18

Tandem COBOL85 Embedded SQL/MP Support

NonStop Structured Query Language/Massively Parallel (NonStop SQL/MP) commands store and retrieve sets of fields from SQL databases. You can embed an SQL command in your Tandem COBOL85 program wherever a COBOL85 statement is allowed in the Procedure Division.

Tandem COBOL85 supports embedded SQL statements in both the COBOL85 environment and the CRE. The Tandem COBOL85 embedded SQL implementation conforms to the ANSI Database—Embedded SQL Standard (ANSI X3.168-1989), with the restrictions and extensions mentioned in this section.

This section explains:

- That embedded SQL statements cannot contain compiler directives
- How to produce an object file from a COBOL85 source file that contains embedded SQL statements
- Differences between SQLCOBOL preprocessing and the COBOL85 and NMCOBOL compilers' own SQL processing
- How to protect SQL databases (which cannot be updated by COBOL85 programs that run as fault-tolerant process pairs)
- How embedded SQL affects Tandem COBOL85 conformance to ISO/ANSI standards
- How embedded SQL affects Tandem COBOL85 performance

For more information on using SQL with COBOL85, see the *NonStop SQL/MP Programming Manual for COBOL85*.

Compiler Directives in Embedded SQL Statements

Embedded SQL statements cannot contain compiler directives. Do not put compiler directives between "EXEC SQL" and "END-EXEC." For example, the following code is not allowed:

```
      EXEC SQL
?SOURCE XYZ
      END-EXEC
```

Producing an Object File

You can produce an object file from a COBOL85 source file that contains embedded SQL statements by submitting the COBOL85 source file directly to the COBOL85 or NMCOBOL compiler. Both compilers require an SQL directive. For the COBOL85 compiler, the SQL directive can be either in the source program or on the compiler invocation command line. For the NMCOBOL compiler, the SQL directive must be on the compiler invocation command line.

If you used to use the SQLCOBOL preprocessor, you might need to make minor changes to your program (see Compiler-SQLCOBOL Differences on page 18-3).

Note. If a TNS COBOL85 program contains SQL statements, you cannot compile it with the directive ENV LIBRARY, and you must use the file COBOLEX0 instead of the file COBOLEXT.

Compiler-SQLCOBOL Differences

The COBOL85 and NMCOBOL compilers accept, process, and list embedded SQL statements as the SQLCOBOL preprocessor did, with these exceptions:

- The compilers accept compiler directives that provide them with information for processing SQL statements (SQLCOBOL did not).
- The compilers support Extended Dynamic SQL (SQLCOBOL did not).
- The compilers handle these differently than SQLCOBOL did:
  - Cursor support
    (see Cursor Support on page 18-5)
  - SQL statement placement
    (see SQL Statement Placement on page 18-5)
  - COPY libraries
    (see COPY Libraries on page 18-6)
  - SQLCA structure declaration
    (see SQLCA Structure Declaration on page 18-6)
  - SQLCODE identifier declaration
    (see SQLCODE Identifier Declaration on page 18-6)
  - Attachments to the SQLCODE identifier
    (see Attachments to the SQLCODE Identifier on page 18-7)
  - SQL data structure placement
    (see SQL Data Structure Placement on page 18-7)
  - Compiler listing
    (see Compiler Listing on page 18-8)
  - INVOKE and INCLUDE directives
    (see INVOKE and INCLUDE Directives on page 18-8)
  - COPY and REPLACE statements
    (see COPY and REPLACE Statements on page 18-9)
- The compilers work differently with these than SQLCOBOL did:
  - Inspect interactive symbolic debugger
    (see Inspect Interactive Symbolic Debugger on page 18-9)
  - CROSSREF utility program and CROSSREF directive
    (see CROSSREF Utility Program and CROSSREF Directive on page 18-10)

Compiler Directives

Table 18-1 lists and briefly describes the compiler directives that support embedded SQL. For complete descriptions, see SQL and NOSQL on page 11-113 and SQLMEM on page 11-115.

Table 18-1. Directives That Support Embedded SQL

Directive	Description	Required or Optional
SQL	Tells the COBOL85 or NMCOBOL compiler to expect SQL statements in the compilation unit. Can also specify the amount of memory to be allocated to the SQL Compiler Interface (SCI) and the SQL information to be printed to the listing file. The NMCOBOL compiler accepts the SQL directive on the compiler invocation command line, but not in the source program.	Required if the compilation unit contains SQL statements
SQLMEM	Tells the COBOL85 compiler whether to declare SQL data structures in the Working-Storage Section or the Extended-Storage Section. The NMCOBOL compiler does not recognize SQLMEM, because the concept of extended memory does not exist in native mode.	Optional—if absent, the compiler declares SQL data structures in the Extended-Storage Section

Extended Dynamic SQL (EDS) Support

Unlike SQLCOBOL, the COBOL85 and NMCOBOL compilers support Extended Dynamic SQL (EDS). EDS does these things:

- Allows statement and cursor name variables
- Allows NonStop TS/MP to access SQL
- Allows users to access an NSSQL server across a network (by means of the OPEN SQL project)

Statement and Cursor Name Variables

Without EDS, statement and cursor names must be specified as literals; for example:

```
EXEC SQL PREPARE S1 FROM :H          END-EXEC
EXEC SQL DECLARE C1 CURSOR FOR S1    END-EXEC
EXEC SQL OPEN C1                     END-EXEC
```

With EDS, statement and cursor names can be specified in variables; for example:

```
MOVE "S1" INTO S
MOVE "C1" INTO C
EXEC SQL PREPARE :S FROM :H          END-EXEC
EXEC SQL DECLARE :C CURSOR FOR :S    END-EXEC
EXEC SQL OPEN :C                     END-EXEC
```

Variables have two advantages:

- Programs can be simpler—a program needs only one PREPARE, DESCRIBE, OPEN, FETCH, and CLOSE statement, rather than one for each name.
- Programs need not predetermine all of the statement, program, and cursor names that they use.

With EDS, you do not need to prespecify the maximum number of concurrent statement names. Without EDS, the maximum number of distinct statement names is 20.

NonStop TS/MP Access to SQL

NonStop TS/MP can access SQL if it has EDS; otherwise it cannot.

Access to NSSQL Servers

Tandem supports an SQL server that accepts popular protocols and a set of commands. The commands correspond to those of SQL with EDS (PREPARE, DESCRIBE, OPEN, FETCH, and CLOSE). The server has a loop that reads these commands, executes the corresponding SQL commands, and returns data and error information. Users on other work stations can call library routines that support the protocol and form the commands.

Cursor Support

The SQLCOBOL preprocessor supported SQL “local” and “foreign” cursors. The COBOL85 and NMCOBOL compilers support only SQL “local” cursors.

SQL Statement Placement

The SQLCOBOL preprocessor did not allow COBOL85 statements and embedded SQL statements to appear on the same line. The COBOL85 and NMCOBOL compilers do allow this, with these restrictions:

- The COBOL85 statements must follow the embedded SQL statement terminator; they cannot precede an embedded SQL statement on a line.
- The COBOL85 statements cannot be COPY or REPLACE statements. (This is an extension to the preprocessor, but a restriction on the ANSI Database—Embedded SQL Standard.)

Note. Like the SQLCOBOL preprocessor, the COBOL85 and NMCOBOL compilers do not allow embedded SQL statements on COBOL85 debugging lines.

COPY Libraries

The SQLCOBOL preprocessor used the SQL SOURCE directive to copy source text from a COPY library into a COBOL85 program. The COBOL85 and NMCOBOL compilers do not support the SQL SOURCE directive, because it is not part of the ANSI Database—Embedded SQL Standard. Instead, the compilers use their own SOURCE directive (described in SOURCE on page 11-110). Because the SQL SOURCE directive and the compilers' SOURCE directive have the same format, this difference between the preprocessor and the compiler does not affect copied files.

SQLCA Structure Declaration

The SQLCOBOL preprocessor declared an SQLCA structure in the Data Division of each program of the compilation unit whether or not each program contained an INCLUDE SQLCA directive. The COBOL85 and NMCOBOL compilers declare an SQLCA structure in a program only if that program contains an INCLUDE SQLCA directive.

The SQLCOBOL preprocessor did not allow an INCLUDE SQLCA directive in the Extended-Storage section; the COBOL85 and NMCOBOL compilers do.

SQLCODE Identifier Declaration

The SQLCOBOL preprocessor declared an SQLCODE identifier implicitly through the declaration of the SQLCA structure (which it declared whether or not a program contained an INCLUDE SQLCA directive). The COBOL85 and NMCOBOL compilers require each program and nested program that contains SQL statements to declare an SQLCODE identifier.

The declaration of the SQLCODE identifier can be implicit or explicit, but not both. To declare an SQLCODE identifier implicitly, the program or nested program must contain an INCLUDE SQLCA directive. To declare an SQLCODE identifier explicitly, the program or nested program must declare the SQLCODE identifier as a PIC S9(4) COMP elementary item.

Attachments to the SQLCODE Identifier

Suppose that you want to attach level-88 items to the SQLCODE data item and still be able to declare an SQLCA structure.

The COBOL85 and NMCOBOL compilers allow the same approach, but it is not recommended, because SQLCODEX is not part of the ANSI Database—Embedded SQL Standard. The recommended approach is to omit the INCLUDE SQLCA directive from each program or nested program. This prevents the compiler from declaring the SQLCODE data item implicitly so that you can declare it explicitly and attach level-88 items to it.

If you choose not to omit the INCLUDE SQLCA directives, follow these steps:

1. Substitute an SQLCODEX data item for the SQLCODE data item.

   To do this, include an INVOKE SQLCODEX statement in each program or nested program in which you want an SQLCODEX data item.

2. Attach level-88 items to the SQLCODEX data item.

3. Declare the SQLCA structure.

SQL Data Structure Placement

The SQLCOBOL preprocessor declared data structures for each SQL statement in a COBOL85 program (SQLIN is an example of such a data structure). If the program declared an Extended-Storage Section, the preprocessor declared these data structures there; if not, it declared them in the Working-Storage Section.

The COBOL85 compiler also declares data structures for each SQL statement in a COBOL85 program, but allows you to use the SQLMEM directive to specify whether to declare them in the Working-Storage Section or the Extended-Storage Section. If you do not specify a section (that is, if you omit the SQLMEM directive), the COBOL85 compiler declares SQL data structures in the Extended-Storage Section.

It is recommended that you allow the COBOL85 compiler to declare SQL data structures in the Extended-Storage Section (that is, specify SQLMEM EXT or do not include an SQLMEM directive in your program). The reasons are:

- The Extended-Storage Section already exists (the COBOL85 run-time environment creates one for every COBOL85 object program, whether or not it declares one).
- The number and size of SQL statements are restricted when their data structures are declared in the Working-Storage Section, because SQL data structures can exhaust the Working-Storage Section.

The COBOL85 or NMCOBOL compiler declares SQL data structures automatically; you do not declare them in the Data Division.

Compiler Listing

The SQLCOBOL preprocessor included processed SQL statements in the new COBOL85 source file that it created. These processed SQL statements were in the form of SQL data structures, PERFORM statements, and calls to TAL routines. When the COBOL85 compiler compiled this new source file, it listed these processed SQL statements in the compiler listing in their processed form; that is, as SQL data structures, PERFORM statements, and calls to TAL routines. If the COBOL85 compiler detected an error in a processed SQL statement, it printed the error message after the offending SQL data structure, PERFORM statement, or TAL routine call.

The COBOL85 and NMCOBOL compilers list SQL statements in the compiler listing exactly as they appear in the source program. They do not list the SQL data structures, PERFORM statements, or calls to TAL routines that the SQL statements generate (except invoked or included data structures—see INVOKE and INCLUDE Directives). If the compiler detects an error in an embedded SQL statement, it prints the error message after the offending SQL statement itself.

INVOKE and INCLUDE Directives

Like the SQLCOBOL preprocessor, the COBOL85 and NMCOBOL compilers replace the SQL directives INVOKE and INCLUDE with COBOL85 data declarations that correspond to the SQL structures being invoked or included. Like the SQLCOBOL preprocessor, the COBOL85 and NMCOBOL compilers list these data declarations in the compiler listing, but for different reasons.

The SQLCOBOL preprocessor listed these data declarations in the COBOL85 compiler listing because it listed all processed SQL statements. The COBOL85 and NMCOBOL compilers list them despite the fact that they do not list any other SQL data declarations (for more information, see Compiler Listing). Their line numbering is consistent with the current format of the program text (Tandem or ASCII). To prevent the COBOL85 or NMCOBOL compiler from listing these data declarations, use the NOLIST directive (explained in “LIST and NOLIST” in Section 11).

The COBOL85 and NMCOBOL compilers do not allow INVOKE or INCLUDE directives when a REPLACE statement is active. (The SQLCOBOL preprocessor did not allow REPLACE statements in any context.)

The COBOL85 and NMCOBOL compilers do not allow text to follow an INVOKE or INCLUDE directive on the same line.

COPY and REPLACE Statements

The SQLCOBOL preprocessor did not allow COPY or REPLACE statements in any context.

The COBOL85 and NMCOBOL compilers have these restrictions on COPY and REPLACE statements:

- COPY and REPLACE statements are not allowed in SQL statements.
- COPY and REPLACE statements cannot contain SQL statements.
- Library text introduced by COPY statements cannot contain SQL statements.
- A REPLACE statement that precedes one or more SQL statements does not affect them (except between BEGIN DECLARE and END DECLARE statements).
- COPY and REPLACE statements do affect SQL statements between BEGIN DECLARE and END DECLARE statements, but using them to manipulate SQL this way is not recommended.

Note. COBOL85 identifiers declared between BEGIN DECLARE and END DECLARE directives are not SQL text, but they can appear in SQL statements.

Inspect Interactive Symbolic Debugger

You can use the Inspect interactive symbolic debugger on any COBOL85 object file; however, the current source line indicated by the Inspect debugger depends on how you produced the object file.

If you used the SQLCOBOL preprocessor, the current source line indicated by the Inspect debugger was not the embedded SQL statement itself but a COBOL85 statement generated by the SQLCOBOL preprocessor.

When you use the COBOL85 or NMCOBOL compiler, the current source line indicated by the Inspect debugger is the embedded SQL statement itself.

For more information on the Inspect debugger, see Inspect Interactive Symbolic Debugger on page 15-1.

Note. The Inspect debugger handles TNS and native COBOL85 programs differently—see Using the Inspect Debugger for Native COBOL85 Programs on page 34-12.

CROSSREF Utility Program and CROSSREF Directive

You can use the CROSSREF utility program or the CROSSREF directive only on TNS COBOL85 programs. You cannot use them on native COBOL85 programs, because the NMCOBOL compiler does not produce a cross-reference listing.

The SQLCOBOL preprocessor and the COBOL85 compiler put the same restrictions on COBOL85 variables, and place and use them in the same way; however, the information that the CROSSREF utility program or directive provides depends on how you produced the object file.

If you used the preprocessor, the CROSSREF utility program or the CROSSREF directive told you whether the COBOL85 variables in an embedded SQL statement were referenced or changed; however, it could not list the embedded SQL statement that contained those variables, because that SQL statement had been broken down into COBOL85 statements.

When you use the COBOL85 compiler, the CROSSREF utility program or the CROSSREF directive assumes that any COBOL85 variables in an embedded SQL statement are both referenced and changed. It lists the embedded SQL statement that contains those variables.

For more information on the CROSSREF utility program, see CROSSREF Utility Program on page 15-3. For more information on the CROSSREF directive, see CROSSREF and NOCROSSREF on page 11-63.

Protecting SQL Databases

A COBOL85 program that runs as a fault-tolerant process pair cannot update an SQL database, because Tandem NonStop SQL/MP does not support checkpointing of SQL tables and views; however, you can protect SQL databases with Transaction Management Facility (TMF) audited files, which the COBOL85 and NMCOBOL compilers and SQL support. For information on TMF audited files, see the *Transaction Management Facility (TMF) Management and Operations Guide*.

Effect on Conformance to ISO/ANSI Standards

When a Tandem COBOL85 program does not use the SQL directive (which tells the COBOL85 or NMCOBOL compiler to expect embedded SQL), embedded SQL does not affect Tandem COBOL85 conformance to the COBOL85 ISO/ANSI standard. When a program does use the SQL directive, embedded SQL affects Tandem COBOL85 conformance to the COBOL85 ISO/ANSI standard only to the extent required to process SQL statements. For information on Tandem COBOL85 conformance to the COBOL85 ISO/ANSI standard, see Tandem COBOL85 Language on page 1-1.

The Tandem COBOL85 embedded SQL implementation conforms to the ANSI Database—Embedded SQL Standard (ANSI X3.168-1989), with the restrictions and extensions mentioned in this section.

Effect on Performance

Embedded SQL has little or no effect on Tandem COBOL85 compilation-time or run-time performance; however, the PARAM SYMBOL-BLOCKS command affects the amount of space that the COBOL85 or NMCOBOL compiler allocates for each embedded SQL statement (among other things) and the PARAM SYMBOL-BLOCKS command affects compilation time and disk use. For more information on the PARAM SYMBOL-BLOCKS command, see PARAM SYMBOL-BLOCKS on page 11-36.