma (,) is used.

'-': A dash (-) is used.

' ': A blank ( ) is used.

***BLANK:** A blank ( ) is used.

TIMFMT

Specifies the format used when accessing time result columns. For input time strings, the specified value is used to determine whether the time is specified in a valid format.

Note: An input date string that uses the format *USA, *ISO, *EUR, or *JIS is always valid.

If a relational database is specified on the RDB parameter and the database is on a system that is not another iSeries system, the time format must be *USA, *ISO, *EUR, *JIS, or *HMS with a time separator of colon or period.

***HMS:** The (hh:mm:ss) format is used.

***USA:** The United States time format (hh:mm *xx*) is used, where *xx* is AM or PM.

***ISO:** The International Organization for Standardization (ISO) time format (hh.mm.ss) is used.

***EUR:** The European time format (hh.mm.ss) is used.

***JIS:** The Japanese Industrial Standard time format (hh:mm:ss) is used.

TIMSEP
Specifies the separator used when accessing time result columns.

Note: This parameter applies only when *HMS is specified on the TIMFMT parameter.

***JOB:** The time separator specified for the job at precompile time is used. Use the Display Job (DSPJOB) command to determine the current value for the job.

':': A colon (:) is used.

'.': A period (.) is used.

',': A comma (,) is used.

' ': A blank ( ) is used.

***BLANK:** A blank ( ) is used.

REPLACE
Specifies whether a new SQL package is created when there is an existing SQL package of the same name in the same library. The value of this parameter is passed to the C command.More information on this parameter is in Appendix A, "Expanded Parameter Descriptions" in the CL Reference book.

***YES:** A new program or SQL package is created, and any existing program or SQL package of the same name and type in the specified library is moved to QRPLOBJ.

***NO:** A new program or SQL package is not created if an object of the same name and type already exists in the specified library.

RDB
Specifies the name of the relational database where the SQL package object is created.

***LOCAL:** The program is created as a distributed SQL program. The SQL statements will access the local database. An SQL package object is not created as part of the precompile process. The Create Structured Query Language Package (CRTSQLPKG) command can be used.

relational-database-name: Specify the name of the relational database where the new SQL package object is to be created. When the name of the local relational database is specified, the program created is still a distributed SQL program. The SQL statements will access the local database.

***NONE:** An SQL package object is not created. The program object is not a distributed program and the Create Structured Query Language Package (CRTSQLPKG) command cannot be used.

USER
Specifies the user name sent to the remote system when starting the conversation. This parameter is valid only when RDB is specified.

***CURRENT:** The user profile under which the current job is running is used.

user-name: Specify the user name being used for the application requester job.

PASSWORD
Specifies the password to be used on the remote system. This parameter is valid only if RDB is specified.

***NONE:** No password is sent. If this value is specified, USER(*CURRENT) must also be specified.

password: Specify the password of the user name specified on the USER parameter.

RDBCNNMTH
Specifies the semantics used for CONNECT statements. Refer to the SQL Reference book for more information.

***DUW:** CONNECT (Type 2) semantics are used to support distributed unit of work. Consecutive CONNECT statements to additional relational databases do not result in disconnection of previous connections.

***RUW:** CONNECT (Type 1) semantics are used to support remote unit of work. Consecutive CONNECT statements result in the previous connection being disconnected before a new connection is established.

DFTRDBCOL
Specifies the schema name used for the unqualified names of tables, views, indexes, and SQL packages. This parameter applies only to static SQL statements.

***NONE:** The naming convention defined on the OPTION parameter is used.

schema-name: Specify the name of the schema identifier. This value is used instead of the naming convention specified on the OPTION parameter.

DYNDFTCOL
Specifies whether the default schema name specified for the DFTRDBCOL parameter is also used for dynamic statements.

***NO:** Do not use the value specified on the DFTRDBCOL parameter for unqualified names of tables, views, indexes, and SQL packages for dynamic SQL statements. The naming convention specified on the OPTION parameter is used.

***YES:** The schema name specified on the DFTRDBCOL parameter will be used for the unqualified names of the tables, views, indexes, and SQL packages in dynamic SQL statements.

SQLPKG
Specifies the qualified name of the SQL package created on the relational database specified on the RDB parameter of this command.

The possible library values are:

***PGMLIB:** The package is created in the library with the same name as the library containing the program.

library-name: Specify the name of the library where the package is created.

***PGM:** The package name is the same as the program name.

package-name: Specify the name of the package created on the remote database specified on the RDBNAME parameter.

SQLPATH
Specifies the path to be used to find procedures, functions, and user defined types in static SQL statements.

***NAMING:** The path used depends on the naming convention specified on the OPTION parameter.

For *SYS naming, the path used is *LIBL, the current library list at runtime.

For *SQL naming, the path used is "QSYS", "QSYS2", "userid", where "userid" is the value of the USER special register. If a schema-name is specified on the DFTRDBCOL parameter, the schema-name takes the place of userid.

***LIBL:** The path used is the library list at runtime.

| *schema-name:* Specify a list of one or more schema names. A maximum of 268 individual schemas may be specified.

SQLCURRULE
Specifies the semantics used for SQL statements.

***DB2:** The semantics of all SQL statements will default to the rules established for DB2. The following semantics are controlled by this option:

- Hexadecimal constants are treated as character data.

***STD:** The semantics of all SQL statements will default to the rules established by the ISO and ANSI SQL standards. The following semantics are controlled by this option:

- Hexadecimal constants are treated as binary data.

SAAFLAG
Specifies the IBM SQL flagging function. This parameter flags SQL statements to verify whether they conform to IBM SQL syntax. More information about which IBM database products IBM SQL syntax is in the *DRDA IBM SQL Reference*, SC26-3255-00.

***NOFLAG:** The precompiler does not check to see whether SQL statements conform to IBM SQL syntax.

***FLAG:** The precompiler checks to see whether SQL statements conform to IBM SQL syntax.

FLAGSTD
Specifies the American National Standards Institute (ANSI) flagging function. This parameter flags SQL statements to verify whether they conform to the following standards.

```
ANSI X3.135-1992 entry
ISO 9075-1992 entry
FIPS 127.2 entry
```

***NONE:** The precompiler does not check to see whether SQL statements conform to ANSI standards.

***ANS:** The precompiler checks to see whether SQL statements conform to ANSI standards.

PRTFILE
Specifies the qualified name of the printer device file to which the listing is directed. The file must have a minimum record length of 132 bytes or information is lost.

The name of the printer file can be qualified by one of the following library values:

***LIBL:** All libraries in the job's library list are searched until the first match is found.

***CURLIB:** The current library for the job is searched. If no library is specified as the current library for the job, the QGPL library is used.

library-name: Specify the name of the printer device file to which the compiler printout is directed.

QSYSPRT: If a file name is not specified, the precompiler printout is directed to the IBM-supplied printer file QSYSPRT.

printer-file-name: Specify the name of the printer device file to which the compiler printout is directed.

SRTSEQ

Specifies the sort sequence table to be used for string comparisons in SQL statements.

Note: *HEX must be specified for this parameter on distributed applications where the application server is not on an iSeries system or the release level is prior to V2R3M0.

***JOB:** The SRTSEQ value for the job is retrieved during the precompile.

***JOBRUN:** The SRTSEQ value for the job is retrieved when the program is run. For distributed applications, SRTSEQ(*JOBRUN) is valid only when LANGID(*JOBRUN) is also specified.

***LANGIDUNQ:** The unique-weight sort table for the language specified on the LANGID parameter is used.

***LANGIDSHR:** The shared-weight sort table for the language specified on the LANGID parameter is used.

***HEX:** A sort sequence table is not used. The hexadecimal values of the characters are used to determine the sort sequence.

The name of the sort sequence table can be qualified by one of the following library values:

***LIBL:** All libraries in the job's library list are searched until the first match is found.

***CURLIB:** The current library for the job is searched. If no library is specified as the current library for the job, the QGPL library is used.

library-name: Specify the name of the library to be searched.

table-name: Specify the name of the sort sequence table to be used.

LANGID

Specifies the language identifier to be used when SRTSEQ(*LANGIDUNQ) or SRTSEQ(*LANGIDSHR) is specified.

***JOB:** The LANGID value for the job is retrieved during the precompile.

***JOBRUN:** The LANGID value for the job is retrieved when the program is run. For distributed applications, LANGID(*JOBRUN) is valid only when SRTSEQ(*JOBRUN) is also specified.

language-id: Specify a language identifier to be used by the program.

USRPRF

Specifies the user profile that is used when the compiled program object is run, including the authority that the program object has for each object in static SQL statements. The profile of either the program owner or the program user is used to control which objects can be used by the program object.

***NAMING:** The user profile is determined by the naming convention. If the naming convention is *SQL, USRPRF(*OWNER) is used. If the naming convention is *SYS, USRPRF(*USER) is used.

***USER:** The profile of the user running the program object is used.

***OWNER:** The user profiles of both the program owner and the program user are used when the program is run.

DYNUSRPRF

Specifies the user profile used for dynamic SQL statements.

***USER:** Local dynamic SQL statements are run under the user profile of the job. Distributed dynamic SQL statements are run under the user profile of the application server job.

***OWNER:** Local dynamic SQL statements are run under the user profile of the program's owner. Distributed dynamic SQL statements are run under the user profile of the SQL package's owner.

TOSRCFILE

Specifies the qualified name of the source file that is to contain the output source member that has been processed by the SQL precompiler. If the specified source file is not found, it will be created. The output member will have the same name as the name that is specified for the SRCMBR parameter.

The possible library values are:

QTEMP: The library QTEMP will be used.

***LIBL:** The job's library list is searched for the specified file. If the file is not found in any library in the library list, the file will be created in the current library.

***CURLIB:** The current library for the job will be used. If no library is specified as the current library for the job, the QGPL library will be used.

library-name: Specify the name of the library that is to contain the output source file.

QSQLTEMP: The source file QSQLTEMP will be used.

source-file-name: Specify the name of the source file to contain the output source member.

TEXT

Specifies text that briefly describes the program and its function. More information about this parameter is in the TEXT parameter topic in the CL Reference section of the Information Center.

***SRCMBRTXT:** The text is taken from the source file member being used to create the RPG program. Text for a database source member can be added or changed by using the Start Source Entry Utility (STRSEU) command, or by using either the Add Physical File Member (ADDPFM) command or the Change Physical File Member (CHGPFM) command. If the source file is an inline file or a device file, the text is blank.

***BLANK:** Text is not specified.

'description': Specify no more than 50 characters of text, enclosed in apostrophes.

Example:

```
CRTSQLRPG   PGM(JONES/ARBR5)
  TEXT('Accounts Receivable Branch 5')
```

This command runs the SQL precompiler which precompiles the source and stores the changed source in member ARBR5 in file QSQLTEMP in library QTEMP. The RPG compiler is called to create program ARBR5 in library JONES by using the source member created by the SQL precompiler.

CRTSQLRPGI (Create SQL ILE RPG Object) Command

Job: B,I   Pgm: B,I   REXX: B,I   Exec

```
               ┌─*CURLIB/──────┐
►►──CRTSQLRPGI──OBJ(──┼───────────────┼──object-name──)──────────────►
               └─library-name/─┘

►──┬──────────────────────────────────────────────────┬──────────────►
   │          ┌─*LIBL/────────┐ ┌─QRPGLESRC────────┐    │
   └─SRCFILE(─┼───────────────┼─┴─source-file-name─┴─)─┘
              ├─*CURLIB/──────┤
              └─library-name/─┘

                                                    (1)
►──┬──────────────────────────────────────────┬──────────────────────►
   │         ┌─*OBJ───────────────────┐        │
   └─SRCMBR(─┴─source-file-member-name─┴─)─┘

►──┬─────────────────────────────┬──┬─────────────────────────┬──────►
   └─OPTION(─┤ OPTION Details ├─)─┘  │          ┌─*CURRENT─┐    │
                                     └─TGTRLS(─┼─*PRV─────┼─)─┘
                                                └─VxRxMx───┘

►──┬─────────────────────────┬───────────────────────────────────────►
   │          ┌─*PGM─────┐    │
   └─OBJTYPE(─┼─*MODULE──┼─)─┘
              └─*SRVPGM──┘

►──┬──────────────────────────────────────────────────┬──────────────►
   │          ┌─*LIBL/────────┐ ┌─*SRCFILE─────────┐    │
   └─INCFILE(─┼───────────────┼─┴─source-file-name─┴─)─┘
              ├─*CURLIB/──────┤
              └─library-name/─┘
```

CRTSQLRPGI

```
►─┬─────────────────────────┬─┬──────────────────────────────────┬─►
  │            ┌─*UR───┐    │ │            ┌─*ENDACTGRP─┐        │
  │            ├─*CHG──┤    │ └─CLOSQLCSR(─┴─*ENDMOD────┴─)──────┘
  └─COMMIT(────┼─*ALL──┼─)──┘
               ├─*RS───┤
               ├─*CS───┤
               ├─*NONE─┤
               ├─*NC───┤
               └─*RR───┘

►─┬──────────────────────────────────┬─┬───────────────────────────┬─►
  │              ┌─*OPTIMIZE─┐       │ │          ┌─*ALLREAD─┐     │
  └─ALWCPYDTA(───┼─*YES──────┼─)─────┘ └─ALWBLK(──┼─*NONE────┼─)──┘
                 └─*NO───────┘                    └─*READ────┘

►─┬──────────────────────┬─┬────────────────────────────────┬─►
  │          ┌─*NO──┐    │ │          ┌─10─────────────┐    │
  └─DLYPRP(──┴─*YES─┴─)──┘ └─GENLVL(──┴─severity-level─┴─)──┘

►─┬──────────────────────┬─┬────────────────────────┬─►
  │           ┌─*JOB─┐   │ │          ┌─*JOB───┐    │
  └─DATFMT(───┼─*USA─┼─)─┘ └─DATSEP(──┼─'/'────┼─)──┘
              ├─*ISO─┤                ├─'.'────┤
              ├─*EUR─┤                ├─','────┤
              ├─*JIS─┤                ├─'-'────┤
              ├─*MDY─┤                ├─' '────┤
              ├─*DMY─┤                └─*BLANK─┘
              ├─*YMD─┤
              └─*JUL─┘

►─┬──────────────────────┬─┬────────────────────────┬─►
  │           ┌─*HMS─┐   │ │          ┌─*JOB───┐    │
  └─TIMFMT(───┼─*USA─┼─)─┘ └─TIMSEP(──┼─':'────┼─)──┘
              ├─*ISO─┤                ├─'.'────┤
              ├─*EUR─┤                ├─','────┤
              └─*JIS─┘                ├─' '────┤
                                      └─*BLANK─┘

►─┬────────────────────────┬─┬────────────────────────────────────────┬─►
  │            ┌─*YES─┐    │ │       ┌─*LOCAL───────────────────┐     │
  └─REPLACE(───┴─*NO──┴─)──┘ └─RDB(──┼─relational-database-name─┼─)──┘
                                     └─*NONE────────────────────┘

►─┬──────────────────────────┬─┬─────────────────────────────┬─►
  │        ┌─*CURRENT──┐     │ │            ┌─*NONE────┐     │
  └─USER(──┴─user-name─┴─)───┘ └─PASSWORD(──┴─password─┴─)───┘

►─┬───────────────────────────┬─┬─────────────────────────────────────┬─►
  │              ┌─*DUW─┐     │ │              ┌─*NONE───────────┐    │
  └─RDBCNNMTH(───┴─*RUW─┴─)───┘ └─DFTRDBCOL(───┴─collection-name─┴─)──┘
```

```
►─┬──────────────────────────────┬──────────────────────────────────────────────────►
  │              ┌─*NO──┐        │
  └─DYNDFTCOL(───┴─*YES─┴──)─────┘

►─┬───────────────────────────────────────────────────────────────┬─────────────────►
  │         ┌─*OBJLIB/───────┐  ┌─*OBJ─────────┐                │
  └─SQLPKG(─┼────────────────┼──┴─package-name─┴──)─────────────┘
            └─library-name/──┘

►─┬─────────────────────────────────────────┬──┬─────────────────────────────┬──────►
  │          ┌─*NAMING──────────────┐       │  │               ┌─*DB2─┐      │
  └─SQLPATH(─┼─*LIBL────────────────┼──)────┘  └─SQLCURRULE(───┴─*STD─┴──)───┘
             │  ┌──────────────────┐│
             │  ▼                  ││
             └───collection-name───┴┘

►─┬────────────────────────────────┬──┬────────────────────────────┬────────────────►
  │          ┌─*NOFLAG─┐           │  │           ┌─*NONE─┐        │
  └─SAAFLAG(─┴─*FLAG───┴──)────────┘  └─FLAGSTD(──┴─*ANS──┴──)─────┘

►─┬────────────────────────────────┬──┬──────────────────────────────┬──────────────►
  │          ┌─*NONE───┐           │  │          ┌─*NAMING─┐         │
  └─DBGVIEW(─┴─*SOURCE─┴──)────────┘  └─USRPRF(──┼─*OWNER──┼──)──────┘
                                                 └─*USER───┘

►─┬──────────────────────────────────┬──────────────────────────────────────────────►
  │              ┌─*USER──┐          │
  └─DYNUSRPRF(───┴─*OWNER─┴──)───────┘

►─┬──────────────────────────────────────────────────┬──────────────────────────────►
  │         ┌─*JOB─────────────────────────────────┐ │
  └─SRTSEQ(─┼─*JOBRUN──────────────────────────────┼──)─┘
            ├─*LANGIDUNQ───────────────────────────┤
            ├─*LANGIDSHR───────────────────────────┤
            ├─*HEX─────────────────────────────────┤
            │  ┌─*LIBL/─────────┐                  │
            └──┼────────────────┼──table-name──────┘
               ├─*CURLIB/───────┤
               └─library-name/──┘

►─┬──────────────────────────────────────────────┬──┬───────────────────────────┬──►
  │         ┌─*JOB──────────────────┐            │  │          ┌─*NONE──┐       │
  └─LANGID(─┼─*JOBRUN───────────────┼──)─────────┘  └─OUTPUT(──┴─*PRINT─┴──)───┘
            └─language-identifier───┘

►─┬───────────────────────────────────────────────────────────────┬─────────────────►
  │          ┌─*LIBL/─────────┐  ┌─QSYSPRT────────────┐           │
  └─PRTFILE(─┼────────────────┼──┴─printer-file-name──┴──)────────┘
             ├─*CURLIB/───────┤
             └─library-name/──┘

►─┬────────────────────────────────────────────────────────────────┬────────────────►
  │             ┌─QTEMP/─────────┐  ┌─QSQLTEMP1────────┐           │
  └─TOSRCFILE(──┼────────────────┼──┴─source-file-name─┴──)────────┘
                ├─*LIBL/─────────┤
                ├─*CURLIB/───────┤
                └─library-name/──┘
```

```
►─┬─────────────────────────────────────┬──────────────────────────►◄
  │           ┌─*SRCMBRTXT───┐          │
  └─TEXT(─────┼─*BLANK───────┼─────)────┘
              └─'description'┘
```

OPTION Details:

```
  ┌─*XREF───┐  ┌─*GEN───┐  ┌─*JOB─────┐  ┌─*SYS─┐  ┌─*NOSECLVL─┐
├─┼─────────┼──┼────────┼──┼──────────┼──┼──────┼──┼───────────┼─────►
  └─*NOXREF─┘  └─*NOGEN─┘  ├─*SYSVAL──┤  └─*SQL─┘  └─*SECLVL───┘
                           ├─*PERIOD──┤
                           └─*COMMA───┘

  ┌─*NOSEQSRC─┐  ┌─*NOEVENTF─┐  ┌─*OPTLOB───┐
►─┼───────────┼──┼───────────┼──┼───────────┼──────────────────────┤
  └─*SEQSRC───┘  └─*EVENTF───┘  └─*NOOPTLOB─┘
```

Notes:

1 All parameters preceding this point can be specified in positional form.

Purpose:

The Create Structured Query Language ILE RPG Object (CRTSQLRPGI) command calls the Structured Query Language (SQL) precompiler which precompiles RPG source containing SQL statements, produces a temporary source member, and then optionally calls the ILE RPG compiler to create a module, create a program, or create a service program.

Parameters:

OBJ

Specifies the qualified name of the object being created.

***CURLIB:** The new object is created in the current library for the job. If no library is specified as the current library for the job, the QGPL library is used.

library-name: Specify the name of the library where the object is created.

object-name: Specify the name of the object being created.

SRCFILE

Specifies the qualified name of the source file that contains the RPG source with SQL statements.

The name of the source file can be qualified by one of the following library values:

***LIBL:** All libraries in the job’s library list are searched until the first match is found.

***CURLIB:** The current library for the job is searched. If no library is specified as the current library for the job, the QGPL library is used.

library-name: Specify the name of the library to be searched.

QRPGLESRC: If the source file name is not specified, the IBM-supplied source file QRPGLESRC contains the RPG source.

source-file-name: Specify the name of the source file that contains the RPG source.

SRCMBR

Specifies the name of the source file member that contains the RPG source. This parameter is specified only if the source file name in the SRCFILE parameter is a database file. If this parameter is not specified, the PGM name specified on the OBJ parameter is used.

***OBJ:** Specifies that the RPG source is in the member of the source file that has the same name as that specified on the OBJ parameter.

source-file-member-name: Specify the name of the member that contains the RPG source.

OPTION

Specifies whether one or more of the following options are used when the RPG source is precompiled. If an option is specified more than once, or if two options conflict, the last option specified is used.

Element 1: Cross-Reference Options

***XREF:** The precompiler cross-references items in the program to the statement numbers in the program that refer to those items.

***NOXREF:** The precompiler does not cross-reference names.

Element 2: Program Creation Options

***GEN:** The precompiler creates the object that is specified by the OBJTYPE parameter.

***NOGEN:** The precompiler does not call the RPG compiler, and a module, program, service program, or SQL package is not created.

Element 3: Decimal Point Options

***JOB:** The value used as the decimal point for numeric constants in SQL is the representation of decimal point specified for the job at precompile time.

***SYSVAL:** The value used as the decimal point for numeric constants in SQL statements is the QDECFMT system value.

Note: If QDECFMT specifies that the value used as the decimal point is a comma(,), any numeric constants in lists (such as in the SELECT clause or the VALUES clause) must be separated by a comma (,) followed by a blank ( ). For example, VALUES(1,1, 2,23, 4,1) is equivalent to VALUES(1.1,2.23,4.1) in which the decimal point is a period (.).

***PERIOD:** The value used as the decimal point for numeric constants in SQL statements is a period (.).

***COMMA:** The value used as the decimal point for numeric constants in SQL statements is a comma (,).

Note: Any numeric constants in lists (such as in the SELECT clause or the VALUES clause) must be separated by a comma (,) followed by a blank( ). For example, VALUES(1,1, 2,23, 4,1) is equivalent to VALUES(1.1,2.23,4.1) where the decimal point is a period (.).

Element 4: Naming Convention Options

***SYS:** The system naming convention (library-name/file-name) is used.

***SQL:** The SQL naming convention is used (schema-name.table-name). When creating a program on a remote database other than an iSeries system, *SQL must be specified as the naming convention.

Element 5: Second-Level Message Text Option

***NOSECLVL:** Second-level text descriptions are not added to the listing.

***SECLVL:** Second-level text with replacement data is added for all messages on the listing.

Element 6: Sequence source

***NOSEQSRC:** The source file member created into QSQLTEMP1 has the same sequence numbers as the original source read by the precompiler.

***SEQSRC:** The source file member created into QSQLTEMP1 contains sequence numbers starting at 000001 and incremented by 000001.

Element 7: Event File Creation

***NOEVENTF:** The compiler will not produce an Event File for use by CoOperative Development Environment/400 (CODE/400).

***EVENTF:** The compiler produces an event file for use by CoOperative Development Environment/400 (CODE/400). The event file will be created as a member in the file EVFEVENT in your source library. CODE/400 uses this file to offer error feedback integrated with the CODE/400 editor. This option is normally specified by CODE/400 on your behalf.

Element 8: Date Conversion

***NOCVTDT:** Date, time and timestamp data types which are retrieved from externally-described files are to be processed using the native RPG language.

***CVTDT:** Date, time and timestamp data types which are retrieved from externally-described files are to be processed as fixed-length character.

Element 9: Large Object Optimization for DRDA

***OPTLOB:** The first FETCH for a cursor determines how the cursor will be used for LOBs (Large Objects) on all subsequent FETCHes. This option remains in effect until the cursor is closed.

If the first FETCH uses a LOB locator to access a LOB column, no subsequent FETCH for that cursor can fetch that LOB column into a LOB host variable.

If the first FETCH places the LOB column into a LOB host variable, no subsequent FETCH for that cursor can use a LOB locator for that column.

***NOOPTLOB:** There is no restriction on whether a column is retrieved into a LOB locator or into a LOB host variable. This option can cause performance to degrade.

TGTRLS

Specifies the release of the operating system on which the user intends to use the object being created.

In the examples given for the *CURRENT and *PRV values, and when specifying the *release-level* value, the format VxRxMx is used to specify the release, where Vx is the version, Rx is the release, and Mx is the modification level. For example, V2R3M0 is version 2, release 3, modification level 0.

***CURRENT:** The object is to be used on the release of the operating system currently running on the user's system. For example, if V2R3M5 is running on the system, *CURRENT means the user intends to use the object on a system with V2R3M5 installed. The user can also use the object on a system with any subsequent release of the operating system installed.

Note: If V2R3M5 is running on the system, and the object is to be used on a system with V2R3M0 installed, specify TGTRLS(V2R3M0) not TGTRLS(*CURRENT).

***PRV:** The object is to be used on the previous release with modification level 0 of the operating system. For example, if V2R3M5 is running on the user's system, *PRV means the user intends to use the object on a system with V2R2M0 installed. The user can also use the object on a system with any subsequent release of the operating system installed.

release-level: Specify the release in the format VxRxMx. The object can be used on a system with the specified release or with any subsequent release of the operating system installed.

Valid values depend on the current version, release, and modification level, and they change with each new release. If you specify a release-level which is earlier than the earliest release level supported by this command, an error message is sent indicating the earliest supported release.

OBJTYPE

Specifies the type of object being created.

***PGM:** The SQL precompiler issues the CRTBNDRPG command to create the bound program.

***MODULE:** The SQL precompiler issues the CRTRPGMOD command to create the module.

***SRVPGM:** The SQL precompiler issues the CRTRPGMOD and CRTSRVPGM commands to create the service program.

Notes:

1. When OBJTYPE(*PGM) or OBJTYPE(*SRVPGM) is specified and the RDB parameter is also specified, the CRTSQLPKG command is issued by the SQL precompiler after the program has been created. When OBJTYPE(*MODULE) is specified, an SQL package is not created and you must issue the CRTSQLPKG command after the CRTPGM or CRTSRVPGM command has created the program.
2. If *NOGEN is specified, only the SQL temporary source member is generated and a module, program, service program, and SQL package are not created.

INCFILE

Specifies the qualified name of the source file that contains members included in the program with any SQL INCLUDE statement.

The name of the source file can be qualified by one of the following library values:

***LIBL:** All libraries in the job's library list are searched until the first match is found.

***CURLIB:** The current library for the job is searched. If no library is specified as the current library for the job, the QGPL library is used.

library-name: Specify the name of the library to be searched.

***SRCFILE:** The qualified source file specified in the SRCFILE parameter contains the source file members specified on any SQL INCLUDE statement.

source-file-name: Specify the name of the source file that contains the source file members specified on any SQL INCLUDE statement. The record length of the source file specified here must be no less than the record length of the source file specified on the SRCFILE parameter.

COMMIT

Specifies whether SQL statements in the compiled unit are run under commitment control. Files referred to in the host language source are not affected by this option. Only SQL tables, SQL views, and SQL packages referred to in SQL statements are affected.

***CHG or *UR:** Specifies the objects referred to in SQL ALTER, CALL, COMMENT ON, CREATE, DROP, GRANT, LABEL ON, RENAME, and REVOKE statements and the rows updated, deleted, and inserted are locked until the end of the unit of work (transaction). Uncommitted changes in other jobs can be seen.

***ALL or *RS:** Specifies the objects referred to in SQL ALTER, CALL, COMMENT ON, CREATE, DROP, GRANT, LABEL ON, RENAME, and REVOKE statements and the rows selected, updated, deleted, and inserted are locked until the end of the unit of work (transaction). Uncommitted changes in other jobs cannot be seen.

***CS:** Specifies the objects referred to in SQL ALTER, CALL, COMMENT ON, CREATE, DROP, GRANT, LABEL ON, RENAME, and REVOKE statements and the rows updated, deleted, and inserted are locked until the end of the unit of work (transaction). A row that is selected, but not updated, is locked until the next row is selected. Uncommitted changes in other jobs cannot be seen.

***NONE or *NC:** Specifies that commitment control is not used. Uncommitted changes in other jobs can be seen. If the SQL DROP SCHEMA statement is included in the program, *NONE or *NC must be used. If a relational database is specified on the RDB parameter and the relational database is on a system that is not on an iSeries, *NONE or *NC cannot be specified.

***RR:** Specifies the objects referred to in SQL ALTER, CALL, COMMENT ON, CREATE, DROP, GRANT, LABEL ON, RENAME, and REVOKE statements and the rows selected, updated, deleted, and inserted are locked until the end of the unit of work (transaction). Uncommitted changes in other jobs cannot be seen. All tables referred to in SELECT, UPDATE, DELETE, and INSERT statements are locked exclusively until the end of the unit of work (transaction).

CLOSQLCSR

Specifies when SQL cursors are implicitly closed, SQL prepared statements are implicitly discarded, and LOCK TABLE locks are released. SQL cursors are explicitly closed when you issue the CLOSE, COMMIT, or ROLLBACK (without HOLD) SQL statements.

***ENDACTGRP:** SQL cursors are closed, SQL prepared statements are implicitly discarded, and LOCK TABLE locks are released when the activation group ends.

***ENDMOD:** SQL cursors are closed and SQL prepared statements are implicitly discarded when the module is exited. LOCK TABLE locks are released when the first SQL program on the call stack ends.

ALWCPYDTA

Specifies whether a copy of the data can be used in a SELECT statement.

***OPTIMIZE:** The system determines whether to use the data retrieved directly from the database or to use a copy of the data. The decision is based on which method provides the best performance. If COMMIT is *CHG or *CS and ALWBLK is not *ALLREAD, or if COMMIT is *ALL or *RR, then a copy of the data is used only when it is necessary to run a query.

***YES:** A copy of the data is used only when necessary.

***NO:** A copy of the data is not allowed. If a temporary copy of the data is required to perform the query, an error message is returned.

ALWBLK

Specifies whether the database manager can use record blocking, and the extent to which blocking can be used for read-only cursors.

***ALLREAD:** Rows are blocked for read-only cursors if *NONE or *CHG is specified on the COMMIT parameter. All cursors in a program that are not explicitly able to be updated are opened for read-only processing even though EXECUTE or EXECUTE IMMEDIATE statements may be in the program.

Specifying *ALLREAD:

- Allows record blocking under commitment control level *CHG in addition to the blocking allowed for *READ.
- Can improve the performance of almost all read-only cursors in programs, but limits queries in the following ways:
  - The Rollback (ROLLBACK) command, a ROLLBACK statement in host languages, or the ROLLBACK HOLD SQL statement does not reposition a read-only cursor when *ALLREAD is specified.
  - Dynamic running of a positioned UPDATE or DELETE statement (for example, using EXECUTE IMMEDIATE), cannot be used to update a row in a cursor unless the DECLARE statement for the cursor includes the FOR UPDATE clause.

***NONE:** Rows are not blocked for retrieval of data for cursors.

Specifying *NONE:

- Guarantees that the data retrieved is current.
- May reduce the amount of time required to retrieve the first row of data for a query.
- Stops the database manager from retrieving a block of data rows that is not used by the program when only the first few rows of a query are retrieved before the query is closed.
- Can degrade the overall performance of a query that retrieves a large number of rows.

***READ:** Records are blocked for read-only retrieval of data for cursors when:

- *NONE is specified on the COMMIT parameter, which indicates that commitment control is not used.
- The cursor is declared with a FOR READ ONLY clause or there are no dynamic statements that could run a positioned UPDATE or DELETE statement for the cursor.

Specifying *READ can improve the overall performance of queries that meet the above conditions and retrieve a large number of records.

DLYPRP
Specifies whether the dynamic statement validation for a PREPARE statement is delayed until an OPEN, EXECUTE, or DESCRIBE statement is run. Delaying validation improves performance by eliminating redundant validation.

***NO:** Dynamic statement validation is not delayed. When the dynamic statement is prepared, the access plan is validated. When the dynamic statement is used in an OPEN or EXECUTE statement, the access plan is revalidated. Because the authority or the existence of objects referred to by the dynamic statement may change, you must still check the SQLCODE or SQLSTATE after issuing the OPEN or EXECUTE statement to ensure that the dynamic statement is still valid.

***YES:** Dynamic statement validation is delayed until the dynamic statement is used in an OPEN, EXECUTE, or DESCRIBE SQL statement. When the dynamic statement is used, the validation is completed and an access plan is built. If you specify *YES on this parameter, you should check the SQLCODE and SQLSTATE after running an OPEN, EXECUTE, or DESCRIBE statement to ensure that the dynamic statement is valid.

Note: If you specify *YES, performance is not improved if the INTO clause is used on the PREPARE statement or if a DESCRIBE statement uses the dynamic statement before an OPEN is issued for the statement.

GENLVL
Specifies the severity level at which the create operation fails. If errors occur that have a severity level greater than this value, the operation ends.

10: The default severity level is 10.

severity-level: Specify a value ranging from 0 through 40.

DATFMT
Specifies the format used when accessing date result columns. All output date fields are returned in the specified format. For input date strings, the specified value is used to determine whether the date is specified in a valid format.

Note: An input date string that uses the format *USA, *ISO, *EUR, or *JIS is always valid.

If a relational database is specified on the RDB parameter and the database is on a system that is not an iSeries system, then *USA, *ISO, *EUR, or *JIS must be specified.

***JOB:** The format specified for the job is used. Use the Display Job (DSPJOB) command to determine the current date format for the job.

***USA:** The United States date format (mm/dd/yyyy) is used.

***ISO:** The International Organization for Standardization (ISO) date format (yyyy-mm-dd) is used.

***EUR:** The European date format (dd.mm.yyyy) is used.

***JIS:** The Japanese Industrial Standard date format (yyyy-mm-dd) is used.

***MDY:** The date format (mm/dd/yy) is used.

***DMY:** The date format (dd/mm/yy) is used.

***YMD:** The date format (yy/mm/dd) is used.

***JUL:** The Julian date format (yy/ddd) is used.

DATSEP

Specifies the separator used when accessing date result columns.

Note: This parameter applies only when *JOB, *MDY, *DMY, *YMD, or *JUL is specified on the DATFMT parameter.

***JOB:** The date separator specified for the job at precompile time is used. Use the Display Job (DSPJOB) command to determine the current value for the job.

'/': A slash (/) is used.

'.': A period (.) is used.

',': A comma (,) is used.

'-': A dash (-) is used.

' ': A blank ( ) is used.

***BLANK:** A blank ( ) is used.

TIMFMT

Specifies the format used when accessing time result columns. For input time strings, the specified value is used to determine whether the time is specified in a valid format.

Note: An input time string that uses the format *USA, *ISO, *EUR, or *JIS is always valid.

If a relational database is specified on the RDB parameter and the database is on a system that is not another iSeries system, the time format must be *USA, *ISO, *EUR, *JIS, or *HMS with a time separator of a colon or period.

***HMS:** The **hh:mm:ss** format is used.

***USA:** The United States time format **hh:mm xx** is used, where **xx** is AM or PM.

***ISO:** The International Organization for Standardization (ISO) time format **hh.mm.ss** is used.

***EUR:** The European time format **hh.mm.ss** is used.

***JIS:** The Japanese Industrial Standard time format **hh:mm:ss** is used.

TIMSEP
Specifies the separator used when accessing time result columns.

Note: This parameter applies only when *HMS is specified on the TIMFMT parameter.

***JOB:** The time separator specified for the job at precompile time is used. Use the Display Job (DSPJOB) command to determine the current value for the job.

':': A colon (:) is used.

'.': A period (.) is used.

',': A comma (,) is used.

' ': A blank ( ) is used.

***BLANK:** A blank ( ) is used.

REPLACE
Specifies if a SQL module, program, service program or package is created when there is an existing SQL module, program, service program, or package of the same name and type in the same library. The value of this parameter is passed to the CRTRPGMOD, CRTBNDRPG, CRTSRVPGM, and CRTSQLPKG commands.

***YES:** A new SQL module, program, service program, or package is created, any existing SQL object of the same name and type in the specified library is moved to QRPLOBJ.

***NO:** A new SQL module, program, service program, or package is not created if an SQL object of the same name and type already exists in the specified library.

RDB
Specifies the name of the relational database where the SQL package object is created.

***LOCAL:** The program is created as a distributed SQL program. The SQL statements will access the local database. An SQL package object is not created as part of the precompile process. The Create Structured Query Language Package (CRTSQLPKG) command can be used.

relational-database-name: Specify the name of the relational database where the new SQL package object is to be created. When the name of the local relational database is specified, the program created is still a distributed SQL program. The SQL statements will access the local database.

***NONE:** An SQL package object is not created. The program object is not a distributed program and the Create Structured Query Language Package (CRTSQLPKG) command cannot be used.

USER
Specifies the user name sent to the remote system when starting the conversation. This parameter is valid only when RDB is specified.

***CURRENT:** The user profile under which the current job is running is used.

user-name: Specify the user name being used for the application server job.

PASSWORD
Specifies the password to be used on the remote system. This parameter is valid only if RDB is specified.

***NONE:** No password is sent. If this value is specified, USER(*CURRENT) must also be specified.

password: Specify the password of the user name specified on the USER parameter.

RDBCNNMTH
Specifies the semantics used for CONNECT statements. Refer to the SQL Reference book for more information.

***DUW:** CONNECT (Type 2) semantics are used to support distributed unit of work. Consecutive CONNECT statements to additional relational databases do not result in disconnection of previous connections.

***RUW:** CONNECT (Type 1) semantics are used to support remote unit of work. Consecutive CONNECT statements result in the previous connection being disconnected before a new connection is established.

DFTRDBCOL
Specifies the schema name used for the unqualified names of tables, views, indexes, and SQL packages. This parameter applies only to static SQL statements.

***NONE:** The naming convention defined on the OPTION parameter is used.

schema-name: Specify the name of the schema identifier. This value is used instead of the naming convention specified on the OPTION parameter.

DYNDFTCOL
Specifies whether the default schema name specified for the DFTRDBCOL parameter is also used for dynamic statements.

***NO:** Do not use the value specified on the DFTRDBCOL parameter for unqualified names of tables, views, indexes, and SQL packages for dynamic SQL statements. The naming convention specified on the OPTION parameter is used.

***YES:** The schema name specified on the DFTRDBCOL parameter will be used for the unqualified names of the tables, views, indexes, and SQL packages in dynamic SQL statements.

SQLPKG
Specifies the qualified name of the SQL package created on the relational database specified on the RDB parameter of this command.

The possible library values are:

***OBJLIB:** The package is created in the library with the same name as the library specified on the OBJ parameter.

library-name: Specify the name of the library where the package is created.

***OBJ:** The name of the SQL package is the same as the object name specified on the OBJ parameter.

package-name: Specify the name of the SQL package. If the remote system is not an iSeries system, no more than 8 characters can be specified.

SQLPATH
Specifies the path to be used to find procedures, functions, and user defined types in static SQL statements.

***NAMING:** The path used depends on the naming convention specified on the OPTION parameter.

For *SYS naming, the path used is *LIBL, the current library list at runtime.

For *SQL naming, the path used is ″QSYS″, ″QSYS2″, ″userid″, where ″userid″ is the value of the USER special register. If a schema-name is specified on the DFTRDBCOL parameter, the schema-name takes the place of userid.

***LIBL:** The path used is the library list at runtime.

schema-name: Specify a list of one or more schema names. A maximum of 268 individual schemas may be specified.

SQLCURRULE
Specifies the semantics used for SQL statements.

***DB2:** The semantics of all SQL statements will default to the rules established for DB2. The following semantics are controlled by this option:

- Hexadecimal constants are treated as character data.

***STD:** The semantics of all SQL statements will default to the rules established by the ISO and ANSI SQL standards. The following semantics are controlled by this option:

- Hexadecimal constants are treated as binary data.

SAAFLAG
Specifies the IBM SQL flagging function. This parameter flags SQL statements to verify whether they conform to IBM SQL syntax. More information about IBM SQL syntax found in IBM database products can be found in the *DRDA IBM SQL Reference*, SC26–3255–00.

***NOFLAG:** The precompiler does not check to see whether SQL statements conform to IBM SQL syntax.

***FLAG:** The precompiler checks to see whether SQL statements conform to IBM SQL syntax.

FLAGSTD
Specifies the American National Standards Institute (ANSI) flagging function. This parameter flags SQL statements to verify whether they conform to the following standards.

```
ANSI X3.135-1992 entry
ISO 9075-1992 entry
FIPS 127.2 entry
```

***NONE:** The precompiler does not check to see whether SQL statements conform to ANSI standards.

***ANS:** The precompiler checks to see whether SQL statements conform to ANSI standards.

DBGVIEW
Specifies the type of source debug information to be provided by the SQL precompiler.

***NONE:** The source view will not be generated.

***SOURCE:** The SQL precompiler will provide the source views for the root and if necessary, SQL INCLUDE statements. A view will be provided which contains the statements generated by the precompiler.

USRPRF

Specifies the user profile that is used when the compiled program object is run, including the authority that the program object has for each object in static SQL statements. The profile of either the program owner or the program user is used to control which objects can be used by the program object.

***NAMING:** The user profile is determined by the naming convention. If the naming convention is *SQL, USRPRF(*OWNER) is used. If the naming convention is *SYS, USRPRF(*USER) is used.

***USER:** The profile of the user running the program object is used.

***OWNER:** The user profiles of both the program owner and the program user are used when the program is run.

DYNUSRPRF

Specifies the user profile to be used for dynamic SQL statements.

***USER:** For local, dynamic SQL statements run under the user of the program's user. For distributed, dynamic SQL statements run under the profile of the SQL package's user.

***OWNER:** For local, dynamic SQL statements run under the profile of the program's owner. For distributed, dynamic SQL statements run under the profile of the SQL package's owner.

SRTSEQ

Specifies the sort sequence table to be used for string comparisons in SQL statements.

Note: *HEX must be specified for this parameter on distributed applications where the application server is not on an iSeries system or the release level is prior to V2R3M0.

***JOB:** The SRTSEQ value for the job is retrieved during the precompile.

***JOBRUN:** The SRTSEQ value for the job is retrieved when the program is run. For distributed applications, SRTSEQ(*JOBRUN) is valid only when LANGID(*JOBRUN) is also specified.

***LANGIDUNQ:** The unique-weight sort table for the language specified on the LANGID parameter is used.

***LANGIDSHR:** The sort sequence table uses the same weight for multiple characters, and is the shared-weight sort sequence table associated with the language specified on the LANGID parameter.

***HEX:** A sort sequence table is not used. The hexadecimal values of the characters are used to determine the sort sequence.

The name of the sort sequence table can be qualified by one of the following library values:

***LIBL:** All libraries in the job's library list are searched until the first match is found.

***CURLIB:** The current library for the job is searched. If no library is specified as the current library for the job, the QGPL library is used.

library-name: Specify the name of the library to be searched.

table-name: Specify the name of the sort sequence table to be used.

LANGID

Specifies the language identifier to be used when SRTSEQ(*LANGIDUNQ) or SRTSEQ(*LANGIDSHR) is specified.

***JOB:** The LANGID value for the job is retrieved during the precompile.

***JOBRUN:** The LANGID value for the job is retrieved when the program is run. For distributed applications, LANGID(*JOBRUN) is valid only when SRTSEQ(*JOBRUN) is also specified.

language-identifier: Specify a language identifier.

OUTPUT

Specifies whether the precompiler listing is generated.

***NONE:** The precompiler listing is not generated.

***PRINT:** The precompiler listing is generated.

PRTFILE

Specifies the qualified name of the printer device file to which the precompiler printout is directed. The file must have a minimum length of 132 bytes. If a file with a record length of less than 132 bytes is specified, information is lost.

The name of the printer file can be qualified by one of the following library values:

***LIBL:** All libraries in the job's library list are searched until the first match is found.

***CURLIB:** The current library for the job is searched. If no library is specified as the current library for the job, the QGPL library is used.

library-name: Specify the name of the library to be searched.

QSYSPRT: If a file name is not specified, the precompiler printout is directed to the IBM-supplied printer file QSYSPRT.

printer-file-name: Specify the name of the printer device file to which the precompiler printout is directed.

TOSRCFILE

Specifies the qualified name of the source file that is to contain the output source member that has been processed by the SQL precompiler. If the specified source file is not found, it will be created. The output member will have the same name as the name that is specified for the SRCMBR parameter.

The possible library values are:

QTEMP: The library QTEMP will be used.

***LIBL:** The job's library list is searched for the specified file. If the file is not found in any library in the library list, the file will be created in the current library.

***CURLIB:** The current library for the job will be used. If no library is specified as the current library for the job, the QGPL library will be used.

library-name: Specify the name of the library that is to contain the output source file.

QSQLTEMP1: The source file QSQLTEMP1 will be used.

source-file-name: Specify the name of the source file to contain the output source member.

TEXT

Specifies the text that briefly describes the function. More information about this parameter is in the TEXT parameter topic in the CL Reference section of the Information Center.

***SRCMBRTXT:** The text is taken from the source file member being used to create the RPG program. Text can be added or changed for a database source member by using the Start Source Entry Utility (STRSEU) command, or by using either the Add Physical File Member (ADDPFM) or Change Physical File Member (CHGPFM) command. If the source file is an inline file or a device file, the text is blank.

***BLANK:** Text is not specified.

'description': Specify no more than 50 characters of text, enclosed in apostrophes.

Example:

```
CRTSQLRPGI  PAYROLL OBJTYPE(*PGM) TEXT('Payroll Program')
```

This command runs the SQL precompiler which precompiles the source and stores the changed source in member PAYROLL in file QSQLTEMP1 in library QTEMP. The ILE RPG compiler is called to create program PAYROLL in the current library by using the source member created by the SQL precompiler.

Appendix C. Using FORTRAN for iSeries Precompiler

This appendix contains the syntax diagrams for the FORTRAN for iSeries precompiler, although this compiler is no longer supported on the iSeries. Another appendix, Appendix D, “Coding SQL Statements in FORTRAN Applications” on page 315, describes the unique application and coding requirements for embedding SQL statements in a FORTRAN/400 program.

For more details, see “Using the FORTRAN/400 precompiler”.

Note: See “Code disclaimer information” on page viii information for information pertaining to code examples.

Using the FORTRAN/400 precompiler

FORTRAN/400 is no longer a supported compiler for the iSeries system. This appendix is intended to help those customers who are using the SQL FORTRAN precompiler with other non-IBM FORTRAN compilers. For a description of using the FORTRAN precompiler, see CRTSQLFTN (Create Structured Query Language FORTRAN) Command. For more information, see “CRTSQLFTN (Create Structured Query Language FORTRAN) Command”.

CRTSQLFTN (Create Structured Query Language FORTRAN) Command

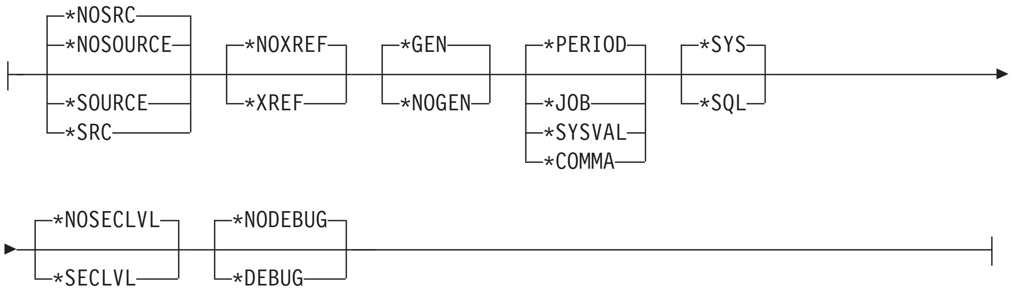
Job: B,I   Pgm: B,I   REXX: B,I   Exec

```
                         ┌─*CURLIB/──────┐
►►──CRTSQLFTN──PGM(──────┼───────────────┼──program-name──)──────────────────►
                         └─library-name/─┘

►──┬──────────────────────────────────────────────────────────────┬──────────►
   └─SRCFILE(──┬─*LIBL/────────┬──┬─QFTNSRC──────────┬──)─┘
               ├───────────────┤  └─source-file-name─┘
               ├─*CURLIB/──────┤
               └─library-name/─┘

                                                      (1)
►──┬─────────────────────────────────────────────────┬───────────────────────►
   └─SRCMBR(──┬─*PGM────────────────────┬──)─┘
              └─source-file-member-name─┘

►──┬──────────────────────────────┬──┬───────────────────────────────┬───────►
   └─OPTION(──┤ OPTION Details ├──)─┘  └─TGTRLS(──┬─*CURRENT─┬──)─┘
                                                  ├─*PRV─────┤
                                                  └─VxRxMx───┘
```

```
►─┬──────────────────────────────────────────────────────────────────────┬──►
  │              ┌─*LIBL/───────────┐  ┌─*SRCFILE──────────┐              │
  └─INCFILE(─────┼──────────────────┼──┴─source-file-name──┴──)──────────┘
                 ├─*CURLIB/─────────┤
                 └─library-name/────┘

►─┬──────────────────────────────┬─┬───────────────────────────────────┬──►
  │             ┌─*UR───┐        │ │               ┌─*ENDPGM─┐         │
  └─COMMIT(─────┼─*CHG──┼──)─────┘ └─CLOSQLCSR(─────┼─*ENDSQL─┼──)──────┘
                ├─*ALL──┤                          └─*ENDJOB─┘
                ├─*RS───┤
                ├─*CS───┤
                ├─*NONE─┤
                ├─*NC───┤
                └─*RR───┘

►─┬────────────────────────────────────┬─┬────────────────────────────────┬──►
  │              ┌─*YES──────┐         │ │             ┌─*READ────┐       │
  └─ALWCPYDTA(───┼─*OPTIMIZE─┼──)──────┘ └─ALWBLK(─────┼─*NONE────┼──)────┘
                 └─*NO───────┘                         └─*ALLREAD─┘

►─┬─────────────────────────────┬─┬───────────────────────────────────────┬──►
  │             ┌─*NO──┐        │ │            ┌─10─────────────┐         │
  └─DLYPRP(─────┴─*YES─┴──)─────┘ └─GENLVL(────┴─severity-level─┴──)──────┘

►─┬─────────────────────────────┬─┬──────────────────────────────┬──►
  │             ┌─*JOB─┐        │ │            ┌─*JOB───┐        │
  └─DATFMT(─────┼─*USA─┼──)─────┘ └─DATSEP(────┼─'/'────┼──)─────┘
                ├─*ISO─┤                       ├─'.'────┤
                ├─*EUR─┤                       ├─','────┤
                ├─*JIS─┤                       ├─'-'────┤
                ├─*MDY─┤                       ├─' '────┤
                ├─*DMY─┤                       └─*BLANK─┘
                ├─*YMD─┤
                └─*JUL─┘

►─┬─────────────────────────────┬─┬──────────────────────────────┬──►
  │             ┌─*HMS─┐        │ │            ┌─*JOB───┐        │
  └─TIMFMT(─────┼─*USA─┼──)─────┘ └─TIMSEP(────┼─':'────┼──)─────┘
                ├─*ISO─┤                       ├─'.'────┤
                ├─*EUR─┤                       ├─','────┤
                └─*JIS─┘                       ├─' '────┤
                                               └─*BLANK─┘

►─┬──────────────────────────────┬──────────────────────────────────►
  │              ┌─*YES─┐        │
  └─REPLACE(─────┴─*NO──┴──)─────┘

►─┬───────────────────────────────────────────────┬─┬────────────────────────────────┬──►
  │         ┌─*LOCAL─────────────────────┐        │ │         ┌─*CURRENT──┐          │
  └─RDB(────┼─relational-database-name───┼──)─────┘ └─USER(───┴─user-name─┴──)───────┘
            └─*NONE──────────────────────┘
```

```
►─┬──────────────────────────────────┬─┬─────────────────────────────┬─►
  │              ┌─*NONE────┐        │ │              ┌─*DUW─┐       │
  └─PASSWORD(────┴─password─┴───)────┘ └─RDBCNNMTH(───┴─*RUW─┴───)───┘

►─┬────────────────────────────────────────┬───────────────────────────►
  │               ┌─*NONE───────────┐      │
  └─DFTRDBCOL(────┴─collection-name─┴───)──┘

►─┬──────────────────────────────────────────────────────────┬─────────►
  │            ┌─*PGMLIB/───────┐ ┌─*PGM─────────┐            │
  └─SQLPKG(────┼────────────────┼─┴─package-name─┴───)───────┘
               └─library-name/──┘

►─┬──────────────────────────────────┬─┬───────────────────────────┬───►
  │             ┌─*NOFLAG─┐          │ │             ┌─*NONE─┐      │
  └─SAAFLAG(────┴─*FLAG───┴───)──────┘ └─FLAGSTD(────┴─*ANS──┴───)──┘

►─┬─────────────────────────────────────────────────────────────┬──────►
  │             ┌─*LIBL/─────────┐ ┌─QSYSPRT───────────┐         │
  └─PRTFILE(────┼────────────────┼─┴─printer-file-name─┴───)────┘
                ├─*CURLIB/───────┤
                └─library-name/──┘

►─┬─────────────────────────────────────────────────────────┬──────────►
  │            ┌─*JOB───────────────────────────────────┐    │
  └─SRTSEQ(────┼─*JOBRUN────────────────────────────────┼─)──┘
               ├─*LANGIDUNQ─────────────────────────────┤
               ├─*LANGIDSHR─────────────────────────────┤
               ├─*HEX───────────────────────────────────┤
               │ ┌─*LIBL/─────────┐                      │
               └─┼────────────────┼─table-name───────────┘
                 ├─*CURLIB/───────┤
                 └─library-name/──┘

►─┬─────────────────────────────────┬─┬─────────────────────────────┬──►
  │            ┌─*JOB─────────┐     │ │             ┌─*NAMING─┐      │
  └─LANGID(────┼─*JOBRUN──────┼─)───┘ └─USRPRF(─────┼─*OWNER──┼──)───┘
               └─language-ID──┘                     └─*USER───┘

►─┬──────────────────────────────────┬─────────────────────────────────►
  │               ┌─*USER──┐         │
  └─DYNUSRPRF(────┴─*OWNER─┴───)─────┘

►─┬───────────────────────────────────────────────────────────┬────────►
  │               ┌─QTEMP/─────────┐ ┌─QSQLTEMP─────────┐      │
  └─TOSRCFILE(────┼────────────────┼─┴─source-file-name─┴──)──┘
                  ├─*LIBL/─────────┤
                  ├─*CURLIB/───────┤
                  └─library-name/──┘

►─┬─────────────────────────────────────┬─────────────────────────────►◄
  │          ┌─*SRCMBRTXT──────┐         │
  └─TEXT(────┼─*BLANK──────────┼─)──────┘
             └─'description'───┘
```

OPTION Details:



Notes:

1 All parameters preceding this point can be specified in positional form.

Purpose of the CRTSQLFTN command

The Create Structured Query Language FORTRAN (CRTSQLFTN) command calls the Structured Query Language (SQL) precompiler which precompiles FORTRAN source containing SQL statements, produces a temporary source member, and then optionally calls the FORTRAN compiler to compile the program.

Parameters of the CRTSQLFTN command

PGM

Specifies the qualified name of the compiled program.

The name of the compiled FORTRAN program can be qualified by one of the following library values:

***CURLIB:** The compiled FORTRAN program is created in the current library for the job. If no library is specified as the current library for the job, the QGPL library is used.

library-name: Specify the name of the library where the compiled FORTRAN program is created.

program-name: Specify the name of the compiled FORTRAN program.

SRCFILE

Specifies the qualified name of the source file that contains the FORTRAN source with SQL statements.

The name of the source file can be qualified by one of the following library values:

***LIBL:** All libraries in the job's library list are searched until the first match is found.

***CURLIB:** The current library for the job is searched. If no library is specified as the current library for the job, the QGPL library is used.

library-name: Specify the name of the library to be searched.

QFTNSRC: If the source file name is not specified, the IBM-supplied source file QFTNSRC contains the FORTRAN source.

source-file-name: Specify the name of the source file that contains the FORTRAN source.

SRCMBR

Specifies the name of the source file member that contains the FORTRAN source. This parameter is specified only if the source file name in the SRCFILE parameter is a database file. If this parameter is not specified, the PGM name specified on the PGM parameter is used.

***PGM:** Specifies that the FORTRAN source is in the member of the source file that has the same name as that specified on the PGM parameter.

source-file-member-name: Specify the name of the member that contains the FORTRAN source.

OPTION
Specifies whether one or more of the following options are used when the FORTRAN source is precompiled. If an option is specified more than once, or if two options conflict, the last option specified is used.

Element 1: Source Listing Options

***NOSOURCE:** or ***NOSRC:** A source printout is not produced by the precompiler unless errors are detected during precompile or create package.

***SOURCE or *SRC:** The precompiler produces a source printout consisting of FORTRAN source input.

Element 2: Cross-Reference Options

***NOXREF:** The precompiler does not cross-reference names.

***XREF:**The precompiler cross-references items in the program to the statement numbers in the program that refer to those items.

Element 3: Program Creation Options

***GEN:**

***NOGEN:** The precompiler does not call the FORTRAN compiler, and a program and SQL package are not created.

Element 4: Decimal Point Options

***PERIOD:** The value used as the decimal point for numeric constants used in SQL statements is a period.

***JOB** The value used as the decimal point for numeric constants in SQL is the representation of decimal point specified for the job at precompile time.

***SYSVAL:** The value used as the decimal point for numeric constants in SQL statements is the QDECFMT system value.

Note: If QDECFMT specifies that the value used as the decimal point is a comma, any numeric constants in lists (such as in the SELECT clause or the VALUES clause) must be separated by a comma followed by a blank. For example, VALUES(1,1, 2,23, 4,1) is equivalent to VALUES(1.1,2.23,4.1) in which the decimal point is a period.

***COMMA:** The value used as the decimal point for numeric constants in SQL statements is a comma.

Note: Any numeric constants in lists (such as in the SELECT clause or the VALUES clause) must be separated by a comma followed by a blank. For example, VALUES(1,1, 2,23, 4,1) is equivalent to VALUES(1.1,2.23,4.1) where the decimal point is a period.

Element 5: Naming Convention Options

***SYS:** The system naming convention (library-name/file-name) is used.

***SQL:** The SQL naming convention is used (schema-name.table-name). When creating a program on a remote database other than an iSeries server, *SQL must be specified as the naming convention.

Element 6: Second-Level Message Text Option

***NOSECLVL:** Second-level text descriptions are not added to the listing.

***SECLVL:** Second-level text with replacement data is added for all messages on the listing.

Element 7: Debug Options

***NODEBUG:** Symbolic extended program model (EPM) debug information is not stored with the program. This option is passed to the compiler and does not affect the SQL precompiler.

***DEBUG:** Symbolic EPM debug information is stored with the program. This option is passed to the compiler and does not affect the SQL precompiler.

TGTRLS

Specifies the release of the operating system on which the user intends to use the object being created.

In the examples given for the *CURRENT and *PRV values, and when specifying the *release-level* value, the format VxRxMx is used to specify the release, where Vx is the version, Rx is the release, and Mx is the modification level. For example, V2R3M0 is version 2, release 3, modification level 0.

***CURRENT:** The object is to be used on the release of the operating system currently running on the user's system. For example, if V2R3M5 is running on the system, *CURRENT means the user intends to use the object on a system with V2R3M5 installed. The user can also use the object on a system with any subsequent release of the operating system installed.

Note: If V2R3M5 is running on the system, and the object is to be used on a system with V2R3M0 installed, specify TGTRLS(V2R3M0) not TGTRLS(*CURRENT).

***PRV:** The object is to be used on the previous release with modification level 0 of the operating system. For example, if V2R3M5 is running on the user's system, *PRV means the user intends to use the object on a system with V2R2M0 installed. The user can also use the object on a system with any subsequent release of the operating system installed.

release-level: Specify the release in the format VxRxMx. The object can be used on a system with the specified release or with any subsequent release of the operating system installed.

Valid values depend on the current version, release, and modification level, and they change with each new release. If you specify a release-level which is earlier than the earliest release level supported by this command, an error message is sent indicating the earliest supported release.

INCFILE

Specifies the qualified name of the source file that contains members included in the program with any SQL INCLUDE statement.

The name of the source file can be qualified by one of the following library values:

***LIBL:** All libraries in the job's library list are searched until the first match is found.

***CURLIB:** The current library for the job is searched. If no library is specified as the current library for the job, the QGPL library is used.

library-name: Specify the name of the library to be searched.

***SRCFILE:** The qualified source file specified in the SRCFILE parameter contains the source file members specified on any SQL INCLUDE statement.

source-file-name: Specify the name of the source file that contains the source file members specified on any SQL INCLUDE statement. The record length of the source file the user specifies here must be no less than the record length of the source file specified on the SRCFILE parameter.

COMMIT

Specifies whether SQL statements in the compiled program are run under commitment control. Files referred to in the host language source are not affected by this option. Only SQL tables, SQL views, and SQL packages referred to in SQL statements are affected.

***CHG or *UR:** Specifies the objects referred to in SQL ALTER, CALL, COMMENT ON, CREATE, DROP, GRANT, LABEL ON, RENAME, and REVOKE statements and the rows updated, deleted, and inserted are locked until the end of the unit of work (transaction). Uncommitted changes in other jobs can be seen.

***ALL or *RS:** Specifies the objects referred to in SQL ALTER, CALL, COMMENT ON, CREATE, DROP, GRANT, LABEL ON, RENAME, and REVOKE statements and the rows selected, updated, deleted, and inserted are locked until the end of the unit of work (transaction). Uncommitted changes in other jobs cannot be seen.

***CS:** Specifies the objects referred to in SQL ALTER, CALL, COMMENT ON, CREATE, DROP, GRANT, LABEL ON, RENAME, and REVOKE statements and the rows updated, deleted, and inserted are locked until the end of the unit of work (transaction). A row that is selected, but not updated, is locked until the next row is selected. Uncommitted changes in other jobs cannot be seen.

***NONE or *NC:** Specifies that commitment control is not used. Uncommitted changes in other jobs can be seen. If the SQL DROP SCHEMA statement is included in the program, *NONE or *NC must be used. If a relational database is specified on the RDB parameter and the relational database is on a system that is not on an iSeries, *NONE or *NC cannot be specified.

***RR:** Specifies the objects referred to in SQL ALTER, CALL, COMMENT ON, CREATE, DROP, GRANT, LABEL ON, RENAME, and REVOKE statements and the rows selected, updated, deleted, and inserted are locked until the end of the unit of work (transaction). Uncommitted changes in other jobs cannot be seen. All tables referred to in SELECT, UPDATE, DELETE, and INSERT statements are locked exclusively until the end of the unit of work (transaction).

CLOSQLCSR

Specifies when SQL cursors are implicitly closed, SQL prepared statements are implicitly discarded, and LOCK TABLE locks are released. SQL cursors are explicitly closed when you issue the CLOSE, COMMIT, or ROLLBACK (without HOLD) SQL statements.

***ENDPGM:** SQL cursors are closed and SQL prepared statements are discarded when the program ends. LOCK TABLE locks are released when the first SQL program on the call stack ends.

***ENDSQL:** SQL cursors remain open between calls and can be fetched without running another SQL OPEN. One of the programs higher on the call stack must have run at least one SQL statement. SQL cursors are closed, SQL prepared statements are discarded, and LOCK TABLE locks are released when the first SQL program on the call stack ends. If *ENDSQL is specified for a program that is the first SQL program called (the first SQL program on the call stack), the program is treated as if *ENDPGM was specified.

***ENDJOB:** SQL cursors remain open between calls and can be fetched without running another SQL OPEN. The programs higher on the call stack do not need to have run SQL statements. SQL cursors are left open, SQL prepared statements are preserved, and LOCK TABLE locks are held when the first SQL program on the call stack ends. SQL cursors are closed, SQL prepared statements are discarded, and LOCK TABLE locks are released when the job ends.

ALWCPYDTA

Specifies whether a copy of the data can be used in a SELECT statement.

***OPTIMIZE:** The system determines whether to use the data retrieved directly from the database or to use a copy of the data. The decision is based on which method provides the best performance. If COMMIT is *CHG or *CS and ALWBLK is not *ALLREAD, or if COMMIT is *ALL or *RR, then a copy of the data is used only when it is necessary to run a query.

***YES:** A copy of the data is used only when necessary.

***NO:** A copy of the data is not allowed. If a temporary copy of the data is required to perform the query, an error message is returned.

ALWBLK

Specifies whether the database manager can use record blocking, and the extent to which blocking can be used for read-only cursors.

***ALLREAD:** Rows are blocked for read-only cursors if *NONE or *CHG is specified on the COMMIT parameter. All cursors in a program that are not explicitly able to be updated are opened for read-only processing even though EXECUTE or EXECUTE IMMEDIATE statements may be in the program.

Specifying *ALLREAD:

- Allows record blocking under commitment control level *CHG in addition to the blocking allowed for *READ.
- Can improve the performance of almost all read-only cursors in programs, but limits queries in the following ways:
  - The Rollback (ROLLBACK) command, a ROLLBACK statement in host languages, or the ROLLBACK HOLD SQL statement does not reposition a read-only cursor when *ALLREAD is specified.
  - Dynamic running of a positioned UPDATE or DELETE statement (for example, using EXECUTE IMMEDIATE), cannot be used to update a row in a cursor unless the DECLARE statement for the cursor includes the FOR UPDATE clause.

***NONE:** Rows are not blocked for retrieval of data for cursors.

Specifying *NONE:

- Guarantees that the data retrieved is current.
- May reduce the amount of time required to retrieve the first row of data for a query.
- Stops the database manager from retrieving a block of data rows that is not used by the program when only the first few rows of a query are retrieved before the query is closed.
- Can degrade the overall performance of a query that retrieves a large number of rows.

***READ:** Records are blocked for read-only retrieval of data for cursors when:

- *NONE is specified on the COMMIT parameter, which indicates that commitment control is not used.
- The cursor is declared with a FOR FETCH ONLY clause or there are no dynamic statements that could run a positioned UPDATE or DELETE statement for the cursor.

Specifying *READ can improve the overall performance of queries that meet the above conditions and retrieve a large number of records.

DLYPRP

Specifies whether the dynamic statement validation for a PREPARE statement is delayed until an OPEN, EXECUTE, or DESCRIBE statement is run. Delaying validation improves performance by eliminating redundant validation.

***NO:** Dynamic statement validation is not delayed. When the dynamic statement is prepared, the access plan is validated. When the dynamic statement is used in an OPEN or EXECUTE statement, the access plan is revalidated. Because the authority or the existence of objects referred to by the dynamic statement may change, you must still check the SQLCODE or SQLSTATE after issuing the OPEN or EXECUTE statement to ensure that the dynamic statement is still valid.

***YES:** Dynamic statement validation is delayed until the dynamic statement is used in an OPEN, EXECUTE, or DESCRIBE SQL statement. When the dynamic statement is used, the validation is completed and an access plan is built. If you specify *YES on this parameter, you should check the SQLCODE and SQLSTATE after running an OPEN, EXECUTE, or DESCRIBE statement to ensure that the dynamic statement is valid.

Note: If you specify *YES, performance is not improved if the INTO clause is used on the PREPARE statement or if a DESCRIBE statement uses the dynamic statement before an OPEN is issued for the statement.

GENLVL

Specifies the severity level at which the create operation fails. If errors occur that have a severity level greater than or equal to this value, the operation ends.

10: The default severity level is 10.

severity-level: Specify a value ranging from 0 through 40.

DATFMT

Specifies the format used when accessing date result columns. All output date fields are returned in the specified format. For input date strings, the specified value is used to determine whether the date is specified in a valid format.

Note: An input date string that uses the format *USA, *ISO, *EUR, or *JIS is always valid.

If a relational database is specified on the RDB parameter and the database is on a system that is not an iSeries server, then *USA, *ISO, *EUR, or *JIS must be specified.

***JOB:** The format specified for the job is used. Use the Display Job (DSPJOB) command to determine the current date format for the job.

***USA:** The United States date format (mm/dd/yyyy) is used.

***ISO:** The International Organization for Standardization (ISO) date format (yyyy-mm-dd) is used.

***EUR:** The European date format (dd.mm.yyyy) is used.

***JIS:** The Japanese Industrial Standard date format (yyyy-mm-dd) is used.

***MDY:** The date format (mm/dd/yy) is used.

***DMY:** The date format (dd/mm/yy) is used.

***YMD:** The date format (yy/mm/dd) is used.

***JUL:** The Julian date format (yy/ddd) is used.

DATSEP

Specifies the separator used when accessing date result columns.

Note: This parameter applies only when *JOB, *MDY, *DMY, *YMD, or *JUL is specified on the DATFMT parameter.

***JOB:** The date separator specified for the job at precompile time is used. Use the Display Job (DSPJOB) command to determine the current value for the job.

'/': A slash (/) is used.

'.': A period (.) is used.

',': A comma (,) is used.

'-': A dash (-) is used.

' ': A blank ( ) is used.

***BLANK:** A blank ( ) is used.

TIMFMT

Specifies the format used when accessing time result columns. For input time strings, the specified value is used to determine whether the time is specified in a valid format.

Note: An input date string that uses the format *USA, *ISO, *EUR, or *JIS is always valid.

If a relational database is specified on the RDB parameter and the database is on a system that is not another iSeries server, the time format must be *USA, *ISO, *EUR, *JIS, or *HMS with a time separator of colon or period.

<u>*HMS:</u> The (hh:mm:ss) format is used.

***USA:** The United States time format (hh:mm *xx*) is used, where *xx* is AM or PM.

***ISO:** The International Organization for Standardization (ISO) time format (hh.mm.ss) is used.

***EUR:** The European time format (hh.mm.ss) is used.

***JIS:** The Japanese Industrial Standard time format (hh:mm:ss) is used.

TIMSEP

Specifies the separator used when accessing time result columns.

Note: This parameter applies only when *HMS is specified on the TIMFMT parameter.

<u>*JOB:</u> The time separator specified for the job at precompile time is used. Use the Display Job (DSPJOB) command to determine the current value for the job.

':': A colon (:) is used.

'.': A period (.) is used.

',': A comma (,) is used.

' ': A blank ( ) is used.

***BLANK:** A blank ( ) is used.

REPLACE

Specifies whether a new program or SQL package is created when a program or SQL package of the same name exists in the same library. The value of this parameter is passed to the CRTFTNPGM command. More information on this parameter is in REPLACE parameter topic in the CL Reference section of the Information Center.

***YES:** A new program or SQL package is created, and any existing program or SQL package of the same name and type in the specified library is moved to QRPLOBJ.

***NO:** A new program or SQL package is not created if an object of the same name and type already exists in the specified library.

RDB

Specifies the name of the relational database where the SQL package object is created.

<u>*LOCAL:</u> The program is created as a distributed SQL program. The SQL statements will access the local database. An SQL package object is not created as part of the precompile process. The Create Structured Query Language Package (CRTSQLPKG) command can be used.

relational-database-name: Specify the name of the relational database where the new SQL package object is to be created. When the name of the local relational database is specified, the program created is still a distributed SQL program. The SQL statements will access the local database.

***NONE:** An SQL package object is not created. The program object is not a distributed program and the Create Structured Query Language Package (CRTSQLPKG) command cannot be used.

USER
Specifies the user name sent to the remote system when starting the conversation. This parameter is valid only when RDB is specified.

***CURRENT:** The user profile under which the current job is running is used.

user-name: Specify the user name being used for the application server job.

PASSWORD
Specifies the password to be used on the remote system. This parameter is valid only if RDB is specified.

***NONE:** No password is sent. If this value is specified, USER(*CURRENT) must also be specified.

password: Specify the password of the user name specified on the USER parameter.

RDBCNNMTH
Specifies the semantics used for CONNECT statements. Refer to the CONNECT (TYPE1) and CONNECT (TYPE2) in the *SQL Reference* book for more information.

***DUW:** CONNECT (Type 2) semantics are used to support distributed unit of work. Consecutive CONNECT statements to additional relational databases do not result in disconnection of previous connections.

***RUW:** CONNECT (Type 1) semantics are used to support remote unit of work. Consecutive CONNECT statements result in the previous connection being disconnected before a new connection is established.

DFTRDBCOL
Specifies the schema name used for the unqualified names of tables, views, indexes, and SQL packages. This parameter applies only to static SQL statements.

***NONE:** The naming convention defined on the OPTION parameter is used.

schema-name: Specify the name of the schema identifier. This value is used instead of the naming convention specified on the OPTION parameter.

SQLPKG
Specifies the qualified name of the SQL package created on the relational database specified on the RDB parameter of this command.

The possible library values are:

***PGMLIB:** The package is created in the library with the same name as the library containing the program.

library-name: Specify the name of the library where the package is created.

***PGM:** The package name is the same as the program name.

package-name: Specify the name of the package created on the remote database specified on the RDB parameter.

SAAFLAG

Specifies the IBM SQL flagging function. This parameter flags SQL statements to verify whether they conform to IBM SQL syntax. More information about which IBM database products IBM SQL syntax is in the *DRDA IBM SQL Reference*, SC26-3255-00.

***NOFLAG:** The precompiler does not check to see whether SQL statements conform to IBM SQL syntax.

***FLAG:** The precompiler checks to see whether SQL statements conform to IBM SQL syntax.

FLAGSTD

Specifies the American National Standards Institute (ANSI) flagging function. This parameter flags SQL statements to verify whether they conform to the following standards.

```
ANSI X3.135-1992 entry
ISO 9075-1992 entry
FIPS 127.2 entry
```

***NONE:** The precompiler does not check to see whether SQL statements conform to ANSI standards.

***ANS:** The precompiler checks to see whether SQL statements conform to ANSI standards.

PRTFILE

Specifies the qualified name of the printer device file to which the listing is directed. The file must have a minimum record length of 132 bytes or information is lost.

The name of the printer file can be qualified by one of the following library values:

***LIBL:** All libraries in the job's library list are searched until the first match is found.

***CURLIB:** The current library for the job is searched. If no library is specified as the current library for the job, the QGPL library is used.

library-name: Specify the name of the library to be searched.

QSYSPRT: If a file name is not specified, the precompiler printout is directed to the IBM-supplied printer file QSYSPRT.

printer-file-name: Specify the name of the printer device file to which the precompiler printout is directed.

SRTSEQ

Specifies the sort sequence table to be used for string comparisons in SQL statements.

Note: *HEX must be specified for this parameter on distributed applications where the application server is not on an iSeries server or the release level is prior to V2R3M0.

***JOB:** The SRTSEQ value for the job is retrieved during the precompile.

***JOBRUN:** The SRTSEQ value for the job is retrieved when the program is run. For distributed applications, SRTSEQ(*JOBRUN) is valid only when LANGID(*JOBRUN) is also specified.

***LANGIDUNQ:** The unique-weight sort table for the language specified on the LANGID parameter is used.

***LANGIDSHR:** The shared-weight sort table for the language specified on the LANGID parameter is used.

***HEX:** A sort sequence table is not used. The hexadecimal values of the characters are used to determine the sort sequence.

The name of the sort sequence table can be qualified by one of the following library values:

***LIBL:** All libraries in the job's library list are searched until the first match is found.

***CURLIB:** The current library for the job is searched. If no library is specified as the current library for the job, the QGPL library is used.

library-name: Specify the name of the library to be searched.

table-name: Specify the name of the sort sequence table to be used.

LANGID

Specifies the language identifier to be used when SRTSEQ(*LANGIDUNQ) or SRTSEQ(*LANGIDSHR) is specified.

***JOB:** The LANGID value for the job is retrieved during the precompile.

***JOBRUN:** The LANGID value for the job is retrieved when the program is run. For distributed applications, LANGID(*JOBRUN) is valid only when SRTSEQ(*JOBRUN) is also specified.

language-id: Specify a language identifier to be used by the program.

USRPRF

Specifies the user profile that is used when the compiled program object is run, including the authority that the program object has for each object in static SQL statements. The profile of either the program owner or the program user is used to control which objects can be used by the program object.

***NAMING:** The user profile is determined by the naming convention. If the naming convention is *SQL, USRPRF(*OWNER) is used. If the naming convention is *SYS, USRPRF(*USER) is used.

***USER:** The profile of the user running the program object is used.

***OWNER:** The user profiles of both the program owner and the program user are used when the program is run.

DYNUSRPRF

Specifies the user profile used for dynamic SQL statements.

***USER:** Local dynamic SQL statements are run under the user profile of the job. Distributed dynamic SQL statements are run under the user profile of the application server job.

***OWNER:** Local dynamic SQL statements are run under the user profile of the program's owner. Distributed dynamic SQL statements are run under the user profile of the SQL package's owner.

TOSRCFILE

Specifies the qualified name of the source file that is to contain the output source member that has been processed by the SQL precompiler. If the specified source file is not found, it will be created. The output member will have the same name as the name that is specified for the SRCMBR parameter.

The possible library values are:

QTEMP: The library QTEMP will be used.

***LIBL:** The job's library list is searched for the specified file. If the file is not found in any library in the library list, the file will be created in the current library.

***CURLIB:** The current library for the job will be used. If no library is specified as the current library for the job, the QGPL library will be used.

library-name: Specify the name of the library that is to contain the output source file.

QSQLTEMP: The source file QSQLTEMP will be used.

source-file-name: Specify the name of the source file to contain the output source member.

TEXT

Specifies the text that briefly describes the LANGID. More information on this parameter is in the TEXT parameter topic in the CL Reference section of the Information Center.

***SRCMBRTXT:** The text is taken from the source file member being used to create the FORTRAN program. Text can be added or changed for a database source member by using the Start Source Entry Utility (STRSEU) command, or by using either the Add Physical File Member (ADDPFM) or Change Physical File Member (CHGPFM) command. If the source file is an inline file or a device file, the text is blank.

***BLANK:** Text is not specified.

'description': Specify no more than 50 characters of text, enclosed in apostrophes.

Example of the CRTSQLFTN command

```
CRTSQLFTN   PAYROLL  TEXT('Payroll Program')
```

This command runs the SQL precompiler, which precompiles the source and stores the changed source in member PAYROLL in file QSQLTEMP in library QTEMP. The FORTRAN compiler is called to create program PAYROLL in the current library by using the source member created by the SQL precompiler.

Appendix D. Coding SQL Statements in FORTRAN Applications

This appendix describes the unique application and coding requirements for embedding SQL statements in a FORTRAN program. Requirements for host variables are defined.

For more details, see the following sections:

- "Defining the SQL Communications Area in FORTRAN applications"
- "Defining SQL Descriptor Areas in FORTRAN applications" on page 316
- "Embedding SQL statements in FORTRAN applications" on page 317
- "Using host variables in FORTRAN applications" on page 319
- "Determining equivalent SQL and FORTRAN data types" on page 321
- "Using indicator variables in FORTRAN applications" on page 322

Note: See "Code disclaimer information" on page viii information for information pertaining to code examples.

Defining the SQL Communications Area in FORTRAN applications

A FORTRAN program that contains SQL statements must include one or both of the following:

- An SQLCOD variable declared as INTEGER
- An SQLSTA (or SQLSTATE) variable declared as CHARACTER*5

Or,

- An SQLCA (which contains an SQLCOD and SQLSTA variable).

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

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

```
EXEC SQL INCLUDE SQLCA
```

The included FORTRAN source statements for the SQLCA are:

```
*
*    The SQL communications area
*
     CHARACTER SQLCA(136)
     CHARACTER SQLCAID*8
     INTEGER*4 SQLCABC
     INTEGER*4 SQLCODE
     INTEGER*2 SQLERRML
     CHARACTER SQLERRMC*70
     CHARACTER SQLERRP*8
     INTEGER*4 SQLERRD(6)
     CHARACTER SQLWARN*11
     CHARACTER SQLSTATE*5
```

```
     EQUIVALENCE (SQLCA(  1), SQLCAID)
     EQUIVALENCE (SQLCA(  9), SQLCABC)
     EQUIVALENCE (SQLCA( 13), SQLCODE)
     EQUIVALENCE (SQLCA( 17), SQLERRML)
     EQUIVALENCE (SQLCA( 19), SQLERRMC)
     EQUIVALENCE (SQLCA( 89), SQLERRP)
     EQUIVALENCE (SQLCA( 97), SQLERRD)
     EQUIVALENCE (SQLCA(121), SQLWARN)
     EQUIVALENCE (SQLCA(132), SQLSTATE)

*
     INTEGER*4 SQLCOD
    C         SQLERR(6)
     INTEGER*2 SQLTXL
     CHARACTER SQLERP*8,
    C         SQLWRN(0:7)*1,
    C         SQLWRX(1:3)*1,
    C         SQLTXT*70,
    C         SQLSTT*5,
    C         SQLWRNWK*8,
    C         SQLWRXWK*3,
    C         SQLERRWK*24,
    C         SQLERRDWK*24
     EQUIVALENCE (SQLWRN(1), SQLWRNWK)
     EQUIVALENCE (SQLWRX(1), SQLWRXWK)
     EQUIVALENCE (SQLCA(97), SQLERRDWK)
     EQUIVALENCE (SQLERR(1), SQLERRWK)
     COMMON /SQLCA1/SQLCOD,SQLERR,SQLTXTL
     COMMON /SQLCA2/SQLERP,SQLWRN,SQLTXT,SQLWRX,SQLSTT
```

SQLSTATE is replaced with SQLSTOTE when a declare for SQLSTATE is found in the program and the SQLCA is provided by the compiler. If compatibility with other IBM SQL implementations is not a primary consideration, it is recommended that the SQLCA be included by coding the FORTRAN variable SQLCOD, SQLSTA, or SQLSTATE in the program. This improves performance, but does not generate a compatible SQLCA.

For More information about SQLCA, see SQL Communication Area in the SQL Reference book.

The SQLCOD, SQLSTA, SQLSTATE, and SQLCA variables must be placed before the first executable SQL statement. All executable SQL statements in a program must be within the scope of the declaration of the SQLCOD, SQLSTA, SQLSTATE, and SQLCA variables.

All SQL statements that can be run in a program must be within the scope of the declaration of the SQLCOD variable or SQLCA variables.

Defining SQL Descriptor Areas in FORTRAN applications

The following statements require 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*

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 described in Dynamic SQL Applications in the *DB2 UDB for iSeries Programming Concepts* information. 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, which are not supported by FORTRAN, an INCLUDE SQLDA statement cannot be specified in a FORTRAN program. Unless an SQLDA is set up by a C, COBOL, PL/I, or ILE RPG program and passed to the FORTRAN program, you cannot use the SQLDA.

For More information about SQLDA, see SQL Descriptor Area in the *SQL Reference* book.

Coding an SQLDA on the multiple-row FETCH statement using a row storage area provides a technique to retrieve multiple rows on each FETCH statement. This technique can improve an application's performance if a large number of rows are read by the application. For More information about using the FETCH statement, see the SQL Reference book.

Embedding SQL statements in FORTRAN applications

SQL statements can be coded in a FORTRAN program wherever a statement that can be run appears. If the SQL statement is within an IF statement, any necessary THEN and END IF statements will be generated.

Each SQL statement in a FORTRAN program must begin with EXEC SQL. The EXEC SQL keywords must appear all on one line, but the remainder of the statement can appear on the same line and on subsequent lines.

Example:

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

```
  EXEC SQL
 C  UPDATE DEPARTMENT
 C  SET MGRNO = :MGRNUM
 C  WHERE DEPTNO = :INTDEPT
```

An SQL statement cannot be followed on the same line by another SQL statement or by a FORTRAN statement.

FORTRAN does not require the use of blanks to delimit words within a statement, but the SQL language does. The rules for embedded SQL follow the rules for SQL syntax, which requires the use of one or more blanks as delimiters.

For more details, see the following sections:

- "Comments in FORTRAN applications that use SQL" on page 318
- "Debug lines in FORTRAN applications that use SQL" on page 318
- "Continuation for SQL statements in FORTRAN applications that use SQL" on page 318
- "Including code in FORTRAN applications that use SQL" on page 318

Comments in FORTRAN applications that use SQL

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

The comment extends to the end of the line. Comment lines can appear between the lines of a continued SQL statement. The character (!) indicates a comment, except when it appears in a character context or in column 6.

Debug lines in FORTRAN applications that use SQL

Lines contain debug statements (’D’ or ’d’ in column 1) are treated as comments lines by the precompiler.

Continuation for SQL statements in FORTRAN applications that use SQL

The line continuation rules for SQL statements are the same as those for other FORTRAN 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 character in column 73 of the continued line and placing the shift-out character in column 6 of the continuation line.

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'<AABBCC>
     <DDEEFFGGHHIIJJKK>'
```

Including code in FORTRAN applications that use SQL

SQL statements or FORTRAN statements can be included by embedding the following SQL statement at the point in the source code where the statements are to be embedded:

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

The FORTRAN INCLUDE compiler directive cannot be used to include SQL statements or FORTRAN host variable declarations that are to be used in an SQL statement.

Margins in FORTRAN applications that use SQL

Code the SQL statements (starting with `EXEC SQL`) in coding columns 7 to 72.

Names in FORTRAN applications that use SQL

Any valid FORTRAN 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 'SQ', 'SQL', 'RDI', or 'DSN'. These names are reserved for the database manager.

Do not use the following keywords to identify host variables:

FUNCTION
IMPLICIT
PROGRAM
SUBROUTINE

Statement Labels in FORTRAN applications that use SQL

Executable SQL statements can have statement numbers associated with them, specified in columns 1 to 5. However, during program preparation, a labelled SQL statement causes a CONTINUE statement with that label to be generated before the code runs the statement. A labelled SQL statement should not be the last statement in a DO loop. Because CONTINUE statements can be run, SQL statements that occur before the first statement that can be run in a FORTRAN program (for example, INCLUDE and BEGIN DECLARE SECTION) should not be labelled.

WHENEVER statement in FORTRAN applications that use SQL

The target for the GOTO clause in the SQL WHENEVER statement must be a label in the FORTRAN source and must reference a statement in the same subprogram. A WHENEVER statement only applies to SQL statements in the same subprogram.

FORTRAN compile-time options in the SQL precompiler

The FORTRAN PROCESS statement can be used to specify the compile-time options for the FORTRAN compiler. Although the PROCESS statement will be recognized by the FORTRAN compiler when it is called by the precompiler to create the program, the SQL precompiler itself does not recognize the PROCESS statement.

Using host variables in FORTRAN applications

All host variables used in SQL statements must be explicitly declared. Implicit declarations of host variables via default typing or by the IMPLICIT statement are not supported. A host variable used in an SQL statement must be declared prior to the first use of the host variable in an SQL statement.

The FORTRAN 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. Note: LOB and ROWID host variables are not supported in FORTRAN.

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

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

The declaration for a character host variable must not use an expression to define the length of the character variable. The declaration for a character host variable must not have an undefined length (for example, CHARACTER(*)).

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 arrays (subscripted variables).

For more details, see “Declaring host variables in FORTRAN applications”.

Declaring host variables in FORTRAN applications

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

Numeric host variables in FORTRAN applications

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

Numeric

```
                                   ┌─,──────────────────────────────────────────┐
                                   ▼                                            │
►►──┬─INTEGER*2────────┬──────variable-name──┬─────────────────────────────┬──┴──────►◄
    ├─INTEGER──┬─────┬─┤                     └─/──numeric-constant──/─────┘
    │          └─*4──┘ │
    ├─REAL──┬─────┬────┤
    │       └─*4──┘    │
    ├─REAL*8───────────┤
    └─DOUBLE PRECISION─┘
```

Character host variables in FORTRAN applications

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

Character

```
►►──CHARACTER──┬─────┬──────────────────────────────────────────────────────►
               └─*n──┘

   ┌─,──────────────────────────────────────────────────────┐
   ▼                                                        │
►──variable-name──┬─────┬──┬───────────────────────────────┬┴──────────────►◄
                  └─*n──┘  └─/──character-constant──/──────┘
```

Note: n must be a constant no greater than 32766.

Determining equivalent SQL and FORTRAN data types

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

Table 12. FORTRAN Declarations Mapped to Typical SQL Data Types

FORTRAN Data Type	SQLTYPE of Host Variable	SQLLEN of Host Variable	SQL Data Type
INTEGER*2	500	2	SMALLINT
INTEGER*4	496	4	INTEGER
REAL*4	480	4	FLOAT (single precision)
REAL*8	480	8	FLOAT (double precision)
CHARACTER*n	452	n	CHAR(n)

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

Table 13. SQL Data Types Mapped to Typical FORTRAN Declarations

SQL Data Type	FORTRAN Equivalent	Explanatory Notes
SMALLINT	INTEGER*2	
INTEGER	INTEGER*4	
BIGINT	No exact equivalent	Use REAL*8
DECIMAL(p,s) or NUMERIC(p,s)	No exact equivalent	Use REAL*8
FLOAT (single precision)	REAL*4	
FLOAT (double precision)	REAL*8	
CHAR(n)	CHARACTER**n*	*n* is a positive integer from 1 to 32766.
VARCHAR(n)	No exact equivalent	Use a character host variable large enough to contain the largest expected VARCHAR value.
GRAPHIC(n)	Not supported	Not supported
VARGRAPHIC(n)	Not supported	Not supported
DATE	CHARACTER**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	CHARACTER**n*	*n* must be at least 6; to include seconds, *n* must be at least 8.

Table 13. SQL Data Types Mapped to Typical FORTRAN Declarations (continued)

SQL Data Type	FORTRAN Equivalent	Explanatory Notes
TIMESTAMP	CHARACTER*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.

For more details, see "Notes on FORTRAN variable declaration and usage".

Notes on FORTRAN variable declaration and usage

In FORTRAN, a string of digits with a decimal point is interpreted as a real constant. In an SQL statement, such a string is interpreted as a decimal constant. Therefore, use exponent notation when specifying a real (floating-point) constant in an SQL statement.

In FORTRAN, a real (floating-point) constant having a length of eight bytes uses a D as the exponent indicator (for example, 3.14159D+04). An 8-byte floating-point constant in an SQL statement must use an E (for example, 3.14159E+04).

Using indicator variables in FORTRAN applications

An indicator variable is a two-byte integer (INTEGER*2). 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.

See the indicator variables topic in the SQL Reference book for more information.

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 the programmer.

Example:

Given the statement:

```
   EXEC SQL FETCH CLS_CURSOR INTO :CLS_CD,
  C                               :DAY :DAY_IND,
  C                               :BGN :BGN_IND,
  C                               :ENDCLS :ENDCLS_IND
```

The variables can be declared as follows:

```
   EXEC SQL BEGIN DECLARE SECTION
    CHARACTER*7 CLS_CD
    INTEGER*2   DAY
    CHARACTER*8 BGN, ENDCLS
    INTEGER*2   DAY_IND, BGN_IND, ENDCLS_IND
   EXEC SQL END DECLARE SECTION
```

Index

D

T

U

V

W

Printed in U.S.A.