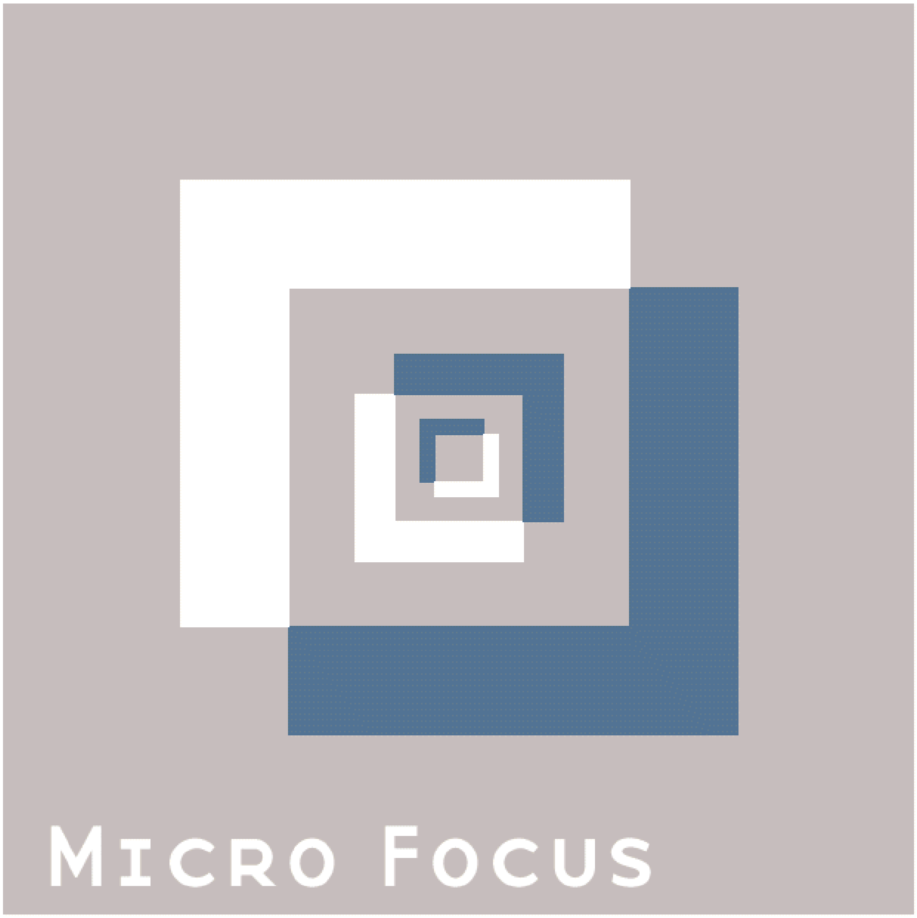
Issue 4, March 2002



Net Express®

Database Access

Micro Focus

Copyright © 2002 Micro Focus International Limited.
All rights reserved.

Micro Focus International Limited has made every effort to ensure that this book is correct and accurate, but reserves the right to make changes without notice at its sole discretion at any time. The software described in this document is supplied under a license and may be used or copied only in accordance with the terms of such license, and in particular any warranty of fitness of Micro Focus software products for any particular purpose is expressly excluded and in no event will Micro Focus be liable for any consequential loss.

Animator®, COBOL Workbench®, EnterpriseLink®, Mainframe Express®, Micro Focus®, Net Express®, REQL® and Revolve® are registered trademarks, and AAI™, Analyzer™, Application to Application Interface™, AddPack™, AppTrack™, AssetMiner™, CCI™, DataConnect™, Dialog System™, EuroSmart™, FixPack™, LEVEL II COBOL™, License Management Facility™, License Server™, Mainframe Access™, Mainframe Manager™, Micro Focus COBOL™, Object COBOL™, OpenESQL™, Personal COBOL™, Professional COBOL™, Server Express™, SmartFind™, SmartFind Plus™, SmartFix™, SourceConnect™, Toolbox™, WebSync™, and Xilerator™ are trademarks of Micro Focus International Limited. All other trademarks are the property of their respective owners.

No part of this publication, with the exception of the software product user documentation contained on a CD-ROM, may be copied, photocopied, reproduced, transmitted, transcribed, or reduced to any electronic medium or machine-readable form without prior written consent of Micro Focus International Limited.

Licensees may duplicate the software product user documentation contained on a CD-ROM, but only to the extent necessary to support the users authorized access to the software under the license agreement. Any reproduction of the documentation, regardless of whether the documentation is reproduced in whole or in part, must be accompanied by this copyright statement in its entirety, without modification.

U.S. GOVERNMENT RESTRICTED RIGHTS. It is acknowledged that the Software and the Documentation were developed at private expense, that no part is in the public domain, and that the Software and Documentation are Commercial Computer Software provided with RESTRICTED RIGHTS under Federal Acquisition Regulations and agency supplements to them. Use, duplication or disclosure by the U.S. Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of The Rights in Technical Data and Computer Software clause at DFAR 252.227-7013 et. seq. or subparagraphs (c)(1) and (2) of the Commercial Computer Software Restricted Rights at FAR 52.227-19, as applicable. Contractor is Micro Focus, 9420 Key West Avenue, Rockville, Maryland 20850. Rights are reserved under copyright laws of the United States with respect to unpublished portions of the Software.

20020417161018

Table of Contents

Part 2: OpenESQL

Part 3: DB2

Part 4: COBSQL

12

Database Access

Preface

This book describes how you can use Net Express to create a COBOL application which uses embedded SQL to access a relational database. Net Express provides:

- OpenESQL
- DB2 ECM
- COBSQL

This book does not provide details about SQL syntax, error messages returned or any use of SQL outside of the COBOL environment. For details of these, refer to the documentation supplied by your database vendor.

Audience

This book is for Net Express COBOL programmers who want to create or modify COBOL applications which access relational databases using embedded SQL.

It is assumed that you are already familiar with both SQL and, if you are using OpenESQL, ODBC. If you are using the DB2 ECM, it is assumed that you are familiar with DB2. If you are using COBSQL, it is assumed that you are familiar with the Oracle or Sybase database that you are accessing.

Notation

The following type styles and conventions have been used in this book:

- Text that you type is shown like this:

  `cat` *`script_name`* | more

  The italic text denotes a variable that you type as part of the command.

- Text in command lines or code examples that is optional is shown in square brackets. In the following example, the expression can specify that the *`column_name`* is like the `pattern_value`, or is not like the *`pattern_value`*, depending on the absence or presence of the optional word `NOT`:

  `column_name` [NOT] LIKE *pattern_value*

- Sections or paragraphs that apply only to a particular database or operating system are identified by a bold italic side heading immediately preceding the paragraph it applies to. For example:

OpenESQL This paragraph only applies to OpenESQL and not to DB2 or COBSQL.

Part 1: Introduction

This part contains the following chapters:

- Chapter 1, “Introduction”
- Chapter 2, “Host Variables”
- Chapter 3, “Data Types”
- Chapter 4, “Cursors”
- Chapter 5, “Data Structures”
- Chapter 6, “Dynamic SQL”

Database Access

1 Introduction

1.1 Overview

Net Express includes a number of SQL preprocessors (OpenESQL, the DB2 ECM and COBSQL) which enable you to access relational databases by embedding SQL statements within your COBOL program:

- OpenESQL

  The OpenESQL preprocessor enables you to access a relational database via an ODBC driver by embedding SQL statements within your COBOL program.

- DB2 ECM

  The DB2 External Checker Module (ECM) is a new type of integrated preprocessor provided with Net Express and designed to work closely with the Micro Focus COBOL Compiler. The DB2 ECM converts embedded SQL statements into the appropriate calls to DB2 database services. It is intended for use with:

  - IBM Software Development Kit (SDK) for Windows 95/NT Version 2.1
  - IBM DB2 for Windows 95/NT Single User Version 2.1
  - IBM Distributed Database Connection Services (DDCS) for Windows NT Version 2.3
  - IBM DB2 Universal Database Version 6.1
  - IBM DB2 Connect Version 5

- COBSQL

  COBSQL is an integrated preprocessor designed to work with COBOL precompilers supplied by relational database vendors. It is intended for use with:

  - Sybase Open Client Embedded SQL/COBOL Version 11.5

- Oracle Pro*COBOL Version 1.8
- Oracle Pro*COBOL Version 8.04
- Informix Embedded SQL/COBOL Version 9.x

If you are already using either of the above precompilers with an earlier version of a Micro Focus COBOL product and want to migrate your application to Net Express, you should use COBSQL. For any other type of embedded SQL development, we recommend that you use OpenESQL.

Note: Use of COBSQL is only supported for standard procedural COBOL programs. You cannot use COBSQL with Object Oriented COBOL syntax (OO programs) or with nested programs.

1.2 Embedded SQL

Each of the preprocessors works by taking the SQL statements that you have embedded in your COBOL program and converting them to the appropriate function calls to the database.

Within your COBOL program, each embedded SQL statement must be preceded by the introductory keywords:

```
EXEC SQL
```

and followed by the keyword:

```
END-EXEC
```

For example:

```
EXEC SQL
   SELECT au_lname INTO :lastname FROM authors
      WHERE au_id = ’124-59-3864’
END-EXEC
```

The embedded SQL statement can be broken over as many lines as necessary following the normal COBOL rules for continuation, but between the EXEC SQL and END-EXEC keywords you can only code an embedded SQL statement, you cannot include any ordinary COBOL code.

Most vendors provide SQL Reference documentation with their database software which will include full information about embedded SQL statements but you should, for example, be able to perform the following typical operations using the statements shown:

Operation	SQL Statement(s)
Add data to a table	INSERT
Change data in a table	UPDATE
Retrieve a row of data from a table	SELECT
Create a named cursor	DECLARE CURSOR
Retrieve multiple rows of data using a cursor	OPEN, FETCH, CLOSE

With the exception of INSERT, DELETE(SEARCHED) and UPDATE(SEARCHED) which are included for your convenience, the embedded SQL statements described here work somewhat differently, or are in addition to, standard SQL statements.

A full syntax description is given in the online help for each of the embedded SQL statements below, together with an example of its use.

Statement	Description
BEGIN DECLARE SECTION	Marks the beginning of a host variable declaration section
BEGIN TRANSACTION[3]	Opens a transaction in AUTOCOMMIT mode
CALL[3]	Executes a stored procedure
CLOSE	Ends row-at-a-time data retrieval initiated by the OPEN statement
COMMIT	Commits a transaction
CONNECT	Connects to a database
DECLARE CURSOR	Defines a cursor for row-at-a-time data retrieval
DECLARE DATABASE	Identifies a database
DELETE (POSITIONED)[1]	Removes the row where the cursor is currently positioned
DELETE (SEARCHED)	Removes table rows that meet the search criteria

Statement	Description
DESCRIBE	Populates an SQLDA data structure
DISCONNECT[2]	Closes connections to one or all databases
END DECLARE SECTION	Marks the end of a host variable declaration section
EXECSP[3]	Executes a stored procedure
EXECUTE	Runs a prepared SQL statement
EXECUTE IMMEDIATE	Runs the SQL statement contained in the specified host variable
FETCH	For a specified cursor, gets the next row from the results set
INCLUDE	Defines a specific SQL data structure for use by an application
INSERT	Adds data to a table or view
OPEN	Begins row-at-a-time data retrieval for a specified cursor
PREPARE	Associates an SQL statement with a name
QUERY ODBC[3]	Queries the ODBC data dictionary
ROLLBACK	Rolls back the current transaction
SELECT DISTINCT	Associates a cursor name with an SQL statement
SELECT INTO[1]	Retrieves one row of results (also known as a singleton select)
SET AUTOCOMMIT[3]	Controls AUTOCOMMIT mode
SET CONCURRENCY[3]	Sets the concurrency option for standard-mode cursors
SET CONNECTION[3]	Specifies which database connection to use for subsequent SQL statements
SET OPTION[3]	Assigns values for query-processing options
SET SCROLLOPTION[3]	Sets the scrolling technique and row membership for standard-mode cursors

Statement	Description
SET TRANSACTION ISOLATION[3]	Sets the transaction isolation level mode for a connection
UPDATE (POSITIONED)[1]	Changes data in the row where the cursor is currently positioned
UPDATE (SEARCHED)	Changes data in existing rows, either by adding new data or by modifying existing data
WHENEVER	Specifies the default action (CONTINUE, GOTO or PERFORM) to be taken after a SQL statement is run

Notes:

[1] These statements have the same name as a standard SQL statement but the syntax given in the online help augments the standard SQL syntax

[2] The DISCONNECT statement is not supported when accessing an Oracle database via COBSQL

[3] These statements are not supported by COBSQL

1.2.1 Case

The case of embedded SQL keywords in your programs is ignored, for example:

```
EXEC SQL CONNECT exec sql connect Exec Sql Connect
```

are all equivalent.

The case of cursor names, statement names and connection names must match that used when the variable is declared. For example, if you declare a cursor as C1, you must always refer to it a C1 (and not as c1).

The settings for the particular database determine whether other words, such as table and column names, are case-sensitive.

Hyphens are not permitted in SQL identifiers (in table and column names, for example).

1.2.2 OpenESQL Assistant

Net Express includes an OpenESQL Assistant. You can use this interactive tool to:

- Prototype SQL SELECT statements and test them against your database
- Design SQL INSERT, UPDATE and DELETE statements

For further information, refer to the chapter *OpenESQL*.

1.3 Building your Application

Once you have written your COBOL application containing embedded SQL, you must compile it specifying the appropriate Compiler directive such that the preprocessor converts the embedded SQL statements into function calls to the database:

- OpenESQL preprocessor:

  Specify the SQL Compiler directive. See the chapter *OpenESQL* for details.

- DB2 ECM preprocessor:

  Specify the DB2 Compiler directive. See the chapter *DB2* for details.

- COBSQL preprocessor:

  Specify the PREPROCESS"COBSQL" Compiler directive. See the chapter *COBSQL* for details.

1.3.1 Internet Application Wizard

Net Express includes an Internet Application Wizard. Use this wizard to generate complete Web applications that access a relational database. You can create a working SQL application within minutes.

For further information, refer to the on-line book ***Internet Applications***.

1.4 Multiple Program Modules

Multiple embedded SQL source files, compiled separately and linked to a single executable file, can share the same database connection at run time. This is also true for programs that are compiled into separate dynamic-link libraries (.dll files). If subsequent program modules (in the same process) do not process a CONNECT statement, they share the same database connection with the module that included the CONNECT statement.

The table below gives guidelines on how to use multiple program modules with the different SQL preprocessors:

SQL preprocessors	**Guidelines**
OpenESQL	In a program that includes multiple, separately compiled modules, you should compile only one module with the INIT option of the SQL Compiler directive. All other modules within the program should share that first automatic connection or make explicit connections using the CONNECT statement.
OpenESQL, DB2	Statement names are local to a particular program module (compilation unit). This means that a statement cannot be prepared in one module and executed in another.
OpenESQL, DB2	Cursor names should be unique within an application
COBSQL	If you specify the INIT directive more than once, Net Express ignores second and subsequent uses

Database Access

2 Host Variables

Host variables are data items defined within a COBOL program. They are used to pass values to and receive values from a database. Host variables can be defined in the File Section, Working-Storage Section, Local-Storage Section or Linkage Section of your COBOL program and have any level number between 1 and 48. Level 49 is reserved for VARCHAR data items.

When a host variable name is used within an embedded SQL statement, the data item name must begin with a colon (:) to enable the Compiler to distinguish between host variables and tables or columns with the same name.

Host variables are used in one of two ways:

- Input host variables

  These are used to specify data that will be transferred from the COBOL program to the database.

- Output host variables

  These are used to hold data that is returned to the COBOL program from the database.

For example, in the following statement, `:book-id` is an input host variable that contains the ID of the book to search for, while `:book-title` is an output host variable that returns the result of the search:

```
EXEC SQL
   SELECT title INTO :book-title FROM titles
      WHERE title_id=:book-id
END-EXEC
```

2.1 Declaring Host Variables

Before you can use a host variable in an embedded SQL statement, you must declare it. Host variable declarations should be bracketed by the embedded SQL statements BEGIN DECLARE SECTION and END DECLARE SECTION, for example:

```
EXEC SQL
   BEGIN DECLARE SECTION
END-EXEC
01 id             pic x(4).
01 name           pic x(30).
EXEC SQL
   END DECLARE SECTION
END-EXEC

display "Type your identification number: "
accept id.

* The following statement retrieves the name of the
* employee whose ID is the same as the contents of
* the host variable "id". The name is returned in
* the host variable "name".

EXEC SQL
   SELECT emp_name INTO :name FROM employees
      WHERE emp_id=:id
END-EXEC
display "Hello " name.
```

Note:

- You can use groups of data items as a single host variable.
- OpenESQL trims trailing spaces from character host variables. If the variable consists entirely of spaces, OpenESQL does not trim the first space character because some servers treat a zero length string as NULL.

2.1.1 OpenESQL and DB2 Preprocessors

You can use data items as host variables even if they have not been declared using BEGIN DECLARE SECTION and END DECLARE SECTION.

When declaring host variables, you should bear the following in mind:

- Host variable names must conform to the COBOL rules for data items.
- Host variables can be declared anywhere that it is legal to declare COBOL data items.
- Underscores (_) are not permitted in host variable names.

2.2 Host Arrays

An array is a collection of data items associated with a single variable name. You can define an array of host variables (called host arrays) and operate on them with a single SQL statement.

You can use host arrays as input variables in INSERT, UPDATE and DELETE statements and as output variables in the INTO clause of SELECT and FETCH statements. This means that you can use arrays with SELECT, FETCH, DELETE, INSERT and UPDATE statements to manipulate large volumes of data.

Host arrays are declared in the same way as simple host variables using BEGIN DECLARE SECTION and END DECLARE SECTION. With host arrays, however, you must use the OCCURS clause to dimension the array. For example:

```
EXEC SQL
   BEGIN DECLARE SECTION
END-EXEC
01 AUTH-REC-TABLES
   05 Auth-id       OCCURS 25 TIMES PIC X(12).
   05 Auth-Lname    OCCURS 25 TIMES PIC X(40).
EXEC SQL
   END DECLARE SECTION
END-EXEC.
.
.
```

```
.
EXEC SQL
   CONNECT USERID ’user’ IDENTIFIED BY ’pwd’ USING ’db_alias’
END-EXEC
EXEC SQL
   SELECT au-id, au-lname
      INTO :Auth-id, :Auth-Lname FROM authors
END-EXEC
display sqlerrd(3)
```

In this example, up to 25 rows (the size of the array) can be returned by the SELECT statement. If the SELECT statement could return more than 25 rows, then 25 rows will be returned and SQLCODE will be set to indicate that more rows are available but could not be returned.

Use a SELECT statement only when you know the maximum number of rows to be selected. When the number of rows to be returned is unknown, use the FETCH statement. Using arrays, it is possible to fetch data in batches. This can be useful when creating a scrolling list of information.

If you use multiple host arrays in a single SQL statement, their dimensions must be the same.

Notes:

OpenESQL and DB2

- You cannot mix host arrays and simple host variables within a single SQL statement. If any of the host variables is an array, they must all be arrays.

COBSQL

- This information on host arrays applies in its entirety if you are using an Oracle database. If you are using a Sybase database, however, you can only use host arrays as output variables in either a SELECT or a FETCH statement.
- Both Oracle and Sybase allow a simple host variable in the WHERE clause of a SELECT statement. This is the only time that you can mix array host variables and simple host variables.

2.2.1 The FOR Clause

By default, the entire array is processed by an SQL statement but you can use the optional FOR clause to limit the number of array elements processed to just those that you want. This is especially useful in UPDATE, INSERT and DELETE statements where you may not want to use the entire array.

The FOR clause must use an integer host variable, for example:

```
EXEC SQL
   BEGIN DECLARE SECTION
END-EXEC
01 AUTH-REC-TABLES
   05 Auth-id       OCCURS 25 TIMES PIC X(12).
   05 Auth-Lname    OCCURS 25 TIMES PIC X(40).
01 maxitems         PIC S9(4) COMP-5 VALUE 10.
EXEC SQL
   END DECLARE SECTION
END-EXEC.
.
.
.
EXEC SQL
   CONNECT USERID ’user’ IDENTIFIED BY ’pwd’ USING ’db_alias’
END-EXEC
EXEC SQL
   FOR :maxitems
      UPDATE   authors
         SET   au_lname = :Auth_Lname
         WHERE au_id  = :Auth_id
END-EXEC
display sqlerrd(3)
```

In this example, 10 rows (the value of `:maxitems`) are modified by the UPDATE statement.

The number of array elements processed is determined by comparing the dimension of the host array with the FOR clause variable. The lesser value is used.

If the value of the FOR clause variable is less than or equal to zero, no rows are processed.

Note:

COBSQL Preprocessor

If you are using COBSQL, this information on the FOR clause is only applicable if you are using an Oracle database. It does not apply if you are using either a Sybase or an Informix database.

2.2.2 Determining the Number of Rows Processed

The third element of `SQLERRD` in the SQLCA, `SQLERRD(3)`, records the number of rows processed for INSERT, UPDATE, DELETE and SELECT INTO statements.

OpenESQL Preprocessor

For FETCH statements, SQLERRD(3) always contains the number of rows fetched by the last FETCH statement.

COBSQL and DB2 Preprocessors

For FETCH statements, SQLERRD(3) records the cumulative sum of rows processed.

2.2.2.1 The DB2 Preprocessor

SQLERRD(3) contains the following:

- If PREPARE is invoked and successful, an estimate of the number of rows that will be returned.
- After INSERT, UPDATE and DELETE, the actual number of rows affected.
- If compound SQL is invoked, a count of the number of successful sub-statements.
- If CONNECT is invoked, either 1 if the database can be updated or 2 if the database is read-only.
- If an error occurs when processing a host array, the last row that was processed successfully.

SQLERRD(4) contains the following:

- If PREPARE is invoked and successful, a relative cost estimate of the resources required to process the statement.
- If compound SQL is invoked, a count of the number of successful sub-statements.
- If CONNECT is invoked, 0 for a one-phase commit from a down-level client; 1 for a one-phase commit; 2 for a one-phase, read-only commit; 3 for a two-phase commit.

SQLERRD(5) contains the following:

- The total number of rows deleted, inserted, or updated as a result of both:
    - The enforcement of constraints after a successful delete operation.
    - The processing of triggered SQL statements from activated triggers.
- If compound SQL is invoked, an accumulation of the number of such rows for all sub-statements. In some cases, when an error is encountered, this field contains a negative value that is an internal error pointer.
- If CONNECT is invoked, an authentication type value of 0 for a server authentication; 1 for client authentication; 2 for authentication using DB2 Connect; 3 for DCE security services authentication; 255 for unspecified authentication.

2.3 Indicator Variables

Embedded SQL enables you to store and retrieve null values from a database by using indicator variables. Indicator variables are always defined as:

```
pic S9(4) comp-5.
```

2.3.1 Null Values

Unlike COBOL, SQL supports variables that can contain null values. A null value means that no entry has been made and usually implies that the value is either unknown or undefined. A null value enables you to distinguish between a deliberate entry of zero (for numerical columns) or a blank (for character columns) and an unknown or inapplicable entry. For example, a null value in a price column does not mean that the item is being given away free, it means that the price is not known or has not been set.

Together, a host variable and its companion indicator variable specify a single SQL value. Both variables must be preceded by a colon (:). When a host variable is null, its indicator variable has the value -1; when a host variable is not null, the indicator variable has a value other than -1.

Within an embedded SQL statement an indicator variable should be placed immediately after its corresponding host variable. For example, the following embedded UPDATE statement uses a `saleprice` host variable with a companion indicator variable, `saleprice-null`:

```
EXEC SQL
   UPDATE closeoutsale
      SET temp_price = :saleprice:saleprice-null,
           listprice = :oldprice
END-EXEC
```

In this example, if `saleprice-null` has a value of -1, when the UPDATE statement executes, the statement is read as:

```
EXEC SQL
   UPDATE closeoutsale
      SET temp_price = null, listprice = :oldprice
END-EXEC
```

You cannot use indicator variables in a search condition. To search for null values, use the `is null` construct. For example, you can use the following:

```
     if saleprice-null equal -1
         EXEC SQL
             DELETE FROM closeoutsale
                    WHERE temp_price is null
         END-EXEC
     else
         EXEC SQL
```

```
             DELETE FROM closeoutsale
                    WHERE temp_price = :saleprice
         END-EXEC
     end-if
```

2.3.2 Data Truncation

Indicator variables serve an additional purpose if truncation occurs when data is retrieved from a database into a host variable. If the host variable is not large enough to hold the data returned from the database, the warning flag `sqlwarn1` in the SQLCA data structure is set and the indicator variable is set to the size of the data contained in the database.

2.3.3 Indicator Arrays

Use indicator arrays in the same ways that you can use indicator variables, that is:

- To assign null values.
- To detect null values.
- To detect data truncation.

In the following example, an indicator array is set to -l so that it can be used to insert null values into a column:

```
EXEC SQL
   BEGIN DECLARE SECTION
END-EXEC
01 sales-id       OCCURS 25 TIMES PIC X(12).
01 sales-name     OCCURS 25 TIMES PIC X(40).
01 sales-comm     OCCURS 25 TIMES PIC S9(9) COMP-5.
01 ind-comm       OCCURS 25 TIMES PIC S9(4) COMP-5.
EXEC SQL
   END DECLARE SECTION
END-EXEC.
.
.
.
MOVE -1 TO ind-comm.
.
```

```
.
.
EXEC SQL
   INSERT INTO SALES (ID, NAME, COMM)
      VALUES (:sales_id, :sales_name, :sales_comm:ind-comm)
END-EXEC
```

COBSQL If you are using COBSQL, this information on indicator arrays is only applicable if you are using an Oracle database. It does not apply if you are using a Sybase database.

3 Data Types

SQL data types differ from those used in COBOL.

SQL has a standard set of data types, but the exact implementation of these varies between databases and many databases do not implement the full set.

3.1 Converting Data Types

Within your COBOL program a host variable can act as a COBOL program variable and as a SQL database variable and so the preprocessor must convert, or map, COBOL data types to the appropriate SQL data type. This means that you need to declare your host variables with the correct COBOL picture clause so that the preprocessor maps it to the correct SQL data type. To do this, you need to know the SQL data types used by the data source to which you are going to connect.

The following sections describe the different SQL data types and how to declare host variables that map directly onto them.

3.1.1 COBSQL Preprocessor

When using Sybase, Informix or Oracle with COBSQL, the database engine is able to perform some sort of conversion to change the data from a COBOL data type to a database data type. A general rule of thumb is that for numeric or integer data types host variables should be defined as:

```
PIC S9(..)..COMP..
```

while character or text data types should be defined as:

```
PIC X(...).
```

Both Oracle and Sybase allow you to define the database data type for a given host variable. This can be useful for the more complex data types.

3.1.1.1 Oracle

For Oracle, this is done as follows:

```
EXEC SQL
BEGIN DECLARE SECTION
END-EXEC.
*
* Define data item as Oracle data type DISPLAY
*
01 emp-comm pic s9(6)v99 DISPLAY SIGN LEADING SEPARATE
*
EXEC SQL
   VAR emp-comm IS DISPLAY(8,2)
END-EXEC.
EXEC SQL
   END DECLARE SECTION
END-EXEC.
```

3.1.1.2 Sybase

For Sybase, this is done as follows:

```
EXEC SQL
   BEGIN DECLARE SECTION
END-EXEC.
*
* Define item as Sybase specific data type
*
01 money-item CS-MONEY.
*
EXEC SQL
   END DECLARE SECTION
END-EXEC.
```

For more information about defining the database type of a host variable, refer to the COBOL precompiler manual supplied by your database vendor.

3.1.1.3 Informix

Informix provides a number system routines that you can call to manipulate different data types. For more information about these routines, please refer to the ***Programming with INFORMIX-ESQL/COBOL*** manual.

3.2 Integer Data Types

3.2.1 Tiny Integer

A tiny integer (TINYINT) is a 1-byte integer SQL data type that can be declared in COBOL as

```
PIC S9(4) COMP-5.
```

Note: DB2 does not support the tiny integer data type.

3.2.1.1 COBSQL Preprocessor

Sybase supports the use of tiny integer host variables. The definition for Sybase is:

```
03 tinyint1     PIC S9(2) COMP-5.
03 tinyint2     PIC S9(2) COMP.
03 tinyint3     PIC S9(2) BINARY.
```

These map onto the Sybase data type TINYINT.

3.2.2 Small Integer

A small integer (SMALLINT) is a 2-byte integer SQL data type that can be declared in COBOL with usage BINARY, COMP, COMP-X, COMP-5 or COMP-4.

For example, all of the following definitions are valid for host variables to map directly onto the SMALLINT data type.

```
03 shortint1      PIC S9(4)  COMP.
03 shortint2      PIC S9(4)  BINARY.
03 shortint3      PIC X(2)   COMP-5.
03 shortint4      PIC S9(4)  COMP-4.
03 shortint5      PIC 9(4)   USAGE DISPLAY.
03 shortint6      PIC S9(4)  USAGE DISPLAY.
```

3.2.2.1 OpenESQL Preprocessor

OpenESQL currently supports signed small integers, but not unsigned small integers.

3.2.2.2 COBSQL Preprocessor - Oracle

With Oracle, it is best to define the host variable as `shortint1` or `shortint2` or as:

```
03 shortint7     PIC S9(4) COMP-5.
```

These map onto the Oracle data type NUMBER(38).

3.2.2.3 COBSQL Preprocessor - Sybase

With Sybase, all except `shortint3` should be accepted. You can also use:

```
03 shortint7     PIC S9(4) COMP-5.
```

These map onto the Sybase data type SMALLINT.

3.2.2.4 COBSQL Preprocessor - Informix

With Informix, it is best to define the host variable as `shortint1` or `shortint2` or as:

```
03 shortint7     PIC S9(4) COMP-5.
```

These map onto the Informix data type SMALLINT.

3.2.3 Integer

An integer (INT) is a 4-byte integer SQL data type that can be declared in COBOL with usage BINARY, COMP, COMP-X, COMP-5 or COMP-4.

All of the following definitions are valid for host variables to map directly onto the INT data type.

```
03 longint1      PIC S9(9)  COMP.
03 longint2      PIC S9(9)  COMP-5.
03 longint3      PIC X(4)   COMP-5.
03 longint4      PIC X(4)   COMP-X.
03 longint5      PIC 9(9)   USAGE DISPLAY.
03 longint6      PIC S9(9)  USAGE DISPLAY.
```

3.2.3.1 OpenESQL Preprocessor

OpenESQL currently supports signed integers, but not unsigned integers.

3.2.3.2 COBSQL Preprocessor - Oracle

With Oracle, it is best to define integer host variables as `longint1`, `longint2` or as:

```
03 longint7     PIC S9(8) COMP-5.
```

These map to the Oracle data type NUMBER(38).

3.2.3.3 COBSQL Preprocessor - Sybase

With Sybase, all except `longint3` should be accepted. You can also use:

```
03 longint7     PIC S9(8) COMP-5.
```

These map to the Sybase data type INT.

3.2.3.4 COBSQL Preprocessor - Informix

With Informix, it is best to define integer host variables as `longint1`, `longint2` or as:

```
03 longint7     PIC S9(9) COMP-5.
```

These map to the Informix data type INT.

3.2.4 Big Integer

A big integer (BIGINT) is an 8-byte integer SQL data type that can be declared in COBOL as:

```
PIC S9(18) COMP-3.
```

Notes:

DB2 Preprocessor DB2 does not support the big integer data type.

COBSQL Preprocessor Oracle, Sybase and Informix do not support big integers.

OpenESQL Preprocessor OpenESQL supports a maximum size of S9(18) for COBOL data items used as host variables to hold values mapped from the SQL data type BIGINT. However, a BIGINT data type can hold a value that is larger than the maximum value that can be held in a `PIC S9(18)` data item; therefore, ensure that your code checks for data truncation.

3.3 Character Data Types

3.3.1 Fixed-length Character Strings

Fixed-length character strings (CHAR) are SQL data types with a driver defined maximum length. They are declared in COBOL as PIC X(*n*) where *n* is an integer between 1 and the maximum length.

For example:

```
03 char-field1      pic x(5).
03 char-field2      pic x(254).
```

COBSQL **Note:** This maps to the Oracle data type CHAR(*n*), to the Sybase data type CHAR(*n*) and to the Informix data type CHAR(*n*). For both Oracle and Sybase, the largest supported fixed length character string is 255 bytes. For Informix, the largest supported fixed length character is 32KB.

3.3.2 Variable-length Character Strings

3.3.2.1 OpenESQL and DB2 Preprocessors

Variable-length character strings (VARCHAR) are SQL data types that can be declared in COBOL in one of two ways:

- As fixed-length character strings (PIC X(*n*)).
- As group items containing only two elementary items, both of which must have a level number of 49. The first item is a 2-byte field declared with usage COMP or COMP-5 that represents the effective length of the character string. The second item is a PIC X(*n*) data type, where *n* is an integer, and holds the actual data.

For example:

```
03 varchar1.
   49 varchar1-len        pic 9(4) comp-5.
   49 varchar1-data       pic x(200).
03 Longvarchar1.
   49 Longvarchar1-len    pic 9(4) comp.
   49 Longvarchar1-data   pic x(30000).
```

If the data being copied to a SQL CHAR, VARCHAR or LONG VARCHAR data type is longer than the defined length, then the data is truncated and the SQLWARN1 flag in the SQLCA data structure is set. If the data is smaller than the defined length, a receiving CHAR data type may be padded with blanks.

3.3.2.2 COBSQL Preprocessor

3.3.2.2.1 Oracle

For Oracle, the host variable is defined using the Oracle keyword VARYING. An example of its use is as follows:

```
EXEC SQL
   BEGIN DECLARE SECTION
END-EXEC.
01 USERNAME     PIC X(20) VARYING.
EXEC SQL
   END DECLARE SECTION
END-EXEC.
```

Oracle will then expand the data item USERNAME into the following group item:

```
01 USERNAME
   02 USERNAME-LEN     PIC S9(4) COMP-5.
   02 USERNAME-ARR     PIC X(20).
```

Within the COBOL code, references must be made to either USERNAME-LEN or USERNAME-ARR but within SQL statments the group name USERNAME must be used. For example:

```
move "SCOTT" to USERNAME-ARR.
move 5 to USERNAME-LEN.
exec sql
   connect :USERNAME identified by :pwd
   using :db-alias
end-exec.
```

This maps to the Oracle data type VARCHAR(*n*) or VARCHAR2(*n*). For very large character items, Oracle provides the data type LONG.

3.3.2.2.2 Sybase

For Sybase, the host variable must be defined with a PIC X(*n*) picture clause as the Sybase precompiler does not support the use of group items to handle VARCHAR SQL data types.

These map to the Sybase data type of VARCHAR(*n*).

3.3.2.3 COBSQL - Informix

For Informix, the host variable must be defined with a PIC X(*n*) picture clause as the Informix precompiler does not support the use of group items to handle VARCHAR SQL data types.

These map to the Informix data type of VARCHAR(*n*). The maximum length of a VARCHAR field depends on the version of Informix being used. For more information on VARCHAR data items, please refer to the ***Informix Guide to SQL*** manual.

3.4 Approximate Numeric Data Types

The 32-bit SQL floating-point data type, REAL, is declared in COBOL as usage COMP-2.

The 64-bit SQL floating-point data types, FLOAT and DOUBLE, are declared in COBOL as usage COMP-2.

For example:

```
01 float1      usage comp-2.
```

3.4.1 OpenESQL Preprocessor

Both 32-bit and 64-bit floating-point data types are mapped to COMP-2 COBOL data items because single-precision floating point is not supported in embedded SQL by OpenESQL.

3.4.2 DB2 Preprocessor

DB2 Universal Database supports single-precision floating point (REAL) as COMP-1 and double-precision floating point (FLOAT or DOUBLE) as COMP-2.

DB2 Version 2.1 only supports double-precision floating point (FLOAT or DOUBLE) as COMP-2.

3.4.3 COBSQL Preprocessor

Oracle supports the use of both COMP-1 and COMP-2 data items. These both map to the Oracle data type NUMBER.

Sybase supports the use of both COMP-1 and COMP-2 data items. COMP-1 data items map to the Sybase data type REAL. COMP-2 data items map to the Sybase data type FLOAT.

Informix does not support either COMP-1 or COMP-2 data items. Informix only supports fixed numeric data items in COBOL, that is `PIC`

S9(*m*)V9(*n*). Informix will convert FLOAT and SMALLFLOAT SQL columns into this format.

3.5 Exact Numeric Data Types

The exact numeric data types DECIMAL and NUMERIC can hold values up to a driver-specified precision and scale.

They are declared in COBOL as COMP-3, PACKED-DECIMAL or as NUMERIC USAGE DISPLAY.

For example:

```
03 packed1      pic s9(8)v9(10) usage comp-3.
03 packed2      pic s9(8)v9(10) usage display.
```

3.5.1 COBSQL Preprocessor

For Oracle, these map to the data type NUMBER(*p*,*s*). For Sybase, they map to either NUMBER(*p*,*s*) or to DECIMAL(*p*,*s*). For Informix, they map to either DECIMAL(*p*,*s*) or to MONEY(*p*,*s*).

- For more information on the difference between the NUMERIC and DECIMAL data types, refer to the chapter on *Using and Creating Datatypes* in the ***Sybase Transact-SQL Users Guide***
- For more information on the difference between the DECIMAL and MONEY data types, refer to the Data Types chapter in the Informix Guide to SQL.

3.6 Date and Time Data Types

COBOL does not have date/time data types so SQL date/time columns are converted to character representations.

If a COBOL output host variable is defined as PIC X(*n*), for a SQL timestamp value, where *n* is greater than or equal to 19, the date and time will be specified in the format *yyyy-mm-dd hh:mm:ss.ff...*, where the number of fractional digits is driver defined.

For example:

```
1994-05-24 12:34:00.000
```

3.6.1 DB2 Preprocessor

For DB2, the TIMESTAMP data type has a maximum length of 26 characters.

3.6.2 COBSQL Preprocessor

3.6.2.1 Oracle

Oracle date items have a unique data definition and Oracle provides functions to convert date, time and datetime fields when used within a COBOL program. These functions are:

- TO_CHAR

  Converts from Oracle's date format to a character string.

- TO_DATE

  Converts a character string into an Oracle date.

Both functions take an item to be converted followed by the date, time or datetime mask to be applied to that data item. An example of this is as follows:

```
exec sql
   select ename, TO_CHAR(hiredate, ’DD-MM-YYYY’)
      from emp
      into :ename, :hiredate
      where empno = :empno
end-exec.

exec sql
   insert into emp (ename, TO_DATE(hiredate, ’DD-MM-YYYY’))
          values   (:ename, :hiredate)
end-exec.
```

This maps to the Oracle data type of DATE. For more information about the DATE data type, refer to the Oracle ***SQL Language Reference Manual***. More information about the use of functions within Oracle SQL statements can be found in this manual.

3.6.2.2 Sybase

Sybase provides a function, called convert, to change the format of a data type. Using the Oracle examples above, the SQL syntax would be:

```
exec sql
   select ename, convert(varchar(12) hiredate, 105)
      from emp
      into :ename, :hiredate
      where empno = :empno
end-exec.

exec sql
   insert into emp (ename, hiredate)
          values   (:ename, convert(datetime :hiredate, 105)
end-exec.
```

This maps to the Sybase data type of either SMALLDATETIME or DATETIME. For more information on the difference between the SMALLDATETIME and the DATETIME data types, refer to the chapter *Using and Creating Datatypes* in the Sybase ***Transact-SQL User’s Guide***.

For more information on the Sybase convert function, refer to the Sybase ***SQL Server Reference Manual: Volume 1 Commands, Functions and Topics***.

3.6.2.3 Informix

Informix expects dates to either be in Julian format, or of the format *mm/dd/yyyy*:

- When using Julian dates, define the field as `PIC S9(9) COMP`.
- For dates in the format *mm/dd/yyyy*:
    - define the COBOL field as `PIC X(10)`.
    - use the DATE_TYPE function.

For more information on passing dates to Informix, refer to the ***INFORMIX-ESQL/COBOL Programmer's Manual***.

3.7 Binary Data Types

3.7.1 OpenESQL Preprocessor

SQL BINARY, VARBINARY and IMAGE data are represented in COBOL as PIC X (*n*) fields. No data conversion is performed. When data is fetched from the database, if the COBOL field is smaller than the amount of data, the data is truncated and the SQLWARN1 field in the SQLCA data structure is set to "W". If the COBOL field is larger than the amount of data, the field is padded with null (x"00") bytes. To insert data into BINARY, VARBINARY or LONG VARBINARY columns, you must use dynamic SQL statements.

3.7.2 DB2 Preprocessor

With DB2, use CHAR FOR BIT DATA to represent BINARY, VARCHAR(*n*) FOR BIT DATA to represent VARBINARY and LONG VARCHAR FOR BIT DATA to represent LONG VARBINARY. If you use the IBM ODBC driver, BINARY, VARBINARY and LONG VARBINARY are the data types returned instead of the IBM equivalent. The IMAGE data type can be represented by BLOB. DB2 uses LOBs (Character Large Object, Binary Large Object or

Graphical Large Object to define very large columns (2 Gigabytes maximum). You can use static SQL with these data types.

3.7.3 COBSQL Preprocessor

3.7.3.1 Oracle

Oracle provides support for binary data. The difference between binary and character data is that Oracle will do codeset conversions on character data, but will leave binary data untouched.

The two Oracle data types are RAW and LONG RAW. There are some restrictions on the use of RAW and LONG RAW - consult your Oracle documentation for further details.

3.7.3.2 Sybase

Sybase provides three binary data types: BINARY, VARBINARY and IMAGE. IMAGE is a complex data type and as such, host variables can be defined as CS-IMAGE, for example:

```
EXEC SQL
   BEGIN DECLARE SECTION
END-EXEC.
*
* Define item as Sybase specific data type.
*
01 image-item     CS-IMAGE.
*
EXEC SQL
   END DECLARE SECTION
END-EXEC.
```

Note: For more information on using the Sybase data types of BINARY, VARBINARY and IMAGE, please refer to the chapter *Using and Creating Datatypes* in the ***Sybase Transact-SQL User's Guide***.

3.7.3.3 Informix

Informix supports two types of Binary data items, TEXT and BYTE. These data types do not store the actual data; in fact, they are file names. Therefore, in COBOL, the corresponding item is a PIC X(*n*).

For more information on TEXT and BYTE data items, refer to the ***Informix Guide to SQL***.

4 Cursors

Where the result returned by a SELECT statement includes more than one row of data, that is the results set, you must declare and use a cursor. A cursor indicates the current position in a results set, in the same way that the cursor on a screen indicates the current position.

A cursor enables you to:

- Fetch rows of data one at a time
- Perform updates and deletions at a specified position within a results set.

The example below demonstrates the following sequence of events:

1. The DECLARE CURSOR statement associates the SELECT statement with the cursor `Cursor1`.
2. The OPEN statement opens the cursor, thereby executing the SELECT statement.
3. The FETCH statement retrieves the data for the current row from the columns `au_fname` and `au_lname` and places the data in the host variables `first_name` and `last_name`.
4. The program loops on the FETCH statement until no more data is available.
5. The CLOSE statement closes the cursor.

```
EXEC SQL DECLARE Cursor1 CURSOR FOR
   SELECT au_fname, au_lname FROM authors
END-EXEC
...
EXEC SQL
   OPEN Cursor1
END-EXEC
...
perform until sqlcode not = zero
   EXEC SQL
      FETCH Cursor1 INTO :first_name,:last_name
   END-EXEC
      display first_name, last_name
```

```
end-perform
...
EXEC SQL
   CLOSE Cursor1
END-EXEC
```

4.1 Declaring a Cursor

Before a cursor can be used, it must be declared. This is done using the DECLARE CURSOR statement in which you specify a name for the cursor and either a SELECT statement or the name of a prepared SQL statement.

Cursor names must conform to the rules for identifiers on the database that you are connecting to, for example, some databases do not allow hyphens in cursor names.

```
EXEC SQL
    DECLARE Cur1 CURSOR FOR
        SELECT first_name FROM employee
         WHERE last_name = :last-name
END-EXEC
```

This example specifies a SELECT statement using an input host variable (:last-name). When the cursor OPEN statement is executed, the values of the input host variable are read and the SELECT statement is executed.

```
EXEC SQL
    DECLARE Cur2 CURSOR FOR stmt1
END-EXEC
...
move "SELECT first_name FROM emp " &
     "WHERE last_name=?" to prep.
EXEC SQL
   PREPARE stmt1 FROM :prep
END-EXEC
...
EXEC SQL
   OPEN Cur2 USING :last-name
END-EXEC
```

In this example, the DECLARE CURSOR statement references a prepared statement (`stmt1`). A prepared SELECT statement can contain question marks (?) which act as parameter markers to indicate that data is to be supplied when the cursor is opened. The cursor must be declared before the statement is prepared.

COBSQL A cursor can be declared in either the data division or the procedure division of your program. The DECLARE CURSOR statement does not generate any code but if a cursor is declared within the procedure division, COBSQL generates an animation breakpoint for the DECLARE CURSOR statement.

4.1.1 Object Oriented COBOL Syntax

Within an Object Oriented (OO) program, you can declare a cursor anywhere that it is valid to declare a data item. Cursors are local to the object that they are opened in, that is, two instances of an object opening the "same" cursor each get their own cursor instance.

You can open a cursor in one method, fetch it in a second and close it in a third but it must be declared in object-storage if you want to do this.

Notes:

COBSQL

- Object oriented COBOL syntax is not supported by COBSQL.
- Some versions of the Informix precompiler can generate incorrect code if a cursor is declared within the Working-Storage section. We therefore recommend that you only declare cursors in the procedure division when using INFORMIX.

4.2 Opening a Cursor

Once a cursor has been declared, it must be opened before it can be used. This is done using the OPEN statement, for example:

```
EXEC SQL
   OPEN Cur1
END-EXEC
```

If the DECLARE CURSOR statement references a prepared statement that contains parameter markers, the corresponding OPEN statement must specify the host variables or the name of an SQLDA structure that will supply the values for the parameter markers, for example:

```
EXEC SQL
   OPEN Cur2 USING :last-name
END-EXEC
```

If an SQLDA data structure is used, the data type, length, and address fields must already contain valid data when the OPEN statement is executed.

COBSQL When a cursor is opened, no locks are applied to the tables the data is being selected from.

COBSQL An Oracle database allows a cursor to be re-opened before it is closed such that the SELECT statement is re-evaluated. If the program has been compiled in ANSI mode, however, re-opening the cursor before it has been closed generates an error. For more information on the MODE precompiler directive, refer to the ***Programmer's Guide to the ORACLE Precompilers***.

4.3 Using a Cursor to Retrieve Data

Once a cursor has been opened, it can be used to retrieve data from the database. This is done using the FETCH statement. The FETCH statement retrieves the next row from the results set produced by the OPEN statement and writes the data returned to the specified host variables (or to addresses specified in an SQLDA structure). For example:

```
perform until sqlcode not = 0
    EXEC SQL
       FETCH Cur1 INTO :first_name
    END-EXEC
    display ’First name: ’ fname
    display ’Last name : ’ lname
    display spaces
end-perform
```

When the cursor reaches the end of the results set, a value of 100 is returned in SQLCODE in the SQLCA data structure and SQLSTATE is set to "02000".

As data is fetched from a cursor. locks can be placed on the tables the data is being selected from. For more information about the different types of cursors, the locked data they can read and the locks they put on data, see the section *Cursor Options* below.

COBSQL The ORACLE precompiler directive, MODE, affects the value put into SQLCODE when no data is found. For more information on the use of the MODE precompiler directive, refer to the ***Programmer’s Guide to the ORACLE Precompilers***.

4.4 Closing a Cursor

When your application has finished using the cursor, it should be closed using the CLOSE statement. For example:

```
EXEC SQL
   CLOSE Cur1
END-EXEC
```

Normally, when a cursor is closed, all locks on data and tables are released. If the cursor is closed within a transaction, however, the locks may not be released.

COBSQL The ORACLE precompiler directive, MODE, affects what happens to a cursor when either the commit or rollback command is used. For more information on the use of the precompiler directive MODE, refer to the ***Programmer's Guide to the ORACLE Precompilers***.

COBSQL When a cursor is closed, the ORACLE client may deallocate the memory and resources associated with the cursor. The following precompiler optins control the deallocatin of cursors: HOLD_CURSOR, MAXOPENCURSORS and RELEASE_CURSOR. For more information on the use of the precompiler directives, refer to the ***Programmer's Guide to the ORACLE Precompilers***.

4.5 Cursor Options

OpenESQL The information given here on cursor options is only applicable to OpenESQL.

The behavior and performance of cursors can be tuned using the following embedded SQL statements:

Embedded SQL Statement	**Description**
SET SCROLLOPTION	Selects how the row membership of the results set of a cursor is determined.

Embedded SQL Statement	Description
SET CONCURRENCY	Activate concurrency control. (With concurrent access, data would soon become unreliable without some kind of control.). Use this statement before the cursor is opened.

Note: SET SCROLLOPTION and SET CONCURRENCY are part of the Extended SQL Syntax and are not supported by all ODBC drivers.

4.6 Positioned UPDATE and DELETE Statements

Positioned UPDATE and DELETE statements are used in conjunction with cursors and include WHERE CURRENT OF clauses instead of search condition clauses. The WHERE CURRENT OF clause specifies the corresponding cursor.

```
EXEC SQL
   UPDATE emp SET last_name = :last-name
      WHERE CURRENT OF Cur1
END-EXEC
```

This will update `last_name` in the row that was last fetched from the database using cursor `Cur1`.

```
EXEC SQL
   DELETE emp WHERE CURRENT OF Cur1
END-EXEC
```

This example will delete the row that was last fetched from the database using cursor `Cur1`.

OpenESQL With some ODBC drivers, cursors that will be used for positioned updates and deletes must include a FOR UPDATE clause. Note that positioned UPDATE and DELETE are part of the Extended ODBC Syntax and are not supported by all drivers.

COBSQL With COBSQL, cursors that will be used for positioned updates and deletes must include a FOR UPDATE clause.

4.7 Using Cursors

Cursors are very useful for handling large amounts of data but there are a number of issues which you should bear in mind when using cursors, namely: data concurrency, integrity and consistency.

To ensure the integrity of your data, a database server can implement different locking methods. Some types of data access do not acquire any locks, some acquire a shared lock and some an exclusive lock. A shared lock allows other processes to access the data but not update it. An exclusive lock does not allow any other process to access the data.

When using cursors there are three levels of isolation and these control the data that a cursor can read and lock:

- Level Zero

  Level zero can only be used by read-only cursors. At level zero, the cursor will not lock any rows but may be able to read data that has not yet been committed. Reading uncommitted data is dangerous (as a rollback operation will reset the data to its previous state) and is normally called a "dirty read". Not all databases will allow dirty reads.

- Level One

  Level one can be used by read-only cursors or updateable cursors. With level one, shared locks are placed on the data unless the FOR UPDATE clause is used. If the FOR UPDATE clause is used, exclusive locks are placed on the data. When the cursor is closed, the locks are released. A standard cursor, that is a cursor without the FOR UPDATE clause will normally be at isolation level one and use shared locks.

- Level Three

  Level three cursors are used with transactions. Instead of the locks being released when the cursor is closed, the locks are released when the transaction ends. With level three it is usual to place exclusive locks on the data.

It is worth pointing out that there can be problems with "dead-locks" or "deadly embraces" where two processes are competing for the same data. The classic example is where one process locks data A and then requests a lock on data B while a second process locks data B and then requests a lock on data A. Both processes have data that the other process requires. The database server should spot this case and send errors to one, or both, processes.

COBSQL Oracle, Sybase and Informix allallow an application to set the isolation level of the cursor and their documentation discusses the types of locks that are applied and how they work. Their documentation also discusses the physical level that the data is locked at. This can be a single row, a set of rows (that is, the page level), or the whole table. Care should be taken when using cursors that scan multiple tables, or tables that are used by most processes, as this will reduce the accessibility of the locked data.

Note:

COBSQL Oracle, Sybase and Informix enable cursors to be defined with a number of different clauses e.g. FOR READ ONLY, FOR UPDATE etc. These clauses effect the isolation level of the cursor and how it acts when involved in transaction processing. For more information on the effect of these difference clauses, refer to the SQL reference book supplied with your database.

---

5 Data Structures

The Net Express SQL preprocessors use two data structures:

Data Structure	Description	Function
SQLCA	SQL Communications Area	Returns status and error information.
SQLDA	SQL Descriptor Area	Describes the variables used in dynamic SQL statements.

5.1 SQL Communications Area (SQLCA)

After each embedded SQL statement is executed, error and status information is returned in the SQL Communications Area (SQLCA).

Full details of the layout of the SQLCA data structure are given in the online help file. Look under SQLCA in the help file index.

The SQLCA contains two variables (`sqlcode` and `sqlstate`) plus a number of warning flags which are used to indicate whether an error has occurred in the most recently executed SQL statement.

COBSQL For COBSQL, `SQLSTATE` is a separate data item. For the currently supported versions of Oracle and Sybase, the SQLCA should be used in preference to the SQLSTATE variable. The SQLSTATE variable will eventually supersede the SQLCA as the preferred method of passing data between the database and the client application, but this is not yet the case.

5.1.1 The SQLCODE Variable

Testing the value of *sqlcode* is the most common way of determining the success or failure of an embedded SQL statement. The possible values for `sqlcode` are:

Value	Meaning
0	The statement ran without error.
1	The statement ran, but a warning was generated. The values of the `sqlwarn` flags should be checked to determine the type of error.
100	Data matching the query was not found or the end of the results set has been reached. No rows were processed.
< 0 (negative)	The statement did not run due to an application, database, system, or network error.

Full details of SQLCODE values are given in the online help file. Look under "SQLCODE" in the help file index.

COBSQL and DB2 For COBSQL and DB2, it is possible to get other positive values. This means that the SQL statement has executed but produced a warning.

COBSQL

- For details about the range of positive values that SQLCODE can be set to, consult your Oracle, Sybase or Informix Error Messages manual. The above SQLCODES are produced by OpenESQL. The values for SQLCODE and the errors reported by Oracle, Sybase and Informix are all different. Please refer to *Oracle*, *Sybase* or *Informix Error Messages* manuals for more information on the errors returned.

- The value +100 is the ANSI standard for 'data not found'. Oracle can return another value for 'data not found'. To get Oracle to return the value +100 for 'data not found', either of the Oracle precompiler directives MODE=ANSI or END_OF_FETCH=100 must be set. This will affect other aspects of the way the Oracle precompiler handles SQL statements. For more details on the Oracle precompiler MODE or END_OF_FETCH=+100 directives, refer to the ***Programmer's Guide to the ORACLE Precompilers***.

- Even when SQLCODE contains zero, a warning may have been generated. The values of the `sqlwarn` flags should be checked to determine the type of warning. For Oracle, Sybase and Informix, sqlwarn0 will always be set when the database server has sent a warning back to the application.

5.1.2 The SQLSTATE Variable

DB2 DB2 Universal Database returns SQL-92 compliant SQLSTATE values. DB2 Version 2.1 does not.

The sqlstate variable was introduced in the SQL-92 standard and is the recommended mechanism for future applications. It is divided into two components:

- The first two characters are called the class code. Any class code that begins with the letters A through H or the digits 0 through 4 indicates a sqlstate value that is defined by the SQL standard or another standard.
- The last three characters are called the subclass code.

A value of "00000" indicates that the previous embedded SQL statement executed successfully.

For specific details of the values returned in SQLSTATE when using Oracle, Sybase or Informix, refer to the relevant *Database Error Messages manual*. Full details of SQLSTATE values are also given in the online file. Look under **SQLSTATE** in the help file index.

5.1.3 The Warning Flags

Some statements may cause warnings to be generated. To determine the type of warning, your application should examine the contents of the sqlwarn flags. The value of the flag will be set to "W" if that particular warning occured, otherwise the value will be a blank (space).

Each sqlwarn flag has a specific meaning. For more information on the meaning of the sqlwarn flags, refer to the online help - look under "SQLCA" in the help file index.

5.1.4 The WHENEVER Statement

Explicitly checking the value of `sqlcode` or `sqlstate` after each embedded SQL statement can involve writing a lot of code; an alternative is to check the status of the SQL statement by using a WHENEVER statement in your application.

The WHENEVER statement is not an executable statement; it is a directive to the Compiler to generate automatically code that handles errors after each executable embedded SQL statement.

The WHENEVER statement allows one of three default actions (CONTINUE, GOTO or PERFORM) to be registered for each of the following conditions:

Condition	**Value of sqlcode**
NOT FOUND	100
SQLWARNING	+1
SQLERROR	< 0 (negative)

COBSQL For Oracle, Sybase and Informix, the 'SQLWARNING' clause will be triggered when `sqlwarn0`is set to 'W'.

Oracle When no data is returned from a SELECT or FETCH statement, the condition NOT FOUND is triggered, regardless of the setting of the Oracle precompiler directive MODE.

Informix Informix allows you to perform a STOP or a CALL from within a WHENEVER statement. These are additions to the ANSI standard and are documented in the **Informix ESQL/COBOL programmers manual**.

A WHENEVER statement for a particular condition replaces all previous WHENEVER statements for that condition.

The scope of a WHENEVER statement is related to its physical position in the source program, not its logical position in the run sequence. For example, in the following code if the first SELECT statement does not return anything, paragraph A is performed, not paragraph C:

```
EXEC SQL
    WHENEVER NOT FOUND PERFORM A
END-EXEC.
perform B.
EXEC SQL
```

```
           SELECT col1 into :host-var1 FROM table1
            WHERE col2 = :host-var2
        END-EXEC.
    A.
        display "First item not found".
    B.
        EXEC SQL
            WHENEVER NOT FOUND PERFORM C.
        END-EXEC.
    C.
        display "Second item not found".
```

5.1.5 SQLERRM

The SQLERRM data area is used to pass error messages to the application from the database server. The SQLERRM data area is split into two parts: SQLERRML and SQLERRMC. SQLERRML holds the length of the error message and SQLERRMC holds the error text. Within an error routine, the following code can be used to display the SQL error message:

```
if (SQLERRML > ZERO) and (SQLERRML < 80)
    display ’Error Message: ’, SQLERRMC(1:SQLERRML)
else
    display ’Error Message: ’, SQLERRMC
end-if.
```

5.1.6 SQLERRD

The SQLERRD data area is an array of six integer status values.

COBSQL Oracle, Sybase and Informix may set one (or more) of the six values within the SQLERRD array. These indicate how may rows were effected by the SQL statement just executed. For example, SQLERRD(3) holds the total number of rows returned by a SELECT or a series of FETCH statements.

5.2 The SQL Descriptor Area (SQLDA)

Use an SQL Descriptor Area (SQLDA) instead of host variables in the following circumstances:

- Dynamic SQL statements

  When the number of parameters to be passed or their data types are unknown at compilation time

- Static SQL statements

  With a cursor FETCH statement.

An SQLDA contains descriptive information about each input parameter or output column:

- Column name
- Data type
- Length
- A pointer to the data buffer for each input or output parameter

The SQLDA is unique to each precompiler and ensures that data is converted in the correct format.

Typically, an SQLDA is used with parameter markers to specify input values for prepared SQL statements. However, to receive data from a prepared SELECT statement, you can also use an SQLDA with either the DESCRIBE statement or the INTO option of a PREPARE statement.

The Oracle SQLDA is not compatible with that used by Sybase, OpenESQL or DB2. Similarly, the Sybase, OpenESQL or DB2 SQLDAs are not compatible with the Oracle SQLDA.

COBSQL For both Oracle and Sybase, the SQLDA is only required if your program uses dynamic SQL.

COBSQL Oracle, Sybase and Informix do not allow the SQLDA to be included in your program using the following syntax statement:

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

For Oracle, Sybase and Informix, the SQLDA must be defined as a standard COBOL copyfile.

COBSQL Oracle provides an extra copyfile, ORACA, for use with dynamic SQL. This can be included in your program using the following syntax:

```
EXEC SQL
   INCLUDE ORACA
END-EXEC
```

You must set the Oracle precompiler option, ORACA=YES before you can use the ORACA. For more information on setting Oracle precompiler options, refer to the ***Programmer's Guide to the Oracle Precompilers***.

COBSQL Oracle does not supply an SQLDA but the ***Programmer's Guide to the Oracle Precompilers*** contains a definition of the layout.

COBSQL Sybase does not supply an SQLDA copyfile. The Sybase precompiler documentation describes the layout of the SQLDA and how to assign values to the various items within it. The documentaion also describes how to get Sybase to convert between COBOL and Sybase data types.

OpenESQL The SQLDA structure is supplied in the file **sqlda.cpy** in the **source** directory under your Net Express base installation directory. You can include it in your COBOL program by adding the following statement to your data division:

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

Full details of the OpenESQL SQLDA are given in the online help file. Look under "SQLDA" in the help file index.

5.2.1 Using the SQLDA

Before an SQLDA structure is used, your application must initialise the following fields:

SQLN This must be set to the maximum number of SQLVAR entries that the structure can hold.

SQLDABC The maximum size of the SQLDA. This is calculated as SQLN * 44 + 16

5.2.1.1 The PREPARE and DESCRIBE Statements

You can use the DESCRIBE statement (or the PREPARE statement with the INTO option) to enter the column name, data type, and other data into the appropriate fields of the SQLDA structure.

Before the statement is executed, the SQLN and SQLDABC fields should be initialised as described above.

After the statement has been executed, the SQLD field will contain the number of parameters in the prepared statement. A SQLVAR record is set up for each of the parameters with the SQLTYPE and SQLLEN fields completed.

If you do not know how big the value of SQLN should be, you can issue a DESCRIBE statement with SQLN set to 1 and SQLD set to 0. No column detail information is moved into the SQLDA structure, but the number of columns in the results set is inserted into SQLD.

5.2.1.2 The FETCH Statement

Before performing a FETCH statement using an SQLDA structure, the application must initialize SQLN and SQLDABC as described above. It must then insert into the SQLDATA field the address of each program variable that will receive the data from the corresponding column. (The SQLDATA field is part of SQLVAR). If indicator variables are used, SQLIND must also be set to the corresponding address of the indicator variable.

The data type field (SQLTYPE) and length (SQLLEN) are filled with information from a PREPARE INTO or a DESCRIBE statement. These values can be overwritten by the application prior to a FETCH statement.

5.2.1.3 The OPEN or EXECUTE Statements

To use an SQLDA structure to specify input data to an OPEN or EXECUTE statement, your application must supply the data for the fields of the entire SQLDA structure, including the SQLN, SQLD, SQLDABC, and SQLTYPE, SQLLEN, and SQLDATA fields for each variable. If the value of the SQLTYPE field is an odd number, the address of the indicator variable must also be supplied in SQLIND.

5.2.2 The DESCRIBE Statement

After a PREPARE statement, you can execute a DESCRIBE statement to retrieve information about the data type, length and column name of each column returned by the specified prepared statement. This information is returned in the SQL Descriptor Area (SQLDA):

```
EXEC SQL
   DESCRIBE stmt1 INTO :sqlda
END-EXEC
```

If you want to execute a DESCRIBE statement immediately after a PREPARE statement, you can use the INTO option on the PREPARE statement to perform both steps at once:

```
EXEC SQL
   PREPARE stmt1 INTO :sqlda FROM :stmtbuf
END-EXEC
```

Database Access

6 Dynamic SQL

If everything is known about a SQL statement when the application is compiled, it is known as a static SQL statement.

In some cases, however, the full text of a SQL statement may not be known when an application is written. For example, you may need to allow the end-user of the application to enter a SQL statement. In this case, the statement needs to be constructed at run-time. This is called a dynamic SQL statement.

6.1 Dynamic SQL Statement Types

There are four types of dynamic SQL statement:

Dynamic SQL Statement Type	**Perform Queries?**	**Return Data?**
Execute a statement once	No	No; can only return success or failure
Execute the same statement more than once	No	No; can only return success or failure
Select a given list of data with a given set of selection criteria	Yes	Yes
Select any amount of data with any selection criteria	Yes	Yes

These types of dynamic SQL statement are described more fully below.

6.1.1 Execute a Statement Once

With this type of dynamic SQL statement, the SQL statement is executed immediately. Each time the statement is executed, it is re-parsed.

6.1.2 Execute the Same Statement More than Once

This type of dynamic SQL statement is either a statement that can be executed more than once or a statement that requires host variables. For the second type, the statement has to be prepared before it can be executed.

6.1.3 Select a Given List of Data

This type of dynamic SQL statement is a SELECT statement where the number of, and type of host variables is known. The normal sequence of SQL statements is:

1. Prepare the statement
2. Declare a cursor to hold the results
3. Open the cursor
4. Fetch the variables
5. Close the cursor.

6.1.4 Select any Amount of Data

This type of dynamic SQL statement, and the hardest to code, is where the type of the variables is only resolved at run-time, the number of variables is only resolved at run-time, or a mixture of both. The normal sequence of SQL statements is:

1. Prepare the statement
2. Declare a cursor for the statement

3. Describe the variables to be used
4. Open the cursor using the variables just described
5. Describe the variables to be fetched
6. Fetch the variables using their descriptions
7. Close the cursor.

If either the input host variables, or the output host variables are known (at compile time), then the OPEN or FETCH can name the host variables and they do not need to be described.

6.2 Preparing Dynamic SQL Statements

The PREPARE statement takes a character string containing a dynamic SQL statement and associates a name with the statement, for example:

```
move "INSERT INTO publishers " &
          "VALUES (?,?,?,?)" to stmtbuf
EXEC SQL
    PREPARE stmt1 FROM :stmtbuf
END-EXEC
```

Dynamic SQL statements can contain parameter markers - question marks (?) that act as a place holder for a value. In the example above, the values to be substituted for the question marks must be supplied when the statement is executed.

Once you have prepared a statement, you can use it in one of two ways:

- You can execute a prepared statement.
- You can open a cursor that references a prepared statement.

COBSQL Oracle does not use question marks as place holders. It uses the host variable notation. By convention the place holders are named V*n*, where *n* is a number to make the place holder unique within a statement. For readability the same place holder can be used more than once, but when the statement is executed (or opened if you are using a cursor), there must still be one host variable for each place holder, for example:

```
string "update ordtab " delimited by size
       "set order_no = :v1, "
       "line_no = :v2, "
       "cust_code = :v3, "
       "part_no = :v4, "
       "part_name = :v5, "
       "order_val = :v6, "
       "pay_value = :v7 "
       "where order_no = :v1 and "
       "line_no = :v2 and "
       "cust_code = :v3 " delimited by size
 into Updt-Ord-Stmt-Arr
end-string
move 190 to Updt-Ord-Stmt-Len

EXEC SQL PREPARE updt_ord FROM :Updt-Ord-Stmt END-EXEC

EXEC SQL EXECUTE updt_ord USING
    :dcl-order-no, :dcl-line-no, :dcl-cust-code,
    :dcl-part-no,  :dcl-part-name:ind-part-name,
    :dcl-order-val,:dcl-pay-value,
    :dcl-order-no, :dcl-line-no, :dcl-cust-code
END-EXEC
```

where `Updt-Ord-Stmt` has been defined as a host variable type of VARYING.

COBSQL When using the Oracle precompiler, the physical location of a PREPARE statement is important. A PREPARE statement must appear before an EXECUTE or a DECLARE statement.

6.3 Executing Dynamic SQL Statements

The EXECUTE statement runs a specified prepared SQL statement.

Note: Only statements that do not return results can be executed in this way.

If the prepared statement contains parameter markers, the EXECUTE statement must include either the `using :hvar` option to supply parameter values using host variables or the `using descriptor :sqlda_struct` option identifying an SQLDA data structure already populated by the application. The number of parameter markers in the prepared statement must match the number of SQLDATA entries (`using descriptor :sqlda`) or host variables (`using :hvar`).

```
move "INSERT INTO publishers " &
          "VALUES (?,?,?,?)" to stmtbuf
EXEC SQL
    PREPARE stmt1 FROM :stmtbuf
END-EXEC
 ...
EXEC SQL
    EXECUTE stmt1 USING :pubid,:pubname,:city,:state
END-EXEC.
```

In this example, the four parameter markers are replaced by the contents of the host variables supplied via the USING clause in the EXECUTE statement.

6.3.1 Execute Immediate

If the dynamic SQL statement does not contain any parameter markers, you can use EXECUTE IMMEDIATE instead of PREPARE followed by EXECUTE, for example:

```
move "DELETE FROM emp " &
          "WHERE last_name = 'Smith'" to stmtbuf
EXEC SQL
    EXECUTE IMMEDIATE :stmtbuf
END-EXEC
```

COBSQL When using EXECUTE IMMEDIATE, the statement is re-parsed each time it is executed. If a statement is likely to be used many times it is better to PREPARE the statement and then EXECUTE it when required.

6.3.2 FREE Statement (COBSQL Informix)

The Informix precompiler provides a FREE statement that will release resources that are allocated to a prepared statement or to a cursor.

Once you have finished with a prepared statement, you would then use the FREE statement, for example:

```
move "INSERT INTO publishers " " &
          "VALUES (?,?,?,?)" to stmtbuf
EXEC SQL
    PREPARE stmt1 FROM :stmtbuf
END-EXEC
 ...
EXEC SQL
    EXECUTE stmt1 USING :pubid,:pubname,:city,:state
END-EXEC.
 ...
EXEC SQL
    FREE stmt1
END-EXEC
```

6.4 Dynamic SQL Statements and Cursors

If a dynamic SQL statement returns a result, you cannot use the EXECUTE statement. Instead, you must declare and use a cursor.

First, declare the cursor using the DECLARE CURSOR statement:

```
EXEC SQL
   DECLARE C1 CURSOR FOR dynamic_sql
END-EXEC
```

In the example above, `dynamic_sql` is the name of a dynamic SQL statement. You must use the PREPARE statement to prepare the dynamic SQL statement before the cursor can be opened, for example:

```
move "SELECT char_col FROM mfesqltest " &
     "WHERE int_col = ?" to sql-text
EXEC SQL
   PREPARE dynamic_sql FROM :sql-text
END-EXEC
```

Now, when the OPEN statement is used to open the cursor, the prepared statement is executed:

```
EXEC SQL
   OPEN C1 USING :int-col
END-EXEC
```

If the prepared statement uses parameter markers, then the OPEN statement must supply values for those parameters by specifying either host variables or an SQLDA structure.

Once the cursor has been opened, the FETCH statement can be used to retrieve data, for example:

```
EXEC SQL
   FETCH C1 INTO :char-col
END-EXEC
```

See the chapter *Cursors* for a full discussion of the FETCH statement.

Finally, the cursor is closed using the CLOSE statement:

```
EXEC SQL
   CLOSE C1
END-EXEC
```

See the chapter *Cursors* for a full discussion of the CLOSE statement.

6.5 CALL Statements

A CALL statement can be prepared and executed as dynamic SQL.

- You can use parameter markers (?) in dynamic SQL wherever you use host variables in static SQL

- Use of the IN, INPUT, OUT, OUTPUT, INOUT and CURSOR keyword following parameter markers is the same as their use after host variable parameters in static SQL.
- The whole call statement must be enclosed in braces to conform to ODBC cannonical stored procedure syntax (the Open ESQL precompiler does this for you in static SQL). For example:

```
move ’{call myproc(?, ? out)}’ to sql-text
exec sql
    prepare mycall from :sql-text
end-exec
exec sql
    execute mycall using :parm1, :param2
end-exec
```

- If you use parameter arrays, you can limit the number of elements used with a FOR clause on the EXECUTE, for example:

```
move 5 to param-count
exec sql
    for :param-count
     execute mycall using :parm1, :param2
end-exec
```

Part 2: OpenESQL

This part contains the following chapters:

- Chapter 7, “OpenESQL”
- Chapter 8, “OpenESQL Assistant”

7 OpenESQL

The OpenESQL preprocessor enables you to access a relational database via an ODBC driver by embedding SQL statements within your COBOL program.

Unlike separate preprocessors, OpenESQL is controlled by specifying the SQL directive when you compile your application.

7.1 ODBC Drivers and Data Source Names

To obtain ODBC support, you must:

1. Install ODBC drivers.
2. Set up an ODBC Data Source Name (DSN).

7.1.1 Installing ODBC Drivers

At the beginning of the Net Express installation, you should indicate that you wish to install a number of ODBC drivers by selecting **ODBC drivers** from the given list.

Note: Most of these drivers will only work if the Client Software for the specific database is present.

7.1.2 Setting up a Data Source Name

The Net Express ODBC support cannot work until you have set up a data source name (DSN) in the ODBC Manager. You can access the ODBC Manager via the **Control Panel** on Windows 95, Windows 98, or

Windows NT desktop. Click on the appropriate **16bit ODBC** or **32bit ODBC** icon. Click on the **System DSN** tab.

The ODBC Data Source Administrator dialog box opens. All those drivers that are installed as a part of Net Express appear with names that begin **Net Express ....** In addition, Net Express installs the Microsoft Access driver. This driver appears as **Net Express Microsoft Access**.

For details of how to assign DSNs, click on the **Help** button in the ODBC Data Source Administrator dialog box.

Notes:

- The ODBC drivers provided with Server Express are licensed for development use only. You may use these drivers to develop Net Express applications but you may not distribute the drivers to your users. To redistribute ODBC drivers with your application, you need run-time licenses for the drivers. Contact your Micro Focus sales representative for information on obtaining licenses for run-time distribution of the ODBC drivers. See ***List of Key Features*** in your ***Getting Started*** for more detailed licensing information.
- An Informix 7 driver is not available for AIX. Therefore, Informix 7 clients are not supported on AIX.

7.2 ORACLE OCI Support

OpenESQL provides an alternative for developers using ORACLE data sources, in the form of the ORACLE OCI interface. ORACLE OCI provides faster processing of SQL statements than the more generic ODBC interface that OpenESQL normally uses. To use this interface, you must compile your applications with the following directive:

```
sql(targetdb=ORACLEOCI)
```

When connecting to the Oracle server, use an Oracle Net8 service name in place of an ODBC data source name in the CONNECT statement. See your Oracle documentation on how to set up ORACLE Net8 services.

When you use Oracle OCI, be aware that the following OpenESQL functions are not supported:

- BEGIN TRANSACTION statement
- CALL statement
- EXECSP statement
- QUERY ODBC statement
- SET AUTOCOMMIT statement
- SET TRANSACTION ISOLATION statement
- Scrolling cursorsd (including SET SCROLLOPTION and SET CONCURRENCY statements)
- WITH PROMPT parameter on CONNECT statement
- SQL (CONNECTIONPOOL) directive

7.3 SQL Compiler Directive

When you compile your program, you must specify the SQL Compiler directive and its appropriate options such that the preprocessor converts the embedded SQL statements into function calls to the data source. The ODBC driver that your program calls depends on the particular data source that you are accessing.

There are two ways of specifying options for the SQL Compiler directive:

- You can use the $SET statement in your program. For example:

  ```
  $set sql(dbman=odbc, datecheck, autocommit)
  ```

- You can use the Net Express **Advanced Directives** screen. Select **Build Settings** from the **Project** menu, select your program, click on the **Compile** tab and then press the **Advanced** button.

Note: You cannot use a mixture of these methods - you must use one or the other.

The table below lists the SQL Compiler directive options.

Option	**Description**
DBMAN=*preprocessor*	Specifies the preprocessor to use. This should always be set to **odbc**, that is **dbman=odbc**. This directive is not required when compiling programs with OpenESQL.
[NO]ANSI92ENTRY	If this is set, OpenESQL conforms to the SQL ANSI 92 entry level standard.
[NO]AUTOCOMMIT	If this is set, each SQL statement is treated as a separate transaction and is committed immediately upon execution. If this is not set, and the ODBC driver you are using supports transactions, statements must be explicitly committed (or rolled back) as part of a transaction.
[NO]CHECK	If this is set, each SQL statement is sent to the database at compilation time. If you specify statement checking at compilation time, you must also set **DB** and **PASS**.
CHECKSINGLETON	Causes OpenESQL to check if singleton SELECTs return more than one row when executed. If this occurs, OpenESQL sets SQLCODE to a -811.
CONCAT=ascii character code	Specifies the ASCII character code to use for the CONCAT symbol (|). Use this directive only if you need to change the default, which is 33.
CONNECTIONPOOL=[DRIVER | ENVIRONMENT | NONE]	Default is NONE. Enables use of ODBC 3.0 connection pooling. When a connection is closed, the Driver Manager actually keeps it alive for a timeout period, and saves the overhead of re-establishing a connection from scratch if the application re-opens an identical connection. ODBC allows you to choose between having a pooling for an ODBC environment or for each driver. See your ODBC documentation for details. This option is only useful for applications that frequently open and close connections. Note that some environments, such as Microsoft Transaction Server (MTS) control connection pooling themselves. This option will probably improve the performance of ISAPI applications that are not running under MTS.
[NO]CTRACE	Causes debug information to be put into a **sqltrace.txt** file. Default is NOCTRACE.

Option	Description
[NO]CURSORCASE	If ESQLVERSION is 2.0, CURSORCASE is implied. Default is NOCURSORCASE which means that cursor names are case insensitive. CURSORCASE means that they are case sensitive. Note that in previous versions of OpenESQL, cursor names have been case sensitive.
[NO]DB	The name of the data source to connect to. This option works in conjunction with the **INIT** and/or **CHECK** options.
[NO]DETECTDATE	Default is NODETECTDATE. If DETECTDATE is set, OpenESQL inspects character host variables for ODBC escape sequences: {d<data>} - date {t<data>} - time {ts<data>} - timestamp and binds the parameter appropriately, rather than as a character column. This is necessary if your server does not have a suitable native character string date representation (for example, Microsoft Access). It is also useful for generic applications. It can, however, cause problems if you have other character columns that can legitimately contain data that starts with "{d", "{t" or "{ts".
[NO]ESQLVERSION	Set OpenESQL syntax level.
[NO]FIPSFLAG	Enables FIPS flagging (NIST certification requirement only). Default is NOFIPSFLAG.
GEN-CC2	Generates OpenESQL database interface calls, using call convention 2 rather than call convention 74. Use this directive to generate applications in the same way that Net Express 3.0 did. If creating an **.exe**, you may need to add **odbcrw32.lib** to your link statement.
[NO]IGNORE-NESTED=program-id	In nested programs, this is the program-id at which to start generating the database interface code. If the program file name matches the program-id, you can just specify IGNORE-NESTED. Default is NOIGNORE-NESTED.
[NO]INIT	If this is set, the preprocessor automatically generates code to make the connection to the database. If you specify INIT, you must also specify **DB** and **PASS**. It is highly recommend that you use EXEC SQL CONNECT statements in your application instead.

Option	Description
[NO]NIST	If this is set, OpenESQL will conform to the NIST interpretation of the SQL ANSI 92 entry level standard.
NOT=ascii character code	Specifies the ASCII character code to use for NOT symbol (¬). Use this only if you need to change the default setting, which is: 170.
ODBCTRACE= [ALWAYS | NEVER | USER]	Default is USER. ODBCTRACE=USER enables you to control ODBC tracing via the OBDC control panel from which you can specify the file that the trace goes into. ALWAYS lets you control OBDC tracing via a directive, which is more convenient from within the IDE. ALWAYS generates the trace into MFSQLTRACE.LOG in the current directory, regardless of the settings on the ODBC Control Panel. Under normal development conditions, and depending on the project's build setting, this is the Debug or Release directory of the current project. NEVER means that the application will never be traced and overrides the control panel. As ODBC trace files can contain sensitive information, use NEVER in production applications in secure environments.
[NO]PARAMARRAY	Default is PARAMARRAY. If PARAMARRAY is set, ODBC array binding is used, if it is supported by the ODBC driver, for all input parameters.
[NO]PASS	The login to use to connect to the data source. This option works in conjunction with the **INIT** and/or **CHECK** options.
[NO]PRE	Default is PRE, which causes the preprocessor to generate code to load the OpenESQL runtime module (**odbcrw32.dll**) dynamically at runtime. This conflicts with the LITLINK compiler directive. So if you use LITLINK, specify NOPRE to stop the dynamic loading code being generated. In this case, you must add **odbcrw32.lib** to the list of LIBs to be linked in your build settings. Then the linker generates code into the executable which causes the operating system to load **odbcrw32.dll** implicitly when the executable is loaded.
[NO]QUALFIX	Causes the preprocessor to append three characters to the name of the host variables when declaring them to ODBC. Default is QUALFIX.

Option	Description
[NO]RESULTARRAY	Default is RESULTARRAY. If RESULTARRAY is set, ODBC array binding is used, if it is supported by the ODBC driver, for all output parameters.
STMTCACHE	Sets the cache size for prepared statements used by OpenESQL. The default is 20. Depending upon your application and data source, performance improvements or data errors can result if this value is set higher than that.
[NO]TARGETDB=[MSQLSERVER | ORACLEOCI | ORACLE | INFORMIX | SYBASE | DB2 | ORACLE7]	Set this directive if you want to optimize performance for a specific data source or have the application generate database calls using ORACLE OCI rather than ODBC calls.
THREAD=[SHARE | ISOLATE]	Default is SHARE. If THREAD is set to ISOLATE, all connections, cursors and so on are local to the thread that creates them. This is required for multi-threaded application server environments such as IIS/ISAPI. With THREAD=SHARE, if you have a hard-coded CONNECT statement and thread 1 executes it and then thread 2 executes it, thread 2 gets an error because the connection is already open. With THREAD=ISOLATE, each thread gets its own connection.
[NO]USECURLIB[NO | YES | IFNEEDED]	Controls use of the ODBC’s Cursor Library. The Cursor Library can provide support for scrolling cursors when the underlying driver doesn’t, and can also allow "simulated" positioned updates. With USECURLIB=YES, the Cursor Library will always be used. With USERCURLIB=NO, it will never be used. With the default USERCURLIB=IFNEEDED, it will be used if the application tries to do something the driver manager thinks the driver doesn’t support. To use a scrolling cursor with the Cursor Library, you must use a STATIC cursor. To do positioned updates using the Cursor Library, you must use OPTCCVAL concurrency. Please beware, a "simulated" positioned update might hit more than one row. We recommend including the primary key in the select for this reason.

7.4 Data Sources

When you install Net Express, two Data Source Names (DSNs) are created automatically. These are **NetExpress Sample1** and **NetExpress Sample2** and they point to the sample Access databases (**demo.mdb** and **sample.mdb**) that are installed as part of Net Express in the **demo\smpldata** directory. If you want to look at an XDB database, the DSN is **NetExpress XDB Sample1**.

7.5 Database Connections

Before your program can access any data in a database, it must make a connection to the database.

There are two methods your program can use to connect to a database.

- Explicit connection (recommended method)

  The CONNECT statement is typically used if the program is designed to access different data sources whose names are not known at compilation time or if the program is going to access multiple databases.

- Implicit connection

  This is generally used if your program is only going to connect to one database which is known at compilation time. If you specify the INIT option of the SQL Compiler directive, the compiler inserts a call at the start of the program to automatically connect the program to the data source specified in the DB option of the SQL Compiler directive using the login information specified in the PASS option.

When your application has finished working with a database, it should disconnect from the database. This is done using the DISCONNECT statement.

If implicit connection is being used, OpenESQL automatically disconnects from the data source when the program terminates.

If you want OpenESQL to perform an implicit disconnect and rollback in the event of abnormal program termination this can be achieved by specifying the INIT=PROT option of the SQL Compiler directive.

7.6 Keywords

A number of keywords are recognized by OpenESQL and should not therefore be used within your program for other purposes. A full list of reserved keywords is given in the online help file. Look under "OpenESQL" in the help file index.

7.7 Building an Application

To build an OpenESQL application, you need to:

1. Write your application, surrounding your SQL statements with the keywords EXEC SQL and END-EXEC.
2. Compile your application using the SQL Compiler directive.
3. Configure a data source via ODBC Data Sources on the Control Panel (see the section *Setting up a Data Source Name*).

The copyfiles **sqlca.cpy** and **sqlda.cpy** are located in the **source** directory under your Net Express base installation directory and can be included in your program in the normal way.

When you build your **.exe**, using Net Express, all the necessary object files are linked in for you automatically unless you compile your program with directive SQL(GEN-CC2).

Note: If you move your application to another system, you should ensure that the file **odbcrw32.dll** is available on that system.

7.8 Demonstration Applications

A number of demonstration applications are supplied in the **odbcesql** directory which is located in the **demo** directory under your Net Express base installation directory.

Before you can use any of the demonstration applications, you need to have installed at least one ODBC driver. A number of ODBC drivers are installed automatically with Net Express including a Microsoft Access driver. In addition, two sample Access databases are supplied in the **demo\smpldata** directory and two data source names which point to them are created automatically when you install Net Express. You can run the demonstration applications against the sample database pointed to by the data source name **NetExpress Sample2** (or **NetExpress XDB Sample1** for XDB).

The OpenESQL demonstration applications all produce a console log displaying their progress and, possibly, query results. They all terminate on receipt of an error, after displaying an error message.

- **connect.app**

  Prompts for a data source, user name and password. Enter a data source name of "NetExpress Sample2" (or "NetExpress XDB Sample1" for XDB), a user name of "admin" and leave the password blank (just press **return**). Four tests which perform connects and disconnects using a variety of syntax options are run. The fifth test displays an SQL Data Sources dialog. Select "NetExpress Sample2" (or "NetExpress XDB Sample1" for XDB) from the **Machine Data Source** list and click on **OK**. A Login dialog is displayed. Enter a login name of "admin", leave the password blank and click on **OK**. The fifth test is run and the program terminates.

- **select.app**

  Connects to the sample database and prompts for a customer code. Enter BLUEL (as the prompt suggests). Two fields from the customer record are displayed and the program prompts for another customer code. Just press the **return** key this time. The program prompts for a region. Enter CA (as the prompt suggests). The program displays a list of customers in that region and prompts for another region. This time, press the **return** key to terminate the program.

- **static.app** and **dynamic.app**

  Both applications run the same sequence of tests, but using different SQL syntax options. They both start by prompting for a data source and user name. Enter "NetExpress Sample2" (or "NetExpress XDB Sample1" for XDB) and "admin" respectively. The test sequence is:

  connect
  drop test table
  create test table
  insert a row
  commit
  update the row
  read and verify
  rollback
  read and verify
  drop test table
  disconnect
  create test table

  Step two may output an error message - this is expected and the programs will continue. The final stage should produce an error message, and again this is not treated as a genuine error (though the absence of an ODBC error is treated as a test failure).

- **whenever.app**

  Attempts to connect and displays an error message. Displays an SQL Data Sources dialog. Select "NetExpress Sample2" (or "NetExpress XDB Sample1" for XDB) and click on **OK**. A Login dialog is displayed. Enter a login name of "admin", leave the password blank and click on **OK**. Tests error handling and outputs two error messages.

- **catalog.app**

  Displays an SQL Data Sources dialog. Select "NetExpress Sample2" (or "NetExpress XDB Sample1" for XDB) and click on **OK**. A Login dialog is displayed. Enter a login name of "admin", leave the password blank and click on **OK**. Performs three data dictionary queries and outputs the results.

7.9 Managing Transactions

With OpenESQL, you can use the COMMIT and ROLLBACK statements to exploit the transaction control facilities of ODBC. Although ODBC specifies transaction AUTOCOMMIT after each statement as the default mode of operation, OpenESQL turns this off for greater compatibility with other SQL systems. If you require this functionality, specify the AUTOCOMMIT option of the SQL Compiler directive.

Note: Not all ODBC drivers implement transaction processing and those that do not may make updates to the database permanent immediately.

7.10 Data Types

The online help includes a table which shows the mappings used by OpenESQL when converting between SQL and COBOL data types. Look up *Reference*, *Database Access*, *OpenESQL*, *Data Types* in the online help contents.

The format of an ODBC date is *yyyy-mm-dd*, and an ODBC time is *hh:mm:ss*. These may not correspond to the native date/time formats for the data source in use. For input character host variables, native data source date/time formats can be used. For most data sources, we recommend a picture clause of PIC X(26), for example:

```
01  mydate      PIC x(26).
...
    EXEC SQL
        INSERT INTO TABLE1 VALUES (1,’1997-01-24 12:24’)
    END-EXEC
...
    EXEC SQL
        SELECT DT INTO :mydate FROM TABLE1 WHERE X = 1
    END-EXEC
    display mydate
```

Alternatively, you can use ODBC escape sequences. ODBC defines escape sequences for date, time and timestamp literals. These escape sequences are recognized by ODBC drivers which replace them with data source specific syntax.

The escape sequences for date, time and timestamp literals take the form:

`{d 'yyyy-mm-dd'}` - for date.
`{t 'hh:mm:ss'}` - for time.
`{ts yyyy-mm-dd hh:mm:ss[.f...]` - for timestamp.

The example program below shows date, time and timestamp escape sequences being used:

```
 working-storage section.
 EXEC SQL INCLUDE SQLCA END-EXEC

 01  date-field1      pic x(26).
 01  date-field2      pic x(26).
 01  date-field3      pic x(26).

 procedure division.
* Connect to the data source.  This is one of the Sample
* datasources supplied with NetExpress
     EXEC SQL
        CONNECT TO 'NetExpress Sample1' USER 'admin'
     END-EXEC
* If the Table is there drop it.
     EXEC SQL
        DROP TABLE DT
     END-EXEC

* Create a table with columns for DATE, TIME, and DATE/TIME
* NOTE:  Access uses DATETIME column for all three.
*        Some databases will have dedicated column types.
* If you are creating DATE/TIME columns on another data
* source, refer to your database documentation to see how to
* define the columns.

     EXEC SQL
        CREATE TABLE DT ( id  INT,
                         myDate DATE NULL,
                         myTime TIME NULL,
                         myTimestamp TIMESTAMP NULL)
     END-EXEC

* INSERT into the table using the ODBC Escape sequences

     EXEC SQL
```

```
            INSERT into DT values (1 ,
                {d ’1961-10-08’},  *> Set just the date part
                {t ’12:21:54’  },  *> Set just the time part
                {ts ’1966-01-24 08:21:56’ } *> Set both parts
                                  )
         END-EXEC

    * Retrieve the values we just inserted

         EXEC SQL
            SELECT myDate
                  ,myTime
                  ,myTimestamp
             INTO  :date-field1
                  ,:date-field2
                  ,:date-field3
             FROM DT
                 where id = 1
         END-EXEC

    * Display the results.

         display ’where the date part has been set :’
                 date-field1
         display ’where the time part has been set :’
                 date-field2
         display ’NOTE, most data sources will set a default ’
                 ’for the date part ’
         display ’where both parts has been set :’
                 date-field3

    * Remove the table.

         EXEC SQL
            DROP TABLE DT
         END-EXEC

    * Disconnect from the data source

         EXEC SQL
            DISCONNECT CURRENT
         END-EXEC

         stop run.
```

In the above example, you can use host variables defined with SQL TYPEs for date/time variables. Define the following host variables as:

```
01  my-id           pic s9(08) COMP-5.
01  my-date         sql type is date.
01  my-time         sql type is time.
01  my-timestamp    sql type is timestamp.
```

and replace the INSERT statement with the following code:

```
*> INSERT into the table using SQL TYPE HOST VARS
    MOVE 1                          TO  MY-ID
    MOVE "1961-10-08"               TO  MY-DATE
    MOVE "12:21:54"                 TO  MY-TIME
    MOVE "1966-01-24 08:21:56"      TO  MY-TIMESTAMP

     EXEC SQL
        INSERT into DT value (
          :MY-ID
         ,:MY-DATE
         ,:MY-TIME
         ,:MY-TIMESTAMP  )
     END-EXEC
```

7.11 Using the SQLCA

The SQLCA data structure is included in the file **sqlca.cpy** in the **source** directory under your Net Express base installation directory. To include it in your program, use the following statement in the data division:

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

If you do not include this statement, the COBOL Compiler automatically allocates an area, but it is not addressable from within your program. However, if you declare either of the data items SQLCODE or SQLSTATE separately, the COBOL Compiler generates code to copy the corresponding fields in the SQLCA to the user-defined fields after each EXEC SQL statement.

If you declare the data item MFSQLMESSAGETEXT, it is updated with a description of the exception condition whenever SQLCODE is non-zero. MFSQLMESSAGETEXT must be declared as a character data item, PIC X(*n*), where *n* can be any legal value. This is particularly useful as ODBC error messages often exceed the 70-byte SQLCA message field.

Note: You do not need to declare SQLCA, SQLCODE, SQLSTATE or MFSQLMESSAGETEXT as host variables.

7.12 Dynamic SQL

The demonstration application, **dynamic.app** is located in the **odbcesql** directory which is in the **demo** directory under your base Net Express installation directory. Open this project and select **Step** from the **Animate** menu (you may need to rebuild the project first) in order to step through the application for a demonstration of how to use dynamic SQL in your COBOL programs.

7.13 Positioned Update

ODBC supports positioned update, which updates the row most recently fetched by using a cursor. However, not all drivers provide support for positioned update.

With some ODBC drivers, the select statement used by the cursor must contain a FOR UPDATE clause to enable positioned update. Most data sources require specific combinations of SCROLLOPTION and CONCURRENCY to be specified either by SET statements or in the DECLARE CURSOR statement. If this fails to work, the ODBC Cursor Library provides a restricted implementation of positioned update which can be enabled by compiling with the directive SQL(USECURLIB=YES) and using SCROLLOPTION STATIC and CONCURRENCY OPTCCVAL (or OPTIMISTIC). To avoid multiple rows being updated when using the ODBC Cursor Library, the cursor query should include the primary key column(s) for the table to be updated.

7.13...1 Example

```
EXEC SQL CONNECT TO 'srv1' USER 'sa' END-EXEC

EXEC SQL DECLARE C1 CURSOR FOR
   SELECT last_name, first_name
   FROM staff
   FOR UPDATE
END-EXEC

EXEC SQL
   OPEN C1
END-EXEC

PERFORM UNTIL SQLCODE NOT = ZERO

   EXEC SQL
       FETCH C1 INTO :fname,:lname
   END-EXEC

   IF SQLCODE = ZERO
      DISPLAY fname " " lname
      DISPLAY "Update?"
      ACCEPT reply
      IF reply = "y"
         DISPLAY "New last name?"
         ACCEPT lname
         EXEC SQL
            UPDATE staff

            SET last_name=:lname WHERE CURRENT OF c1
         END-EXEC
         DISPLAY "update sqlcode=" SQLCODE
       END-IF
   END-IF
END-PERFORM

EXEC SQL DISCONNECT ALL END-EXEC
STOP RUN.
```

7.13.1 Limitations

You cannot use host arrays with positioned update. The other form of UPDATE used in standard SQL statements is known as a searched update. See the Help for more details about setting DECLARE CURSOR options and examples.

7.14 Using OpenESQL with Web and Application Servers

This section describes what you must do to use OpenESQL in environments controlled by Web and Application Servers, such as IIS, MTS, COM+, CICS and Tuxedo.

All servers differ, but the principle described here is common to them all.

7.14.1 Thread Safety

OpenESQL is thread safe. Normally all threads in an application share SQL resources such as connections and cursors. When running with an application server, however, threads will be scheduled to handle requests from different users. Therefore, you must use the directive:

```
SQL(THREAD=ISOLATE)
```

to ensure that each thread’s resources are isolated from each other.

7.14.2 Connection Management

In most environments, the application server will manage a pool of connections, which means that actual connect and disconnect requests to the database will be rare. When an application runs, it will reuse an existing connection. In most cases, connection pooling will be managed by the ODBC Driver Manager and will be transparent to the application. In other cases, the application server itself will manage the connection pool, and the application must use a "set connection" statement before it does anything else.

Where the application uses OpenESQL connect and disconnect statements, and it is not clear that the application server itself enables ODBC connection pooling, it might be worth experimenting with the SQL(CONNECTIONPOOL=...) directive. However, you are unlikely to need to do this.

7.14.3 Transactions

In many cases, the application server will provide transaction management. This will be determined when your component is placed under the control of the application server. If the application server is not providing transaction management, it must use OpenESQL COMMIT and ROLLBACK statements to manage transactions. If, however, the application server is providing transaction management, you must do the following:

- If connection management is not provided by the application server (that is, your application uses connect and disconnect statements), you must use the SQL(AUTOCOMMIT) directive. This does not mean that your SQL statements will be committed automatically, rather that your application will not be handling transactions by COMMIT and ROLLBACK statements.
- Your application must not use OpenESQL COMMIT and ROLLBACK statements. Instead, your application server will provide API calls or return codes for you to signal success or failure.

If you are using MTS or COM+, the default transaction isolation level, when the application server is managing transactions, can be serialized. This might cause excessive locking, and reduce concurrency. Your application must be prepared to deal with its transaction being aborted when the application server attempts to resolve a deadlock. To ease these problems, you can use the following statement:

```
exec sql set transaction isolation read committed end-exec
```

to set a less stringent isolation level. In this case, the statement must be the first statement to be executed after the CONNECT statement. Note that when SQL(AUTOCOMMIT) is not used, a commit or rollback is required immediately prior to executing a SET TRANSACTION ISOLATION statement.

7.14.4 User Accounts, Schemas and Authentication

When running in an application server environment, the user account in which your application executes might be different from the account used for development. ODBC data sources set up as user data sources might, therefore, not be available. You will probably find it more convenient to set up data sources as system data sources on deployment systems. When a file-based data source is used, the database files must be accessible by the account used by the application server. When accessing the database, particularly if integrated security is used (that is, when the DBMS uses the same account number as the operating system), the default schema might be different from that used during development. This might well be what is intended, if different schema names are used for development and deployment. Alternatively, it might mean that tables are no longer visible to the application. It must either use explicit owner qualification, or execute a database-specific statement, which will select the correct schema to be the default.

7.14.5 Transaction Wrapper Sample

The following is an example of an transaction wrapper generated by the OCX Wizard modified to include the OpenESQL logic to handle the following scenarios using a SQL Server data source:

- standalone transaction with no MTS/COM+ involved
- component managed by MTS/COM+ but without MTS/COM+ handling transactions
- component and transactions handled by MTS/COM+

```
$set ooctrl(+p) sql(thread=isolate autocommit)
      *>-----------------------------------------------------------
      *> Class description
      *>-----------------------------------------------------------
       class-id. cblsqlwrapper
                 inherits from olebase.
       object section.
       class-control.
           cblsqlwrapper is class "cblsqlwrapper"
```

```
*> OCWIZARD - start list of classes
     objectcontext is class "objectcontext"
     olebase is class "olebase"
     oleSafeArray is class "olesafea"
     oleVariant is class "olevar"
*> OCWIZARD - end list of classes
*>---USER-CODE. Add any additional class names below.
     *>-------------------------------------------------------------
 working-storage section. *> Definition of global data
*>-------------------------------------------------------------

*>-------------------------------------------------------------
 class-object.   *> Definition of class data and methods
*>-------------------------------------------------------------
 object-storage section.

*> OCWIZARD - start standard class methods
*>-------------------------------------------------------------
*> Return details about the class.
*> If you have a type library, theClassId and theInterfaceId
*> here MUST match.
*> theProgId must match the registry entry for this class
*>   (a zero length string implies using the class file name)
*> theClassId must match the CLSID stored in the registry.
*> theVersion is currently ignored (default 1 used).
*>-------------------------------------------------------------
 method-id. queryClassInfo.
 linkage section.
 01 theProgId            pic x(256).
 01 theClassId           pic x(39).
 01 theInterfceId        pic x(39).
 01 theVersion           pic x(4) comp-5.
 01 theDescription       pic x(256).
 01 theThreadModel       pic x(20).
 procedure division using by reference theProgId
                          by reference theClassId
                          by reference theInterfceId
                          by reference theVersion
                          by reference theDescription
                          by reference theThreadModel.
   move z"{3EADD92C-06C5-46F2-A2E0-7EB0794C14DF}" to theClassId
   move z"{5BF3F966-9932-4835-BFF6-2582CA2592AD}" to theInterfceId
   move z"Description for class cblsqlwrapper"
        to theDescription
   move z"Apartment" to theThreadModel
   exit method.
 end method queryClassInfo.
.
```

```
      *>-------------------------------------------------------------
      *> Return details about the type library - delete if unused.
      *> theLocale is currently ignored (default 0 used).
      *> theLibraryName is a null terminated string used for auto
      *> registration, and supports the following values:
      *>    <no string> - Library is embedded in this binary
      *>    <number>    - As above, with this resource number
      *>    <Path>      - Library is at this (full path) location
      *>-------------------------------------------------------------
       method-id. queryLibraryInfo.
       linkage section.
       01 theLibraryName        pic x(512).
       01 theMajorVersion       pic x(4) comp-5.
       01 theMinorVersion       pic x(4) comp-5.
       01 theLibraryId          pic x(39).
       01 theLocale             pic x(4) comp-5.
       procedure division using by reference theLibraryName
                                by reference theMajorVersion
                                by reference theMinorVersion
                                by reference theLibraryId
                                by reference theLocale.
         move 1 to theMajorVersion
         move 0 to theMinorVersion
         move z"{24207F46-7136-4285-A660-4594F5EE7B87}" to theLibraryId
         exit method.
       end method queryLibraryInfo.

      *>-------------------------------------------------------------

      *> OCWIZARD - end standard class methods

       end class-object.

      *>-------------------------------------------------------------
       object.         *> Definition of instance data and methods
      *>-------------------------------------------------------------
       object-storage section.

      *> OCWIZARD - start standard instance methods
      *> OCWIZARD - end standard instance methods

      *>-----------------------------------------------------------------
       method-id. "RetrieveString".
       working-storage section.

       01 mfsqlmessagetext pic x(400).
       01 ESQLAction       pic x(100).
```

```
 COPY DFHEIBLK.

 COPY SQLCA.
*>...your transaction program name
 01 transactionPgm           PIC X(7) VALUE ’mytran’.


 local-storage Section.
 01 theContext              object reference.
 01 transactionStatusFlag   pic 9.
   88 transactionPassed     value 1.
   88 transactionFailed     value 0.
*>---USER-CODE. Add any local storage items needed below.

 01 ReturnValue             pic x(4) comp-5.
   88 IsNotInTransaction    value 0.

 01 transactionControlFlag  pic 9.
   88 TxnControlledByMTS    value 0.
   88 TxnNotControlledByMTS value 1.

 linkage Section.

*>...Info passed to transaction
 01 transaction-Info.
    05 transaction-Info-RC   pic 9.
    05 transaction-Info-data pic x(100).

*>...Info returned from transaction via
 01 transaction-Info-Returned pic x(100).


 procedure division using by reference transaction-Info
                             returning transaction-Info-Returned.

*>...initialisation code
     perform A-Initialise
     perform B-ConnectToDB
     if TxnNotControlledByMTS
        perform C-SetAutoCommitOff
     end-if

*>...set isolation level to override SQLServer default, serialize
     perform D-ResetDefaultIsolationLevel

*>...set cursor type to overrde the OpenESQL default, dynamic+lock
     perform E-ResetDefaultCursorType
```

```
*>...call the transaction
     perform F-CallTransaction

*>...finalisation code - issue Commit/Rollback if not controlled
*>...by MTS/COM+
     if TxnNotControlledByMTS
        if transactionPassed
           perform X-Commit
        else
           perform X-Rollback
     end-if

     perform Y-Disconnect

*>...Transaction Server - use setAbort if the method fails:
     if theContext not = null
        if transactionPassed
           invoke theContext "setComplete"
        else
           invoke theContext "setAbort"
        end-if
        invoke theContext "finalize" returning theContext
     end-if

     exit method
     .

 A-Initialise.

*>...Transaction Server - get the context we are running in
     invoke objectcontext "GetObjectContext" returning theContext

*>...check if this component is enlisted in an MTS transation
     if theContext = null
        set TxnNotControlledByMTS to true
     else
        invoke theContext "IsInTransaction" returning ReturnValue
        if IsNotInTransaction
           set TxnNotControlledByMTS to true
        else
           set TxnControlledByMTS    to true
        end-if
     end-if

*>...initialise program variables
     set transactionPassed to true

     INITIALIZE DFHEIBLK
     .

 B-ConnectToDB.

*>...connect to data source

     EXEC SQL
         CONNECT TO ’SQLServer 2000’ USER ’SA’
     END-EXEC

     if sqlcode  zero
        move z"connection failed " to ESQLAction
        perform Z-ReportSQLErrorAndExit
     end-if
     .

 C-SetAutoCommitOff.

     EXEC SQL
         SET AUTOCOMMIT OFF
     END-EXEC
     if sqlcode  zero
         move z"Set Autocommit Off failed " to ESQLAction
         perform Z-ReportSQLErrorAndExit
     end-if

     perform X-Commit
    .

 D-ResetDefaultIsolationLevel.
*> the default isolation level for SQLServer is "Serialized", so
*> here we reset it to something more appropriate

     EXEC SQL
         SET TRANSACTION ISOLATION READ COMMITTED
     END-EXEC
     if sqlcode  zero
         move z"set transaction isoation failed " to ESQLAction
         perform Z-ReportSQLErrorAndExit
     end-if
     .

 E-ResetDefaultCursorType.
*> the default cursor type for OpenESQL is dynamic + lock
*> the most efficient is a "client" or "firehose" cursor - this is
*> a cursor declared as forward + read only - doing this here will
*> set it as a default from now on.  If Forward causes a problem,
*> change the concurrency to fast forward (but note that it will
*> then no longer be a client cursor)

     EXEC SQL
         SET CONCURRENCY READ ONLY
     END-EXEC
     if sqlcode  zero
         move z"Set Concurrency Read Only" to ESQLAction
         perform Z-ReportSQLErrorAndExit
     end-if

     EXEC SQL
         SET SCROLLOPTION FORWARD
     END-EXEC
     if sqlcode  zero
         move z"Set Concurrancy Read Only" to ESQLAction
         perform Z-ReportSQLErrorAndExit
     end-if
     .

 F-CallTransaction.

*>...call the program to process the transaction
     move 0               to  transaction-Info-RC
     call tranactionPgm using dfheiblk transaction-Info

*>...check if processing was okay
     if transaction-Info-RC = 0
        set transactionPassed to true
     else
        set transactionFailed to true
     end-if
     .

 X-Commit.

     EXEC SQL
         COMMIT
     END-EXEC
     if sqlcode  zero
         move z"Commit failed " to ESQLAction
         perform Z-ReportSQLErrorAndExit
     end-if
     .

 X-Rollback.

     EXEC SQL
         ROLLBACK
     END-EXEC
     if sqlcode  zero
         move z"Rollback failed " to ESQLAction
         perform Z-ReportSQLErrorAndExit
     end-if
     .

 Y-Disconnect.

     EXEC SQL
         DISCONNECT CURRENT
     END-EXEC
     if sqlcode  zero
         move z"Disconnect failed " to ESQLAction
         perform Z-ReportSQLErrorAndExit
     end-if
     .

 Z-ReportSQLErrorAndExit.

     move spaces to transaction-Info-Returned
     string ESQLAction delimited by x"00"
            "SQLSTATE = "
            SQLSTATE
            "  "
            mfsqlmessagetext
            into transaction-Info-Returned
     end-string

     exit method
     .

 exit method.
 end method "RetrieveString".
*>---------------------------------------------------------------

 end object.
 end class cblsqlwrapper.
```

See the ***Distributed Computing*** book for additional details on setting up MTS/COM+ or WebSphere transactions.

7.15 XML Support

Net Express comes with XML ODBC drivers, which you can set up to access XML files just like any ODBC data source. You can then build SQL statements using the OpenESQL Assistant from the XML data source.

7.15.1 PERSIST Statement

To help you in converting information to XML, OpenESQL has added the PERSIST statement which allows you to save information defined in a cursor SELECT statement as XML files. The syntax is:

```
PERSIST cursor_name TO xml_destination
```

where xml_destination may be an identifier, a host variable or a literal enclosed in single or double quotes. The cursor must also have SCROLLOPTION set to static. For example:

```
01  hv  pic x(50).
procedure-division.

*> set whenever clause to handle sql errors
exec sql whenever sqlerror goto sql-error end-exec
exec sql whenever sqlwarning perform sql-warning end-exec

*> connect to data source
exec sql connect to "data source"  end-exec

*> declare static cursor with column info you want to save to xml file
exec sql
  declare c static cursor for
    select * from emp
end-exec

*> open cursor
exec sql open c end-exec

*> save data to xml file using double quoted literal
exec sql
  persist c to "c:\XML Files\xmltest1.xml"
end-exec
```

```
 *> save data to xml file using single quoted literal
 exec sql
   persist c to ’c:\XML Files\xmltest2.xml’
 end-exec

 *> save data to xml file using a host variable
 move "c:\XML Files\xmltest3.xml" to hv
 exec sql
  persist c to :hv
end-exec

 *> close the cursor
 exec sql close c end-exec

 *> disconnect from datasource
 exec sql disconnect current end-exec

 goback.
```

Note: Data Direct ODBC version 3.70 or later drivers are required to use this statement.

8 OpenESQL Assistant

The OpenESQL Assistant is an interactive tool that makes it easy for you to:

- Prototype SQL SELECT statements and test them against your database.
- Design SQL INSERT, UPDATE and DELETE statements.

Once you have created your SQL queries, you can use the OpenESQL Assistant to insert them into your Net Express COBOL code. All you have to do is open the appropriate project and program, and the OpenESQL Assistant will insert the SQL query at the current insertion point. And if you want it to, the OpenESQL Assistant will even create and insert any auxiliary code made necessary by the insertion of your SQL queries.

This chapter is in the form of a tutorial which shows you how to:

- Specify OpenESQL Assistant Options
- Start the OpenESQL Assistant
- Connect to a data source
- Select a table
- Select a column
- De-select a column
- Select all the columns in a table
- De-select a table
- Display column details
- Create a new query
- Select a different table
- Change the query type
- Connect to a different data source

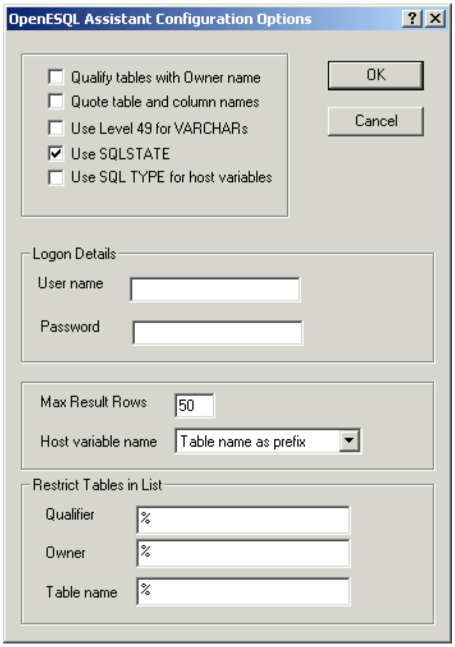
- Run a select query
- Specify search criteria
- Sort data retreived
- Disconnect from a data source
- Create a table join
- Add an embedded SQL statement to your program
- Add auxiliary code to your program
- Change a select(cursor) query to do an Array FETCH
- Generate Query as Stored Procedure
- Close the OpenESQL Assistant

8.1 Setting OpenESQL Assistant Options

You can change certain default settings to control how OpenESQL Assistant generates a query via the OpenESQL Assistant Configuration Options dialog box. To do this, click **Embedded SQL** on the **Options** menu.

The following dialog box is displayed:

Figure 8-1. OpenESQL Assistant Configuration Options



OpenESQL Assistant Configuration Options

☐ Qualify tables with Owner name
☐ Quote table and column names
☐ Use Level 49 for VARCHARs
☑ Use SQLSTATE
☐ Use SQL TYPE for host variables

Logon Details

User name:

Password:

Max Result Rows: 50

Host variable name: Table name as prefix

Restrict Tables in List

Qualifier: %

Owner: %

Table name: %

The following options can be set:

- Qualify tables with owner name

  By default, OpenESQL Assistant builds all queries unqualified. However, if the data source has multiple tables with the same table name, you might want to build the query with a specific owner. Check this option and the table list will include the owner name in parentheses after the table name in the tree view. Then select the table name with the owner that you want to use in the query and all table names in the query will be qualified with the owner name.

- Quote table and column names

  By default, OpenESQL Assistant does not enclose column and table names with the quote identifier associated with the data source unless the column or table name has embedded blanks, special character such as "$" or DBCS characters in it. You can force all table and column names to be enclose by checking this option.

- Use level 49 for VARCHARs

  By default, OpenESQL Assistant generates a host variable as a PIC X(n) field for VARCHAR columns. When data is mapped to the host variable, it is null terminated. However, if you'd like to get a length of the column data returned, you need to check this option and OpenESQL Assistant will generate the host variable with two level-49 variables; one for the length of the data mapped and one for the actual text data.

- Use SQLSTATE

  By default, OpenESQL Assistant generates SQL statements using SQLCODE checking. Check this option if you want OpenESQL Assistant to generate SQL statements using SQLSTATE checking.

- Use SQL TYPE for host variables

  By default, OpenESQL Assistant now generates COBOL host variables using SQL TYPE definitions where appropiate. This allows the SQL precompiler to better determine the type of useage a host variable will be used for. You can still generate host variables the old way by unchecking this option. You should also uncheck this option if you are using the copybook with Array Fetches.

- Logon details

  If you always connect to the same data source or always use the same logon information to connect to data sources, you can save that information using the following fields:

  - User name

    The userid or logon id required to connect to the data source.

  - Password

    The password required to connect to the data source for the User name specified previously. Information entered in this field is not displayed on entry but instead is displayed as asterisks .

- Max result rows

  By default, OpenESQL Assistant only returns the first 50 rows when you run a query. This is to prevent queries from returning huge number of rows that may crash your machine or overload your network if testing with network servers since the purpose of this tool is to interactively build/test queries and you may not yet have specified all the criteria required. You can set this value to a higher number but choose a reasonable number that reflects the amount of data you expect to be returned or need to test for.

- Host variable name

  OpenESQL Assistant will create host variables using a combination of the column name and a prefix setting selected in this drop-down list. If the combination is either too long or contains invalid characters to be a valid COBOL name, the column number is used. To make a valid COBOL name, OpenESQL Assistant converts all underscores into hyphens. You can choose from the following options:

  - Table name as prefix

    Use the table name as a prefix to column name. The combination must be 31 characters or fewer or column numbers will be substituted for the column name. This is the default prefix setting.

  - No prefix

    Generate variable names using only the column name. If name is greater than 31 characters or contains invalid COBOL characters, the column number is generated with a prefix of FLD.

  - Alphabetic character prefix

    Generate variable names using "A" as prefix for the first table selected, "B" as the prefix for the second table selected, and so on.

- Restrict tables in list

  By default, OpenESQL Assistant will use the tree view to show all tables, views, aliases, and synonymns found in a data source. For certain types of data sources, this could create a very large list. To improve response time in building this list, you can restrict this list by qualifying what tables to return. For example, you could request that only tables that begin with OWNER that starts with "XYZ%" be returned. In general, this is comparable to the LIKE function where:

  - The underscore character (_) represents any single character.

  - The percent sign (%) represents a string of zero or more characters.

- Any other character represents itself.

There are three fields that you can use to restrict the list:

- Qualifier
- Owner
- Table name

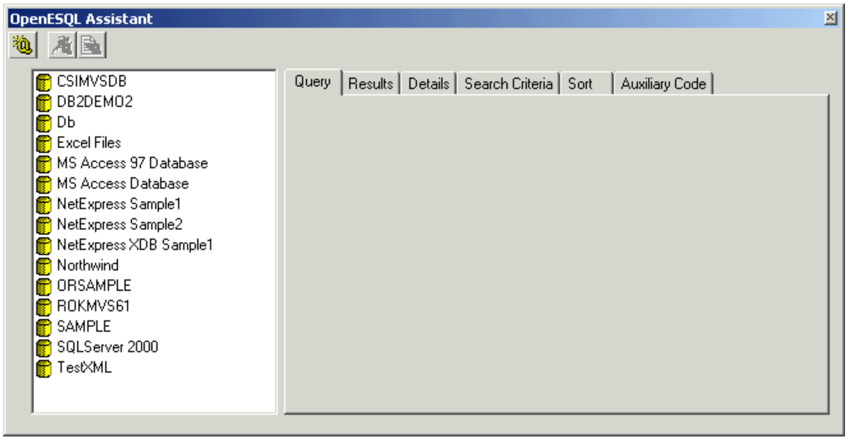
Not all data sources support all three, so you should look at the data source properties to determine which fields are supported by the data source you want to connect to.

In the example above, you would enter in the Owner field "XYZ%", click **OK** to save you settings and then connect to the data source. If your restrictions results in no tables, OpenESQL Assistant will return **"< no tables selected >"** as the table name.

8.2 Starting the OpenESQL Assistant

To start the OpenESQL Assistant, select **Dockable Windows** from the Net Express **View** menu. Select the **OpenESQL Assistant** checkbox and close the **Dockable Windows** dialog box.

Figure 8-2. OpenESQL Assistant



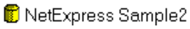
The OpenESQL Assistant window is a dockable window, although you can toggle this facility by right-clicking in the grey area immediately below the title bar and checking (or unchecking) **Allow docking**. You can also hide the window by right-clicking and checking **Hide**. If you are unfamiliar with docking or hiding windows, look up **Docking** in the online help file and select **Rearranging views and dockable windows**.

8.3 Connecting to a Data Source

Once you have started the OpenESQL Assistant, a list of all the ODBC data sources that you have set up is displayed. In *Figure 8-1* above, for example, **Excel_Files** and **Oracle_Database** are two of the existing data sources that are displayed by the OpenESQL Assistant.

To connect to a data source, double-click on its name, or on the appropriate data source icon, for example:

Figure 8-3. Data Source Icon and Name



NetExpress Sample2

Note: You can connect to one data source only at any one time. For information on changing the data source to which you are connected to, see the section *Connecting to a Different Data Source*.

Depending on how the data source is set up, you may be prompted to enter one or more of the following:

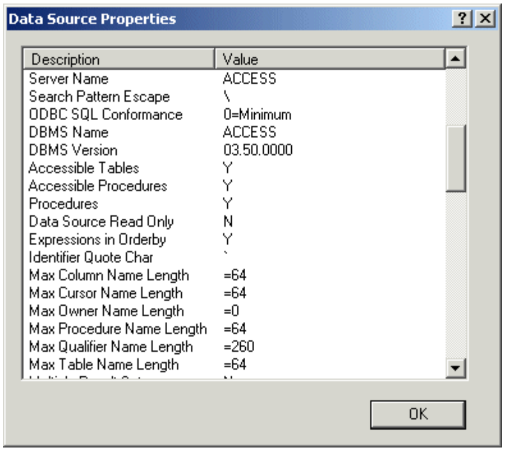
- a user-id
- a password
- a database name

If you haven't set up any data sources yourself, you can use one of the sample data sources that is set up automatically when you install Net Express. One of these, **NetExpress Sample2** points to a sample

Microsoft Access database, **sample.mdb** which is supplied with Net Express and installed in the **demo\smpldata\access** directory. In the remainder of this chapter, references to the Microsoft Access data source **NetExpress Sample2** can be simply exchanged for **NetExpress XDB Sample1** if you want to examine an XDB database. This applies equally to the directory structures shown within the Figures.

You can look at the data source properties after you are connected to a data source by right-clicking on the data source and clicking **Data Source Properties** on the popup menu.

Figure 8-4. Data Source Properties



Description	Value
Server Name	ACCESS
Search Pattern Escape	\
ODBC SQL Conformance	0=Minimum
DBMS Name	ACCESS
DBMS Version	03.50.0000
Accessible Tables	Y
Accessible Procedures	Y
Procedures	Y
Data Source Read Only	N
Expressions in Orderby	Y
Identifier Quote Char	`
Max Column Name Length	=64
Max Cursor Name Length	=64
Max Owner Name Length	=0
Max Procedure Name Length	=64
Max Qualifier Name Length	=260
Max Table Name Length	=64

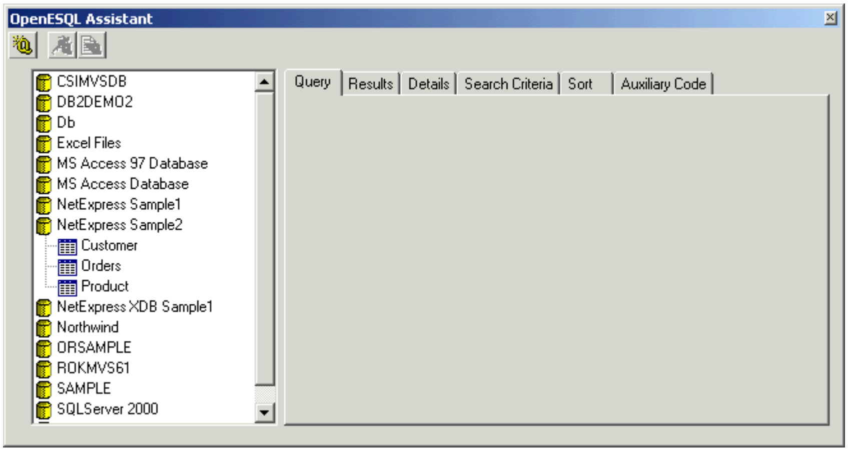
Information about the data source includes the identifier quote character that is used, and the maximum size of column and table names. In this example, the Owner Name Length is 0, which indicates that this data source does not support the OWNER field, so you would not use this field to restrict list of tables in the Setup function discussed previously.

The remainder of this tutorial assumes that you are using the **sample.mdb** database and this is the database which is shown in all of the illustrations.

8.4 Selecting a Table

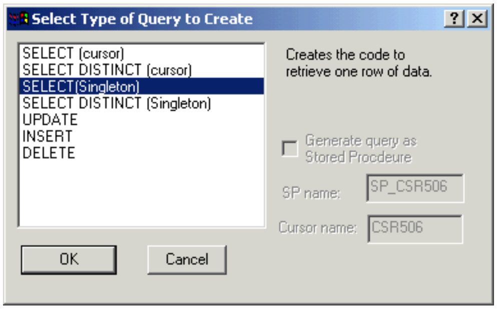
Once you have connected to a data source, the names of all the tables in that data source are displayed underneath the data source name:

Figure 8-5. Selecting a Table



You can select a table by double-clicking on its name. You will be prompted to select the type of query that you want. At this point, it doesn't matter which type of query you select, so accept the default (a singleton select) by simply clicking on the **OK** button:

Figure 8-6. Selecting a Query


Select Type of Query to Create
SELECT (cursor)
SELECT DISTINCT (cursor)
SELECT(Singleton)
SELECT DISTINCT (Singleton)
UPDATE
INSERT
DELETE
Generate query as Stored Procdeure
OK
Cancel

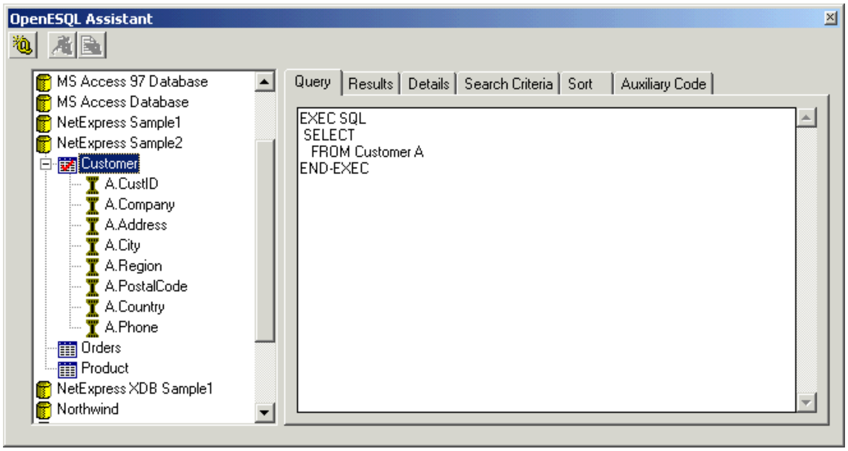
Creates the code to retrieve one row of data.

SP name: SP_CSR506

Cursor name: CSR506

Once you have selected a query, the COBOL code for the query you have selected is automatically generated and displayed under the **Query** tab. At the same time, a list of all the columns in the table is displayed underneath the table name:

Figure 8-7. Displaying Columns



```
EXEC SQL
 SELECT
  FROM Customer A
END-EXEC
```

You will notice that a table alias is generated automatically by the OpenESQL Assistant:

```
SELECT FROM Customer A
```

As this is the first table that you have selected, the alias used is the letter "A". If you were to select a second table, the OpenESQL Assistant would generate an alias of "B" and so on.

Notice also that each column name is prefixed by its alias (A.CustID, A.Company). This enables you to distinguish the columns in one table from those in another.

Notes:

- Some databases uses special characters in the names of system tables. For example, Oracle can use system tables whose names contain a dollar ($). Therefore, the OpenESQL Assistant sets the option to enclose column and table names in quotes automatically whenever the assistant sees a name that would be illegal if not enclosed in quotes.

- If the name generated from the column name (including any prefixes and suffixes) would be illegal in COBOL because it is longer than 31 characters or contains illegal characters, a host variable is generated using the column number (for example, COL005) rather than the column name.

8.4.1 Selecting a Column

To select a column, simply double-click on the column name. Notice that the COBOL code is automatically updated.

8.4.2 De-selecting a Column

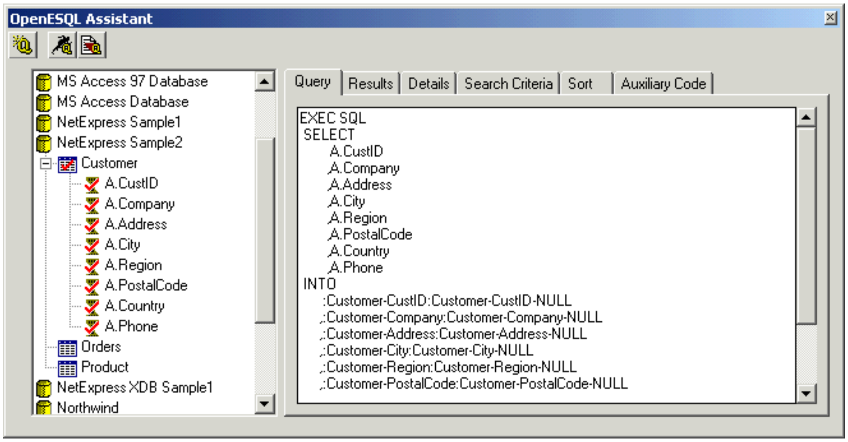
You can de-select a selected column by double-clicking on it. Each time you select or de-select a column, the COBOL code is automatically updated.

8.4.3 Selecting all the Columns in a Table

You can select all of the columns in a table by:

- Right-clicking on the table name
- Clicking on **Select All Columns**

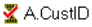
Figure 8-8. Selecting Columns



```
EXEC SQL
 SELECT
        A.CustID
        A.Company
        A.Address
        A.City
        A.Region
        A.PostalCode
        A.Country
        A.Phone
 INTO
       :Customer-CustID:Customer-CustID-NULL
      ,:Customer-Company:Customer-Company-NULL
      ,:Customer-Address:Customer-Address-NULL
      ,:Customer-City:Customer-City-NULL
      ,:Customer-Region:Customer-Region-NULL
      ,:Customer-PostalCode:Customer-PostalCode-NULL
```

You can always tell whether a column is currently selected because, if it is, the column icon is checked:

Figure 8-9. Column Icon

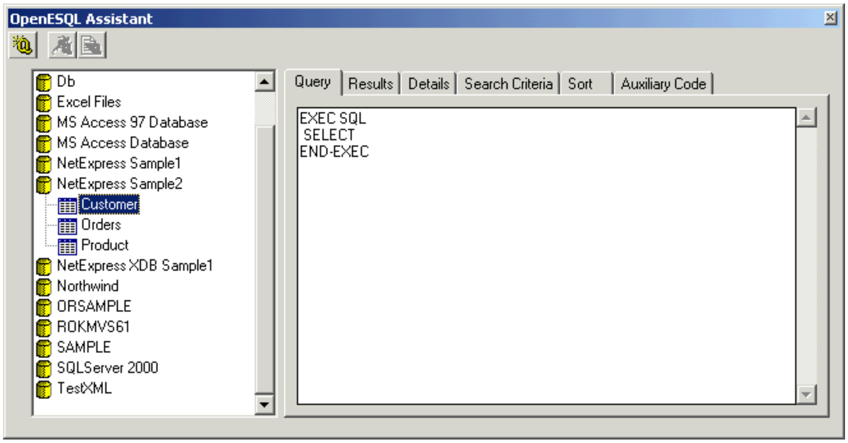

A.CustID

8.5 De-selecting a Table

You can de-select a table by double-clicking on it a second time. If you have already selected columns from this table, you will be prompted to confirm that you want to de-select it. Click on the **Yes** button.

The table is de-selected and the code that was generated is amended:

Figure 8-10. De-selecting a Table

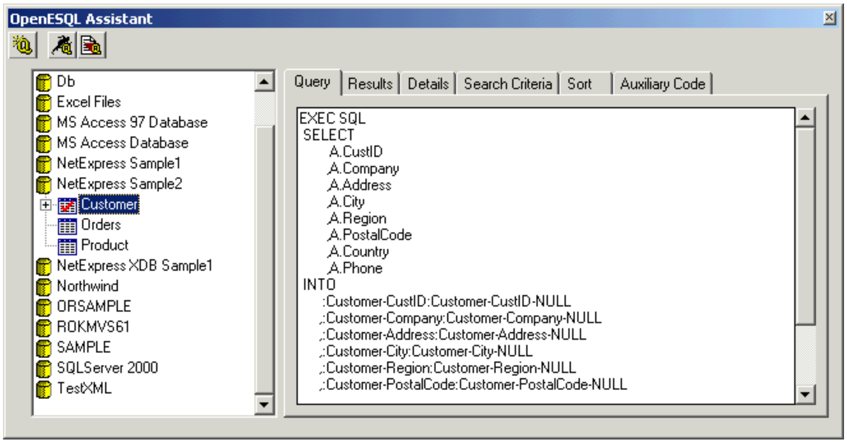


```
EXEC SQL
 SELECT
END-EXEC
```

Note that you must double-click on the table name or table icon to de-select the table. Clicking on the minus sign (-) simply toggles the view of the table (+ displays all the columns, - hides them).

For example, select the **Customer** table again by double-clicking on it. Select all the columns by right-clicking on the table name and clicking **Select All Columns**. Now click on the minus sign to hide the columns:

Figure 8-11. Hiding the Columns in a Selected Table

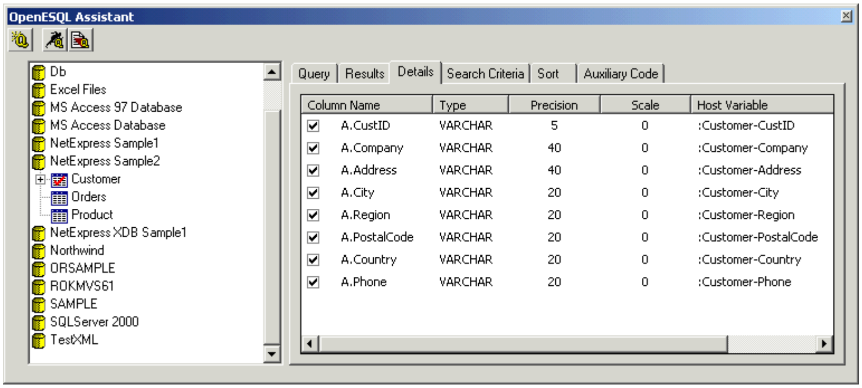


```
EXEC SQL
 SELECT
      A.CustID
      A.Company
      A.Address
      A.City
      A.Region
      A.PostalCode
      A.Country
      A.Phone
 INTO
     :Customer-CustID:Customer-CustID-NULL
    ,:Customer-Company:Customer-Company-NULL
    ,:Customer-Address:Customer-Address-NULL
    ,:Customer-City:Customer-City-NULL
    ,:Customer-Region:Customer-Region-NULL
    ,:Customer-PostalCode:Customer-PostalCode-NULL
```

8.6 Displaying Column Details

To see more detailed information on the columns in a table, simply click on the **Details** tab.

Figure 8-12. Displaying Column Details



	Column Name	Type	Precision	Scale	Host Variable
☑	A.CustID	VARCHAR	5	0	:Customer-CustID
☑	A.Company	VARCHAR	40	0	:Customer-Company
☑	A.Address	VARCHAR	40	0	:Customer-Address
☑	A.City	VARCHAR	20	0	:Customer-City
☑	A.Region	VARCHAR	20	0	:Customer-Region
☑	A.PostalCode	VARCHAR	20	0	:Customer-PostalCode
☑	A.Country	VARCHAR	20	0	:Customer-Country
☑	A.Phone	VARCHAR	20	0	:Customer-Phone

The information that is displayed is as follows:

- Column Name

  The column name of each column in the table is displayed. Notice that each column name is prefixed by the table alias, for example, CustID is shown as A.CustID. A tick in the box to the left of the column name indicates that the column is currently selected.

- Type

  The data type of the column is displayed. These are the data types used by the data source to which you are connected. The data type of a column must match the COBOL picture clause of the host variable that is used to pass values for that column to and from the data source.

  You can get the OpenESQL Assistant to generate a copyfile (in the current directory and called ***tablename*****.cpy**) which declares all the necessary host variables with the correct COBOL picture clauses (that is, matching the data type of the table columns). See the section *Adding Auxiliary Code* below.

- Precision

  The total number of digits in the column is displayed. Precision is only displayed for those columns for which it is relevant. If the column is a text column, precision shows the length of the column.

- Scale

  The number of digits to which the column is rounded is displayed. Scale is only displayed for those columns for which it is relevant.

- Host Variable

  The OpenESQL Assistant automatically generates a host variable for each column. The host variable name takes the form:

  :<*table-name*>**-**<*column-name*>

  For example, the host variable generated by the OpenESQL Assistant for the CustID column is `:Customer-CustID`, that for the City column is `:Customer-City` and that for the Phone column is `:Customer-Phone`.

- Indicator Variable

  As well as a host variable, the OpenESQL Assistant also generates an indicator variable for each column. The indicator variable name takes the form:

  :<*table-name*>**-**<*column-name*>**-NULL**

  For example, the indicator variable generated by the OpenESQL Assistant for the CustID column is `:Customer-CustID-NULL`, that for the City column is `:Customer-City-NULL` and that for the Phone column is `:Customer-Phone-NULL`.

8.7 Creating a New Query

The following sections explain the ways in which you can modify an existing query or create a new query.

8.7.1 Selecting a Different Table

To select a different table you must first de-select the currently selected table by double-clicking on it (if you have any columns currently selected you will be prompted to confirm that you want to de-select the table) and then select the new table by double-clicking on that. You will be prompted to select a query type.

8.7.2 Changing the Query Type

You cannot change the query type without first de-selecting the currently selected table and then either re-selecting it (if you want to create a different type of query for the same table) or selecting a new table. Once you have selected a table, you will be prompted to select a query. Select the query that you want and click on the **OK** button.

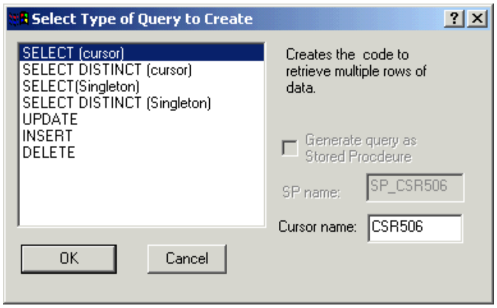
8.7.3 Connecting to a Different Data Source

If you want to connect to a different data source, click on the **Create a New Query** button, . This disconnects you from the current data source thus allowing you to select a new data source by double-clicking on its name. If you attempt to connect to a new data source without first disconnecting from the current data source, an error message is displayed informing you that you cannot connect to a second data source.

8.8 Running a Select Query

Connect to the **NetExpress Sample2** data source and select the **Customer** table. You will be prompted for a query type. Select **SELECT (cursor)**, enter "CSR506" in the **Cursor name** entry field and click on the **OK** button.

Figure 8-13. Selecting a SELECT (cursor) Query



Creates the code to retrieve multiple rows of data.

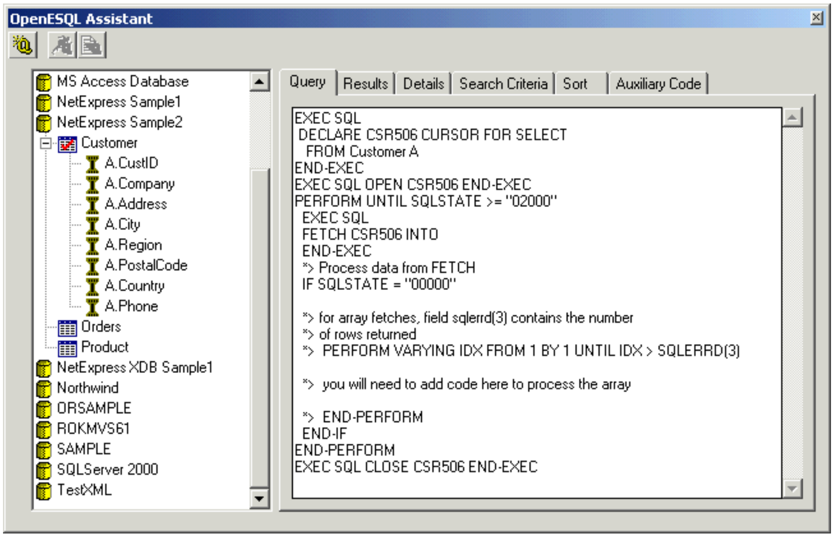
□ Generate query as Stored Procdeure

SP name: SP_CSR506

Cursor name: CSR506

You will notice that the COBOL code for the SELECT statement is automatically generated and displayed:

Figure 8-14. SELECT (cursor) Query Code



```
EXEC SQL
 DECLARE CSR506 CURSOR FOR SELECT
   FROM Customer A
END-EXEC
EXEC SQL OPEN CSR506 END-EXEC
PERFORM UNTIL SQLSTATE >= "02000"
  EXEC SQL
  FETCH CSR506 INTO
  END-EXEC
  *> Process data from FETCH
  IF SQLSTATE = "00000"

  *> for array fetches, field sqlerrd(3) contains the number
  *> of rows returned
  *>  PERFORM VARYING IDX FROM 1 BY 1 UNTIL IDX > SQLERRD(3)

  *>   you will need to add code here to process the array

  *>  END-PERFORM
  END-IF
END-PERFORM
EXEC SQL CLOSE CSR506 END-EXEC
```

This code includes:

- A DECLARE CURSOR statement which creates the cursor to be used in the SELECT:

  ```
  DECLARE CSR506 CURSOR FOR SELECT FROM Customer A
  ```

- An OPEN statement which executes the SELECT statement specified in the corresponding DECLARE CURSOR statement:

  ```
  OPEN CSR506
  ```

- A FETCH statement to retrieve the next row from the cursor's results set:

  ```
  FETCH CSR506 INTO
  ```

- A CLOSE statement to close the cursor:

  ```
  CLOSE CSR506
  ```

As the query stands, it would cause a syntax error - click on the **Run the Query** button, , to see the error displayed - because it does not specify which column(s) to select or what to select them into.

To select a column, double-click on it. Select **A.CustID**. For each column that you select, the automatically generated code is updated as follows:

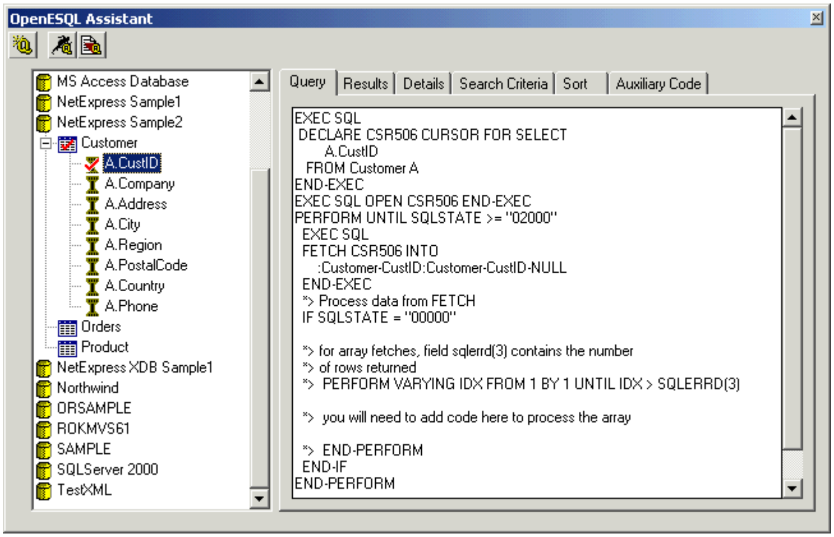
- The column is added to the DECLARE CURSOR statement, for example:

  ```
  DECLARE CSR506 CURSOR FOR SELECT
     A.CustID
  FROM Customer A
  ```

- A host variable (into which the column will be read) is added to the INTO clause of the FETCH statement, for example:

  ```
  FETCH CSR506 INTO
     :Customer-CustID:Customer-CustID-NULL
  ```

Figure 8-15. Adding Columns to a SELECT Statement

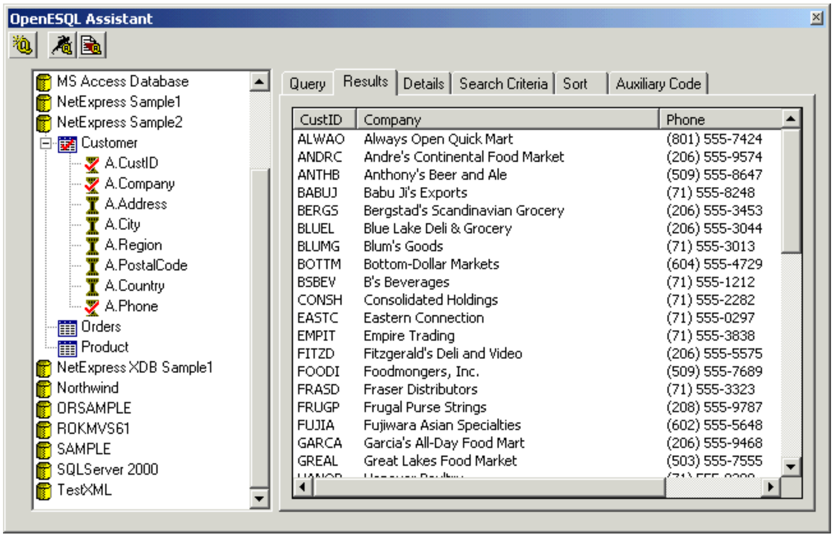


```
EXEC SQL
 DECLARE CSR506 CURSOR FOR SELECT
      A.CustID
  FROM Customer A
END-EXEC
EXEC SQL OPEN CSR506 END-EXEC
PERFORM UNTIL SQLSTATE >= "02000"
  EXEC SQL
  FETCH CSR506 INTO
     :Customer-CustID:Customer-CustID-NULL
  END-EXEC
  *> Process data from FETCH
  IF SQLSTATE = "00000"

  *> for array fetches, field sqlerrd(3) contains the number
  *> of rows returned
  *>  PERFORM VARYING IDX FROM 1 BY 1 UNTIL IDX > SQLERRD(3)

  *>  you will need to add code here to process the array

  *>  END-PERFORM
  END-IF
END-PERFORM
```

Now select **A.Company** and **A.Phone**. Once you have selected all the columns that you want, you can run the query by clicking on the **Run**

the Query button, . The OpenESQL Assistant automatically displays the results of the query:

Figure 8-16. Query Results



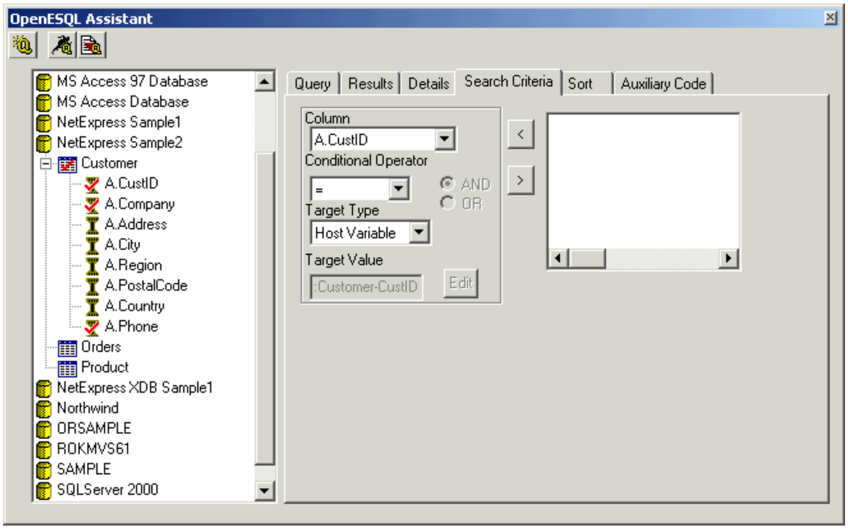
CustID	Company	Phone
ALWAO	Always Open Quick Mart	(801) 555-7424
ANDRC	Andre's Continental Food Market	(206) 555-9574
ANTHB	Anthony's Beer and Ale	(509) 555-8647
BABUJ	Babu Ji's Exports	(71) 555-8248
BERGS	Bergstad's Scandinavian Grocery	(206) 555-3453
BLUEL	Blue Lake Deli & Grocery	(206) 555-3044
BLUMG	Blum's Goods	(71) 555-3013
BOTTM	Bottom-Dollar Markets	(604) 555-4729
BSBEV	B's Beverages	(71) 555-1212
CONSH	Consolidated Holdings	(71) 555-2282
EASTC	Eastern Connection	(71) 555-0297
EMPIT	Empire Trading	(71) 555-3838
FITZD	Fitzgerald's Deli and Video	(206) 555-5575
FOODI	Foodmongers, Inc.	(509) 555-7689
FRASD	Fraser Distributors	(71) 555-3323
FRUGP	Frugal Purse Strings	(208) 555-9787
FUJIA	Fujiwara Asian Specialties	(602) 555-5648
GARCA	Garcia's All-Day Food Mart	(206) 555-9468
GREAL	Great Lakes Food Market	(503) 555-7555

8.9 Specifying Search Criteria

You can limit which rows are returned by a SELECT statement by specifying search criteria (a WHERE clause). The OpenESQL Assistant provides a special screen which enables you, quickly and easily, to specify the search criteria used in the WHERE clause.

To limit the number of rows returned by the SELECT statement, click on the **Search Criteria** tab:

Figure 8-17. Search Criteria Tab



OpenESQL Assistant

Column: A.CustID

Conditional Operator: =

● AND ○ OR

Target Type: Host Variable

Target Value: :Customer-CustID

To specify search criteria:

1 Select a column name from the **Column** list box. This list box displays the column name of every column in the currently selected table(s).

2 Select a conditional operator. Click on the down-pointing arrow to scroll through the valid conditional operators.

3 Select a **Target Type**. The target type must be a host variable, a literal, a special register or a column name.

4 Select, or enter, a **Target Value**. If you have selected column name as the target type, a list of all the valid column names generated by the OpenESQL Assistant is displayed in the target value list box. If you have selected host variable as the target type, a list of all the valid host variables generated by the OpenESQL Assistant is displayed in the target value list box. If you have selected either literal or special register, the **Edit** button to the right of the target value list box is enabled. This means that you can either enter a

value directly into the target value list box or you can click on the **Edit** button to display the **Literal Value Editor**.

Once you are happy with what you have selected, click on the right-pointing arrow (>). This moves your search criteria across into the right-hand pane and, at the same time, updates the COBOL code in the query window.

For example, you can limit the customer IDs returned by the SELECT statement created above as follows:

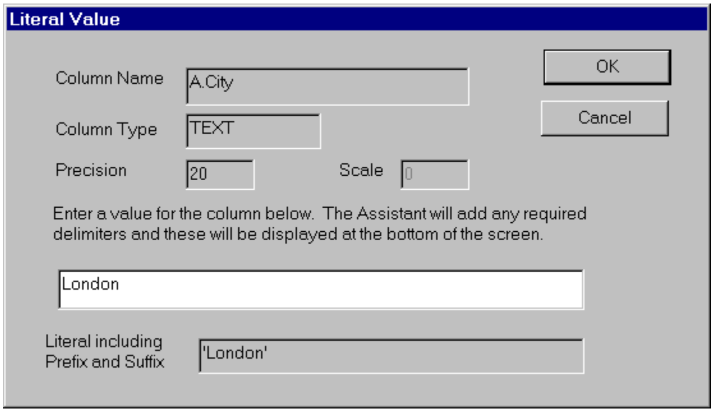
1 Select **A.City** in the **Column** list box.

2 Select = as the conditional operator.

3 Select **Literal** as the target type.

4 The **Edit** button to the right of the target value list box is now enabled. Click on it to display the **Literal Value Editor** dialog box:

Figure 8-18. Literal Value Editor



Literal Value

Column Name: A.City

Column Type: TEXT

Precision: 20

Scale: 0

Enter a value for the column below. The Assistant will add any required delimiters and these will be displayed at the bottom of the screen.

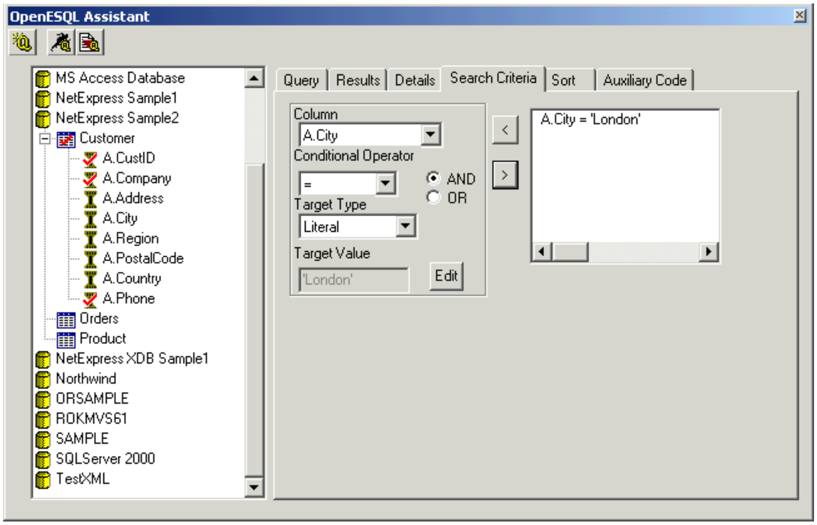
London

Literal including Prefix and Suffix: 'London'

5 Enter **London** as the target value. Notice that the OpenESQL Assistant automatically adds the correct delimiters (in this case single quotation marks). Click on the **OK** button.

6 Now click on the right-pointing arrow to move your search criterion across into the right-hand pane.

Figure 8-19. Specifying Search Criteria



Column: A.City

Conditional Operator: =

● AND ○ OR

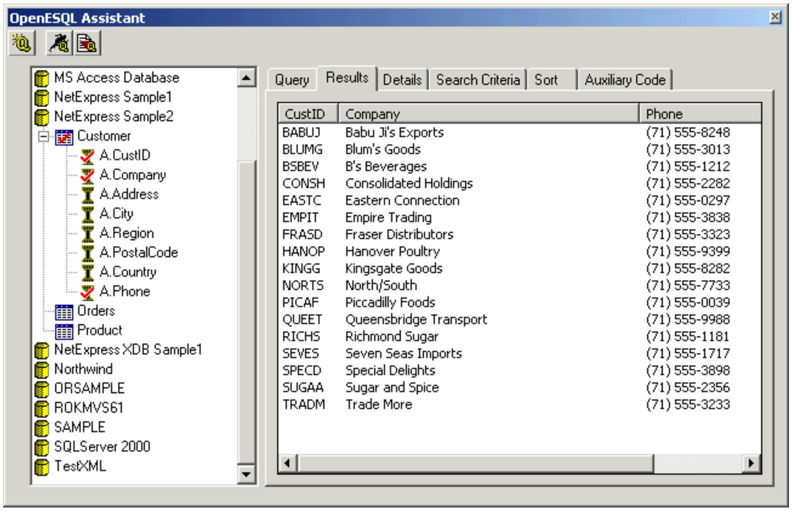
Target Type: Literal

Target Value: 'London'

A.City = 'London'

Now when you run this query again (click on the **Run the Query** button,), only the customer IDs of those customers with an entry of **London** in the **City** columm are displayed:

Figure 8-20. Limiting the Customer IDs by Specifying Search Criteria



CustID	Company	Phone
BABUJ	Babu Ji's Exports	(71) 555-8248
BLUMG	Blum's Goods	(71) 555-3013
BSBEV	B's Beverages	(71) 555-1212
CONSH	Consolidated Holdings	(71) 555-2282
EASTC	Eastern Connection	(71) 555-0297
EMPIT	Empire Trading	(71) 555-3838
FRASD	Fraser Distributors	(71) 555-3323
HANOP	Hanover Poultry	(71) 555-9399
KINGG	Kingsgate Goods	(71) 555-8282
NORTS	North/South	(71) 555-7733
PICAF	Piccadilly Foods	(71) 555-0039
QUEET	Queensbridge Transport	(71) 555-9988
RICHS	Richmond Sugar	(71) 555-1181
SEVES	Seven Seas Imports	(71) 555-1717
SPECD	Special Delights	(71) 555-3898
SUGAA	Sugar and Spice	(71) 555-2356
TRADM	Trade More	(71) 555-3233

Once you have specified the first search criterion and moved it to the right-hand pane, the second and subsequent criteria must follow a logical AND or logical OR which you select by checking the appropriate radio button.

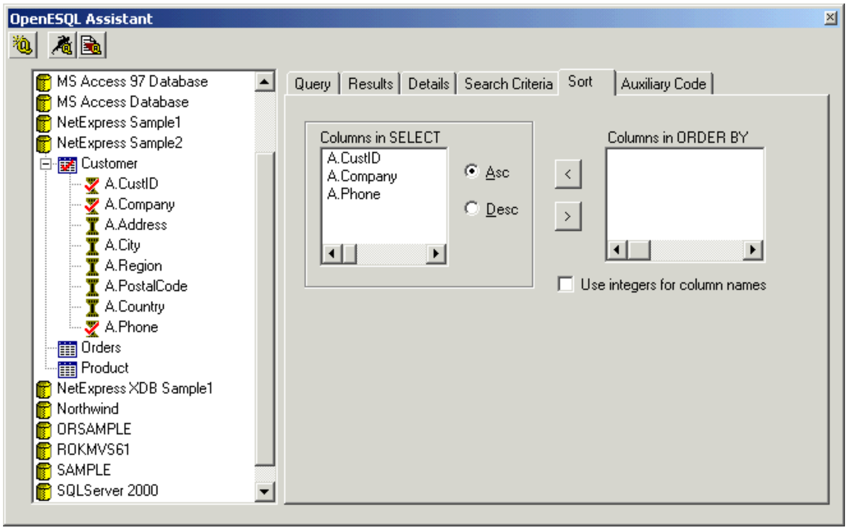
You can select a search criterion displayed in the right-hand pane and click on the left-pointing arrow (<). This removes the search criterion from the right-hand pane and places it in the appropriate boxes in the left-hand pane such that you can edit it. Moving search criteria from the right to the left-hand pane also removes the associated COBOL code from the automatically generated query.

8.10 Specifying Order Data is Retrieved

You can order the rows that are returned by a SELECT statement by specifying which columns to sort (an ORDER BY clause). The OpenESQL Assistant provides a dialog box that enables you to quickly and easily select which columns to order on in the ORDER BY clause.

To order data returned by the SELECT statement, click the **Sort** tab:

Figure 8-21. Sort Tab



Columns in SELECT: A.CustID, A.Company, A.Phone

● Asc ○ Desc

Columns in ORDER BY:

□ Use integers for column names

To specify which columns are to be in an ORDER BY clause:

1. Select a column name from the Column in SELECT list box. This list box displays the column name of every column that currently has been selected for the query.

2. Select a sort sequence for the column selected by clicking on either the Asc (ascending) or Desc (descending) radio buttonl operators.

3. Check **Use integers for column names** if you want the query to be generated using a number to represent the column name. The column names will still be displayed in the **Columns in ORDER BY** field, but the integer value of the column will be generated for the query.

Part 3: DB2

This part contains the following chapters:

- Chapter 9, “DB2”
- Chapter 10, “SQL Option for DB2”
- Chapter 11, “Stored Procedures”

9 DB2

This chapter describes how you can access a DB2 database from a COBOL program which contains embedded SQL statements and has been compiled and linked using Net Express.

The DB2 External Compiler Module (ECM) is an integrated preprocessor provided with Net Express and designed to work more closely with the Micro Focus COBOL Compiler. The DB2 ECM converts embedded SQL statements into the appropriate calls to DB2 database services.

9.1 Data Types

In addition to the data types described in the chapter *Data Types* as being supported, DB2 also supports the following data types:

9.1.1 Decimal

The DECIMAL data type describes a packed-decimal item, with or without a decimal point. In COBOL, such items can be declared either as COMP-3 or as PACKED-DECIMAL.

9.1.2 Additional Data Types

All additional data types must be declared using SQL syntax of the form:

```
>>--level_number--name-+------------+-SQL-+-----------+-->
                       |            |     |           |
                       +-USAGE-+----+     +-TYPE-+----+
                               |    |            |    |
                               +-IS-+            +-IS-+

 >--sql_type--+----------+--><
              |          |
              +-(-size-)-+
```

where

level_number is within the range 1 to 48

sql_type is one of the new SQL data types BLOB, CLOB, DBCLOB, BLOB-FILE, CLOB-FILE, DBCLOB-FILE, BLOB-LOCATOR, CLOB-LOCATOR, DBCLOB-LOCATOR or TIMESTAMP. TIMESTAMPs are not new to DB2 V2 and are provided as a convenience by the DB2 ECM.

size may only be specified for BLOBs, CLOBs and DBCLOBs, and is mandatory. It may be qualified with K (Kilobytes), M (Megabytes) or G (Gigabytes).

VALUE clauses are not permitted on the new SQL data types.

Depending on the *sql_type* specified, the actual data created may be an elementary or group item. The names of elements in the group item are generated automatically.

The table below shows the structure of the data items created using SQL syntax by showing the equivalent native COBOL definition. Note that although the same data is created in each case, the items must be declared using the SQL syntax in order to be recognised as acceptable host variables by the DB2 ECM. (This is because the COBOL definitions are ambiguous: various of the new SQL types, and existing group items which are expanded to individual host variables, are indistinguishable). All previously existing data types continue to be declared using normal

COBOL syntax. The only exception to this rule is TIMESTAMP, which may be declared using either form.

SQL syntax	Equivalent COBOL syntax

`01  MY-BLOB SQL BLOB(125M).`	`01  MY-BLOB.`
	`49  MY-BLOB-LENGTH PIC S9(9) COMP-5.`
	`49  MY-BLOB-DATA   PIC X(131072000).`

`03  A SQL CLOB(3K).`	`03  A.`
	`49  A-LENGTH PIC S9(9) COMP-5.`
	`49  A-DATA   PIC X(3072).`

`03  HV SQL DBCLOB(125).`	`03  HV.`
	`49  HV-LENGTH PIC S9(9) COMP-5.`
	`49  HV-DATA   PIC G(125).`

`01  B SQL BLOB-LOCATOR.`	`01  B PIC S9(9) COMP-5.`

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

`01  TS SQL TIMESTAMP.`	`01  TS PIC X(29).`

9.2 Compound SQL

Compound SQL is supported, including the extended form now available in DB2 V2. Note that incomplete Compound SQL statements are detected by the DB2 ECM and cause an error to be produced. However, DB2 may not always recover from this condition and valid SQL statements later in the program source may generate additional errors.

9.3 User Defined Functions

A program containing a reference to a User Defined Function (UDF) causes a separate module to be invoked; it contains user-supplied code which returns an appropriate value or values. The UDF code itself does not contain any SQL.

Running a program containing embedded SQL statements causes DB2 to be invoked and this in turn may invoke the UDF module. The declaration of the UDF should specify the language this module is written in. DB2 currently allows this to be C only, although on some platforms it is possible to write the module in COBOL. The following section demonstrates by use of example how this may be achieved. More complete descriptions of User Defined Functions and parameter descriptions are provided in the DB2 documentation.

User Defined Functions written in COBOL are not currently supported on UNIX.

Note: In a client/server configuration, the UDF module is invoked on the server and these restrictions apply to the server only - any client can access UDFs if the server is suitable.

The entry points in the UDF should be defined using C calling conventions. The following sample code segments show the use and definition of a simple UDF to calculate an exponent:

Program 1 declares the function to DB2. This program must be compiled and executed before program 2 can be compiled.

```
exec sql
   create function mfexp(integer, integer)
      returns integer
      fenced
      external name ’db2v2fun!mfexp’
      not variant
      no sql
      parameter style db2sql
      language cobol
      no external action
end-exec
```

Note the LANGUAGE COBOL clause. This is provided by Micro Focus COBOL as an extension to the DB2 syntax. It is equivalent to LANGUAGE C and, regardless of which is used, the called module should conform to the C calling convention. The EXTERNAL NAME clause specifies, in this case, that the called module is called db2v2fun (.dll or .dlw dependent on platform) and the entry point within this is mfexp.

Program 2 uses the UDF:

```
     move 2 to hv-integer
     move 3 to hv-integer-2
     exec sql
         values (mfexp(:hv-integer, :hv-integer-2))
          into :hv-integer-3
     end-exec
```

Program 3 is a pure COBOL program containing the UDF itself.

```
$set case
 special-names.
     call-convention 0 is cc.
 linkage section.
 01  a pic s9(9) comp-5.
 01  b pic s9(9) comp-5.
 01  c pic s9(9) comp-5.
 01  an pic s9(4) comp-5.
 01  bn pic s9(4) comp-5.
 01  cn pic s9(4) comp-5.
 01  udf-sqlstate pic x(6).
 01  udf-fname pic x(28).
 01  udf-fspecname pic x(19).
 01  udf-msgtext pic x(71).
 procedure division cc.
    goback
    .
 entry "mfexp" cc
     using a b c an bn cn
           udf-sqlstate
           udf-fname
           udf-fspecname
           udf-msgtext.
     if an not = 0 or bn not = 0
         move -1 to cn
     else
         compute c = a ** b
         move 0 to cn
     end-if
     goback
     .
```

This module should be compiled to create a dynamically loadable executable (dll) and placed somewhere where the operating system can locate it (on the PATH).

Note: Entry-point names are case sensitive on all systems. Care should be exercised in matching case names, and the CASE Compiler directive should be specified (as per the $SET statement in the example program above).

9.4 Extensions to Embedded SQL Support

This section discusses Micro Focus extensions to the embedded SQL support.

9.4.1 The INCLUDE Statement

Statements of the form:

```
exec sql
   include filename
end-exec
```

are permitted and are processed in exactly the same way as the statement:

```
copy filename
```

The included file can contain any COBOL statements that a copyfile can, including further EXEC SQL statements.

UNIX On AIX, the filename is converted to lower case for the special case of sqlca or sqlda, no matter how it is specified.

9.4.2 The DECLARE TABLE Statement

Statements of the form:

```
exec sql
   DECLARE table-name TABLE
   ...
end-exec
```

are permitted and are treated as comments.

9.4.3 Integer Host Variables

The embedded SQL support requires the format of integers to be USAGE COMP-5. For your convenience, the DB2 ECM also allows host variables to use USAGE COMP, COMP-4 and BINARY and generates additional code to convert the format. The most efficient code is generated when COMP-5 is used.

9.4.4 Qualified Host Variables

Host variables can be qualified using DB2 for MVS compatible syntax.

For example, suppose you have defined some host variables as follows:

```
01 block-1.
   03 hostvar pic s9(4) comp-5.
01 block-2.
   03 hostvar pic s9(4) comp-5.
```

You can qualify which instance of hostvar to use with syntax of the form:

```
exec sql
   fetch s2 into :block-1.hostvar
end-exec
```

9.4.5 Host Variable Groups and Indicator Arrays

When host variables are declared in a group item, an SQL statement which needs to refer to each of these variables in turn can be abbreviated by referring instead to the group-name. If you need to associate indicator variables with these host variables, define a table of indicator variables with as many instances as there are host variables, and reference this table (the item with the OCCURS clause, not a group item containing it).

For example, suppose you have defined some host variables as follows:

```
01  host-structure.
   03 sumh           pic s9(9) comp-5.
   03 avgh           pic s9(9) comp-5.
   03 minh           pic s9(9) comp-5.
   03 maxh           pic s9(9) comp-5.
   03 varchar.
      49 varchar-l   pic s9(4) comp.
      49 varchar-d   pic x(1000).
 01  indicator-table.
     03 indic          pic s9(4) comp-5 occurs 4.
 01  redefines indicator-table.
     03 indic1         pic s9(4) comp-5.
     03 indic2         pic s9(4) comp-5.
     03 indic3         pic s9(4) comp-5.
     03 indic4         pic s9(4) comp-5.
```

In such an example, the procedural statement:

```
exec sql fetch s3 into
  :host-structure:indic
end-exec
```

is equivalent to:

```
exec sql fetch s3 into
  :sumh:indic1, :avgh:indic2, :minh:indic3,
  :maxh:indic4, :varchar
end-exec
```

The four declared indicator variables are allocated to the first four host variables. If five or more had been declared, all five host variables would have an associated indicator variable.

The table of indicator variables is redefined only to show the equivalent SQL statement (subscripting is not allowed in SQL statements). The redefinition can be omitted and the COBOL program can refer to the indicator variables using subscripting, if desired.

9.4.6 The NOT Operator

DB2 allows the operators ¬=, ¬> and ¬<. These are mapped to <>, <= and >=. The character representation of the NOT operator varies from system to system, so you can define it using the NOT option of the DB2 Compiler directive.

9.4.7 The Concat Operator (|)

In some countries the symbol used for the concat operator is not the ASCII character (|). The DB2 ECM enables you to specify a different ASCII character for the concat operator via the CONCAT option of the DB2 Compiler directive.

9.4.8 SQL Communications Area

After any SQL statement has executed, important information is returned to the program in an area called the SQL Communications Area (SQLCA). The SQL Communications Area is usually included in your program using the statement:

```
exec sql include sqlca end-exec
```

This causes the source file **sqlca.cpy** (on Windows) or **sqlca.cbl** (on UNIX) to be included in your source code. This source file, supplied with the DB2 ECM, contains a COBOL definition of the SQLCA.

If you do not include this statement, the DB2 ECM automatically allocates an area, but this area is not addressable in your program. However, if you declare either or both of SQLCODE and SQLSTATE, the DB2 ECM generates code to copy the corresponding fields in the SQLCA area to the user-defined fields after each EXEC SQL statement. We recommend you define the entire SQLCA (this facility is provided for ANSI compatibility).

After any non-zero condition in SQLCODE, the DB2 ECM updates the contents of the MFSQLMESSAGETEXT data item with a description of the exception condition, provided it has been defined. If it is, it must be declared as a character data item (PIC X(n), where n can be any legal value; if the message does not fit into the data item it is truncated).

None of SQLCA, SQLCODE, SQLSTATE and MFSQLMESSAGETEXT is required to be declared as host variables.

9.4.9 Support for Object Oriented COBOL Syntax

The DB2 ECM has been enhanced to work with Object Oriented COBOL syntax (OO programs). There are, however, a couple of restrictions that you should be aware of:

- The INIT option of the DB2 Compiler directive is disabled if it is used within an OO program. This means that if you want the functionality of the DB2(INIT=PROT) Compiler directive, you will need to include a non-OO module and compile it with this directive.
- If you use an EXEC SQL WHENEVER statement within a METHOD, any additional METHODs coded in the same CLASS that have SQL statements in them need to have the section that is referenced in the preceding WHENEVER statement defined. Not doing this results in a compilation error indicating that the section has not been defined. You can get around this restriction by defining another EXEC SQL WHENEVER statement.

9.4.10 Support for Nested COBOL programs

The DB2 ECM allows you to work with nested COBOL programs.

By default, DB2 interface code is generated for every nested COBOL program. To avoid generating DB2 interface code for each nested program, use the DB2 directive IGNORE-NESTED. To use the IGNORE-

NESTED directive properly, there is one restriction that you should be aware of:

- You must specify the PROGRAM-ID statement in the program for which you wish DB2 interface code to be generated. Otherwise, a compile error occurs.

9.5 DB2 INIT Directive

In previous versions of Micro Focus products, Micro Focus included a SQLINIT or SQLINI2 module to perform the CONNECT function. These routines are no longer provided as IBM supports the EXEC SQL CONNECT statement which provides more options than available with the SQLINIT modules.

The INIT directive has the additional option of ensuring that the database connection is correctly closed down even if the application is abnormally terminated, to avoid possible database corruption. If the application is abnormally terminated, all changes since the last COMMIT are rolled back. This database protection can be selected by specifying the option INIT=PROT on the DB2 Compiler directive.

The INIT=PROT option must only be set once for an application. SQL programs called by other SQL programs should not have the INIT=PROT option set. Alternatively, you can specify the INIT=PROT option for the first SQL program to be executed in a run unit. Compiling more than one module in an application with the INIT=PROT option may cause your program to terminate abnormally.

9.6 Compiling

Compiling your SQL program with the COBOL Compiler is logically equivalent to two steps: precompiling to change SQL lines into host language statements, and then compiling the resulting source. These two steps actually occur in a single process, which is performed by the COBOL Compiler in conjunction with the DB2 ECM.

Before you can compile a SQL program, you must have been granted authorization. This is usually done by the DB2 Database Administrator. You must have one of the following:

- sysadm or dbadm authority
- BINDADD privilege if a package does not exist, and one of the following:
    - IMPLICIT_SCHEMA authority on database if the schema name of the package does not exist
    - CREATIN privilege on the schema if the schema name of the package exists
    - ALTERIN privilege on the schema if the package exists
    - BIND privilege on the package if it exists

The user also needs all table privileges required to compile any static SQL statement in the application. Note that privileges granted to groups are not used for authorization checking of static SQL statements. If a program fails to compile because of lack of authority on an SQL object, please contact your company's DB2 Database Administrator.

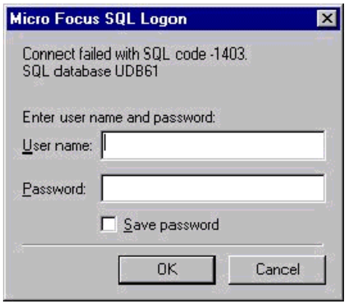
You use the DB2 Compiler directive to give the DB2 ECM information such as the fact that you are using SQL, and which database you are using. See the section *DB2 Compiler Directive* below.

Normally, programs containing embedded SQL are compiled in the same way as non-SQL programs, except that the DB2 Compiler directive is required. Special action is required only when creating an executable (binary) file when additional modules need to be linked in. Programs containing SQL code can be animated like any other program. You can examine host variables inside SQL statements as they are regular COBOL data items.

9.6.1 Compiling Programs that use a Remote DB2 Server

To compile a program that uses a remote DB2 server, you must first connect to that remote server. The DB2 ECM first attempts to connect to the database using the default values for the client workstation you logged on with. If the logon fails, the DB2 ECM will invoke the Micro Focus SQL Logon dialog in which you can then enter a logon ID and password for the database you are trying to compile your program against. The dialog box is shown below.

Figure 9-1. Micro Focus SQL Logon Dialog Box.



Micro Focus SQL Logon

Connect failed with SQL code -1403.
SQL database UDB61

Enter user name and password:

User name:

Password:

□ Save password

There is an option to save your logon ID and password in memory so that you do not need to be prompted the next time you try to compile a program using the same database. This information goes away the next time you re-boot your client machine or if you type the following command from a Net Express command prompt:

```
MFDAEMON CLOSE
```

9.6.1.1 Automated Compiles

Having the graphical logon dialog appear might not be acceptable for automating compiles from a background process such as a command file. There is a way to supply the logon information by setting an environment variable and pointing the variable at a text file that contains the logon ID and password. To do this, set the environment variable **SQLPASS.TXT** to the name of the text file that contains the logon ID and password. For example:

```
SET SQLPASS.TXT=D:\BATCH.TXT
```

Then in the file **batch.txt**, specify the logon ID and password in the format **id.password**. For example:

```
MyId.Mypassword
```

If the security system used to validate your logon ID and password is case sensitive, you need to specify **id.password** in the correct case in this text file.

Note: Specifying the logon and password in a text file does raise security concerns, so care should be used when implementing this facility.

9.6.2 DB2 Compiler Directive

You can specify options for the DB2 Compiler directive using the $SET statement in your program.

For example:

```
$SET DB2(INIT=PROT BIND COLLECTION=MYSCHEMA)
```

Compiler directives have a default value which is used if no other value is specified. This also applies to all existing DB2 directive options. Many of the options are passed straight to DB2 at compilation time and the Compiler default is used when no other value is specified. In these cases, however, the suitability of, and default values for these options is dependent on the DB2 configuration, notably whether it is connected to a DRDA server via DDCS. Because of this, the default Compiler setting of these options is "not set". This means that no value is passed to DB2 and the default value (if applicable) is determined by DB2 itself. Consult your IBM DB2 reference documentation for these values.

The table below lists the DB2 Compiler directive options. The default value is highlighted and underlined.

Option	Description	
ACCESS=*package name*,ACCESS, NOACCESS	Specifies the name of the package to be created and stored in the database. If ACCESS is specified without a parameter, the package name defaults to the program name (without the .CBL extension). Synonym is **PACKAGE**.	
ACTION={ADD | REPLACE }, NOACTION	ACTION indicates whether the package can be added or replaced. This DRDA precompile/bind option is not supported by DB2.	
	ADD	Indicates that the named package does not exist, and that a new package is to be created. If the package already exists, execution stops, and a diagnostic error message is returned.
	REPLACE	Indicates that the old package is to be replaced by a new one with the same location, collection, and package name.
BIND=*bindfile*, BIND, NOBIND	Specifies the name of the bind file to be created. When BIND is specified without a parameter, the bind file defaults to the program name with the filename extension replaced by .BND. Synonym is **BINDFILE**.	
BLOCK={UNAMBIG | ALL | NO}	Specifies the record blocking mode to be used on package creation. For information about row blocking, see the ***IBM DB2 Administration Guide*** or the ***Application Programming Guide***.	
	ALL	Specifies blocking for read-only cursors or cursors not specified as FOR UPDATE OF. Ambiguous cursors are treated as read-only.
	NO	Specifies no blocking of any cursors. Ambiguous cursors are treated as updateable.
	UNAMBIG	Specifies blocking for read-only cursors or cursors not specified as FOR UPDATE OF. Ambiguous cursors are treated as updateable.
	Synonym is **BLOCKING**.	

<table>
<tr><th>Option</th><th colspan="2">Description</th></tr>
<tr><td>CCSIDG=double-ccsid , NOCCSIDG</td><td colspan="2">An integer specifying the coded character set identifier (CCSID) to be used for double byte characters in character column definitions (without a specific CCSID clause) in CREATE and ALTER TABLE SQL statements. This DRDA precompile/bind option is not supported by DB2. The DRDA server will use a system defined default value if this option is not specified.</td></tr>
<tr><td>CCSIDM=mixed-ccsid , NOCCSIDM</td><td colspan="2">An integer specifying the coded character set identifier (CCSID) to be used for mixed byte characters in character column definitions (without a specific CCSID clause) in CREATE and ALTER TABLE SQL statements. This DRDA precompile/bind option is not supported by DB2. The DRDA server will use a system defined default value if this option is not specified.</td></tr>
<tr><td>CCSIDS=sbcs-ccsid, NOCCSIDS</td><td colspan="2">An integer specifying the coded character set identifier (CCSID) to be used for single byte characters in character column definitions (without a specific CCSID clause) in CREATE and ALTER TABLE SQL statements. This DRDA precompile/bind option is not supported by DB2. The DRDA server will use a system defined default value if this option is not specified.</td></tr>
<tr><td rowspan="5">CHARSUB={DEFAULT | BIT | SBCS | MIXED}, NOCHARSUB</td><td colspan="2">Designates the default character sub-type that is to be used for column definitions in CREATE and ALTER TABLE SQL statements. This DRDA precompile/bind option is not supported by DB2.</td></tr>
<tr><td><b>BIT</b></td><td>Use the FOR BIT DATA SQL character sub-type in all new character columns for which an explicit sub-type is not specified.</td></tr>
<tr><td><b>DEFAULT</b></td><td>Use the target system defined default in all new character columns for which an explicit sub-type is not specified.</td></tr>
<tr><td><b>MIXED</b></td><td>Use the FOR MIXED DATA SQL character sub-type in all new character columns for which an explicit sub-type is not specified.</td></tr>
<tr><td><b>SBCS</b></td><td>Use the FOR SBCS DATA SQL character sub-type in all new character columns for which an explicit sub-type is not specified.</td></tr>
<tr><td>COLLECTION=schema name, NOCOLLECTION</td><td colspan="2">Specifies a 30-character collection identifier for the package. If this is not specified, the authorization identifier for the user processing the package is used.</td></tr>
</table>

Option	Description	
COMMIT={1 | 2 | 3 | 4}	Specifies where implicit COMMIT statements should be generated.	
	1	No COMMIT statements implicitly generated
	2	COMMIT statements are implicitly generated on STOP RUN statements and at the end of the program
	3	COMMIT statements are implicitly generated on STOP RUN and EXIT PROGRAM statements and at the end of the program
	4	COMMIT statements are implicitly generated after every SQL statement
CONCAT=(ascii character code | 33 }	Specifies the ASCII character code to use for the CONCAT symbol (|).	
CONNECT={1 | 2}, NOCONNECT	Specifies that a CONNECT statement is to be processed as either a type 1 CONNECT or a type 2 CONNECT.	
CTRACE, NOCTRACE	Creates a trace file for submission to technical support if requested. The filename of the file that is created is **sqltrace.txt**.	
DB=*database name*, DB	Specifies the name of the database that the program accesses. If DB is specified without a parameter, the database specified in the environment variable **DB2DBDFT** is used.	
DEC={31 | 15} , NODEC	Specifies the maximum precision to be used in decimal arithmetic operations. This DRDA precompile/bind option is not supported by DB2. The DRDA server will use a system defined default value if this option is not specified. Use **15** to specify 15-digit precision is used in decimal arithmetic operations. Use **31** to specify 31-digit precision is used in decimal arithmetic operations.	
DECDEL={PERIOD | COMMA}, NODECDEL	Designates whether a period (.) or a comma (,) will be used as the decimal point indicator in decimal and floating point literals. This DRDA precompile/bind option is not supported by DB2. The DRDA server will use a system defined default value if this option is not specified.	
	COMMA	Use a comma (,) as the decimal point indicator.
	PERIOD	Use a period (.) as the decimal point indicator.

Option	Description	
DEFERRED_PREPARE={NO | YES | ALL}, NODEFERRED_PREPARE	Provides a performance enhancement when accessing DB2 common server databases or DRDA databases. This option combines the SQL PREPARE statement flow with the associated OPEN, DESCRIBE, or EXECUTE statement flow to minimize inter-process or network flow.	
	NO	The PREPARE statement will be executed at the time it is issued.
	YES	Execution of the PREPARE statement will be deferred until the corresponding OPEN, DESCRIBE, or EXECUTE statement is issued. The PREPARE statement will not be deferred if it uses the INTO clause, which requires an SQLDA to be returned immediately. However, if the PREPARE INTO statement is issued for a cursor that does not use any parameter markers, the processing will be optimized by pre-OPENing the cursor when the PREPARE is executed.
	ALL	Same as YES, except that a PREPARE INTO statement which contains parameter markers is deferred. If a PREPARE INTO statement does not contain parameter markers, pre-OPENing of the cursor will still be performed. If the PREPARE statement uses the INTO clause to return an SQLDA, the application must not reference the content of this SQLDA until the OPEN, DESCRIBE, or EXECUTE statement is issued and returned.
DEGREE={1 | degree-of-parallelism | ANY}, NODEGREE	Specifies whether or not the query is to be executed using I/O parallel processing.	
	1	Prohibits parallel I/O operations
	degree-of-I/O-parallelism	Specifies the degree of parallel I/O operations, a value between 2 and 32767 (inclusive)
	ANY	Allows parallel I/O operations.

Option	Description	
DISCONNECT={EXPLICIT | CONDITIONAL | AUTOMATIC}, NODISCONNECT	**AUTOMATIC**	Specifies that all database connections are to be disconnected at commit.
	CONDITIONAL	Specifies that the database connections that have been marked RELEASE or have no open WITH HOLD cursors are to be disconnected at commit.
	EXPLICIT	Specifies that only database connections that have been explicitly marked for release by the RELEASE statement are to be disconnected at commit.
DYNAMICRULES={BIND | RUN | DEFINE | INVOKE}, NODYNAMICRULES	Specifies which authorization identifier to use when dynamic SQL in a package is executed.	
	BIND	Indicates that the authorization identifier used for the execution of dynamic SQL is the package owner.
	RUN	Indicates that the authorization identifier used for the execution of dynamic SQL is the authid of the person executing the package.
	DEFINE	Indicates that the authorization identifier used for execution of dynamic SQL is the definer of the UDF or stored procedure. This option is not supported by DB2.
	INVOKE	Indicates that the authorization identifier used for the execution of dynamic SQL is the invoker of the UDF or stored procedure. This option is not supported by DB2.

Option	Description	
EXPLAIN={NO | YES | ALL}, NOEXPLAIN	Stores information in the Explain tables about the access plans chosen for each SQL statement in the package. DRDA does not support the ALL value for this option.	
	NO	Explain information will not be captured.
	YES	Explain tables will be populated with information about the chosen access plan.
	ALL	Explain information for each eligible static SQL statement will be placed in the Explain tables. In addition, Explain information will be gathered for eligible dynamic SQL statements at run time, even if the CURRENT EXPLAIN SNAPSHOT register is set to NO. For more information about special registers, see the ***IBM DB2 SQL Reference***.
EXPLSNAP={NO | YES | ALL}, NOEXPLSNAP	Stores Explain Snapshot information in the Explain tables. This DB2 precompile/bind option is not supported by DRDA.	
	NO	An Explain Snapshot will not be captured.
	YES	An Explain Snapshot for each eligible static SQL statement will be placed in the Explain tables.
	ALL	An Explain Snapshot for each eligible static SQL statement will be placed in the Explain tables. In addition, Explain Snapshot information will be gathered for eligible dynamic SQL statements at run time, even if the CURRENT EXPLAIN SNAPSHOT register is set to NO. For more information about special registers, see the ***IBM DB2 SQL Reference***.
FEDERATED={NO|YES}	Specifies whether a static SQL statement references a nickname or a federated view. SQL errors are returned if the package does not refer to a federated view or nickname and this option is specified, or if the package does refer to a federated view or nickname and the option is not specified.	
	NO	The program will connect to a DB2 Universal Database. This is the default value.
	YES	The program will access a DB2 federated system.

Option	Description	
FORMAT={DEF | USA | EUR | ISO | JIS | LOC}	Specifies the date and time format when date/time fields are assigned to string representations in host variables. DEF is a date and time format associated with the country code of the database.	
	EUR	is the IBM standard for European date and time format.
	ISO	is the date and time format of the International Standards Organization.
	JIS	is the date and time format of the Japanese Industrial Standard.
	LOC	is the date and time format in local form associated with the country code of the database.
	USA	is the IBM standard for U.S. date and time format.
	Synonym is **DATETIME**.	
FUNCPATH=schema-name , NOFUNCPATH	Specifies the function path to be used in resolving user-defined distinct types and functions in static SQL. If this option is not specified, the default function path is: `"SYSIBM","SYS FUN",USER` where USER is the value of the USER special register. This DB2 precompile/bind option is not supported by DRDA.	
	schema-name	is a short SQL identifier, either ordinary or delimited, which identifies a schema that exists at the application server. No validation that the schema exists is made at precompile or at bind time. The same schema cannot appear more than once in the function path. The number of schemas that can be specified is limited by the length of the resulting function path, which cannot exceed 254 bytes. The schema SYSIBM does not need to be explicitly specified; it is implicitly assumed to be the first schema if it is not included in the function path. For more information, see the ***IBM DB2 SQL Reference***.
GENERIC=string	Provides a means of passing new bind options to a target DRDA database. Each option must be separated by one or more spaces and enclosed in double quotes. For example: DB2(GENERIC="keepdynamic yes")	

<table>
<tr><th>Option</th><th colspan="2">Description</th></tr>
<tr><td>IGNORE-NESTED=<i>program-id</i>,
IGNORE-NESTED,
NOIGNORE-NESTED</td><td colspan="2">In nested programs, specifies the program-id at which to start generating DB2 interface code. Any nested program encountered before the program-id is ignored and no DB2 interface code is generated. You must specify a program-id in the COBOL source code; otherwise, a compile error results. If you specify IGNORE-NESTED without a parameter, the program-id defaults to the program name with the filename extension replaced by .CBL.</td></tr>
<tr><td rowspan="2">INIT={PROT }, NOINIT</td><td colspan="2">Makes the program initialize SQL. This option is disabled if it is used within an OO program.</td></tr>
<tr><td><b>PROT</b></td><td>For SQL programs that need to protect the database on STOP RUN but do not want to initialize.</td></tr>
<tr><td rowspan="3">INSERT={DEF | BUF},
NOINSERT</td><td colspan="2">Allows a program being precompiled or bound from a DB2 V2.1 client to a DATABASE 2 Parallel Edition server to request that data inserts be buffered to increase performance.</td></tr>
<tr><td><b>BUF</b></td><td>Specifies that inserts from an application should be buffered.</td></tr>
<tr><td><b>DEF</b></td><td>Specifies that inserts from an application should not be buffered.</td></tr>
<tr><td rowspan="6">ISOLATION={CS | RR | UR | RS | NC}</td><td colspan="2">Determines how far a program bound to this package can be isolated from the effect of other executing programs. For more information about isolation levels, see the <b><i>IBM DB2 SQL Reference</i></b>.</td></tr>
<tr><td><b>CS</b></td><td>specifies Cursor Stability as the isolation level.</td></tr>
<tr><td><b>NC</b></td><td>(No Commit) specifies that commitment control is not to be used. This isolation level is not supported by DB2.</td></tr>
<tr><td><b>RR</b></td><td>specifies Repeatable Read as the isolation level.</td></tr>
<tr><td><b>RS</b></td><td>specifies Read Stability as the isolation level. Read Stability ensures that the execution of SQL statements in the package is isolated from other application processes for rows read and changed by the application.</td></tr>
<tr><td><b>UR</b></td><td>specifies Uncommitted Read as the isolation level.</td></tr>
</table>

Option	Description	
LANGLEVEL={SAA1 | NONE | MIA|SQL92E}	For more information about this option, see the ***IBM DB2 Application Programming Guide***.	
	MIA	The FOR UPDATE clause is optional for positioned updates. C null-terminated strings are padded with blank characters, and always include the null-terminating character. This option is not supported by DB2 CONNECT.
	SAA1	Requires the FOR UPDATE clause for all columns that are updated in a positioned update. C null-terminated strings are not padded with blank characters, and do not include a null-terminating character if truncation occurs.
	NONE	Synonym for SAA1.
	SQL92E	Similar to MIA. See the manual for the differences.
	Synonym is **STDLVL**.	
LEVEL=consistency-token, NOLEVEL	Defines the level of a module using the consistency token. The consistency token is any alphanumeric value up to 8 characters in length. The RDB package consistency token verifies that the requester's application and the relational database package are synchronized. This DRDA precompile option is not supported by DB2. Note: This option is not recommended for general use.	
MSGAREA={data-item-name | MFSQLMESSAGETEXT },NOMSGAREA)	Specifies the name of an alphanumeric data item. If this item is present in the program source it will automatically contain a description of a DB2 error condition (when SQLCODE is non zero).	
NOT={ascii character code | 170 }	Specifies the ASCII character code to use for NOT character (¬).	
OWNER=authorization-id, NOOWNER	Designates a 30-character authorization identifier for the package owner. The owner must have the privileges required to execute the SQL statements in the package. The default is the primary authorization ID of the precompile/bind process if this option has not been explicitly specified. Synonym is **SCHEMA**.	

<table>
<tr><th>Option</th><th colspan="2">Description</th></tr>
<tr><td rowspan="3">QUALFIX, NOQUALFIX</td><td colspan="2">Causes the DB2 ECM to append three characters to the name of the host variables when declaring them to DB2. This ensures problems caused by qualification (where two or more host variables have identical names when not qualified) are avoided but has the side-effect that</td></tr>
<tr><td>(1)</td><td>host variable names have a maximum length of 27 characters unless you are using DB2 Universal Database 5.0 or later.</td></tr>
<tr><td>(2)</td><td>DB2 error messages will sometimes display the names of host variables with the three additional characters appended to them.</td></tr>
<tr><td>QUALIFIER=qualifier-name, NOQUALIFIER</td><td colspan="2">Provides a 30-character implicit qualifier for unqualified table names, views, indexes, and aliases contained in the package. The default is the owner's authorization ID.</td></tr>
<tr><td>QUERYOPT= optimization-level, NOQUERYOPT</td><td colspan="2">Indicates the desired level of optimization for all static SQL statements contained in the package. The default value is 5. For the complete range of optimization levels available, see the SET CURRENT QUERY OPTIMIZATION statement in the <b><i>SQL Reference</i></b>. This DB2 precompile/bind option is not supported by DRDA.</td></tr>
<tr><td rowspan="3">RELEASE={COMMIT | DEALLOCATE}, NORELEASE</td><td colspan="2">Indicates whether resources are released at each COMMIT point, or when the application terminates. This DRDA precompile/bind option is not supported by DB2.</td></tr>
<tr><td><b>COMMIT</b></td><td>Release resources at each COMMIT point. Used for dynamic SQL statements.</td></tr>
<tr><td><b>DEALLOCATE</b></td><td>Release resources only when the application terminates.</td></tr>
<tr><td>REPLVER=version-id, NOREPLVER</td><td colspan="2">Replaces a specific version of a package. The version identifier specifies which version of the package is to be replaced. Maximum length is 254 characters.</td></tr>
</table>

Option	Description	
RETAIN={YES | NO} , NORETAIN	RETAIN indicates whether EXECUTE authorities are to be preserved when a package is replaced. If ownership of the package changes, the new owner grants the BIND and EXECUTE authority to the previous package owner.	
	NO	does not preserve EXECUTE authorities when a package is replaced.
	YES	preserves EXECUTE authorities when a package is replaced.
SQLERROR={NOPACKAGE | CHECK | CONTINUE}, NOSQLERROR	Indicates whether to create a package or a bind file if an error is encountered.	
	CHECK	Specifies that the target system performs all syntax and semantic checks on the SQL statements being bound. A package will not be created as part of this process. If, while creating a package, an existing package with the same name and version is encountered, the existing package is neither dropped nor replaced if action replace was specified.
	CONTINUE	A package or a bind file is created even when SQL errors are encountered. This option is not supported by DB2.
	NOPACKAGE	A package or a bind file is not created if an error is encountered.
	If syntax is used together with the package option, package is ignored. Synonym is **ERROR**.	

Option	Description	
SQLFLAG={MVSDB2V23| MVSDB2V31 | MVSDB2V41|SQL92E}- SYNTAX, NOSQLFLAG	Identifies and reports on deviations from the SQL language syntax specified. A bind file or a package is created only if the bindfile or the package option is specified in addition to the sqlflag option. Local syntax checking is performed only if one of the following options is specified: bindfile, package, sqlerror check, syntax. If sqlflag is not specified, the flagger function is not invoked, and the bind file or the package is not affected.	
	MVSDB2V23-SYNTAX	The SQL statements will be checked against MVS DB2 Version 2.3 SQL language syntax. Any deviation from the syntax is reported in the precompiler listing.
	MVSDB2V31-SYNTAX	The SQL statements will be checked against MVS DB2 Version 3.1 SQL language syntax. Any deviation from the syntax is reported in the precompiler listing.
	MVSDB2V41-SYNTAX	The SQL statements will be checked against MVS DB2 Version 4.1 SQL language syntax. Any deviation from the syntax is reported in the precompiler listing.
	SQL92E-SYNTAX	The SQL statements will be checked against ANSI or ISO SQL92 SQL language syntax. Any deviation from the syntax is reported in the compiler listing.
	Synonym is **FLAG**.	
SQLRULES={DB2 | STD}, NOSQLRULES	Specifies whether type 2 CONNECTs are to be processed according to the DB2 rules or the Standard (STD) rules based on ISO/ANS SQL92.	
	DB2	Allow the use of the SQL CONNECT statement to switch the current connection to another established (dormant) connection.
	STD	Allow the use of the SQL CONNECT statement to establish a new connection only. The SQL SET CONNECTION statement must be used to switch to a dormant connection.
	Synonym is **RULES**.	

Option	Description	
SQLWARN={YES | NO}, NOSQLWARN	Indicates whether warnings will be returned from the compilation of dynamic SQL statements (via PREPARE or EXECUTE IMMEDIATE), or from describe processing (via PREPARE...INTO or DESCRIBE). This DB2 precompile/bind option is not supported by DRDA.	
	NO	Warnings will not be returned from the SQL compiler.
	YES	Warnings will be returned from the SQL compiler.
	Note: SQLCODE +238 is an exception. It is returned regardless of the sqlwarn option value. Synonym is **WARN**.	
STRDEL={APOSTROPHE | QUOTE}, NOSTRDEL	Designates whether an apostrophe (’) or double quotation marks (") will be used as the string delimiter within SQL statements. This DRDA precompile/bind option is not supported by DB2. The DRDA server will use a system defined default value if this option is not specified. Specify **APOSTROPHE** to use an apostrophe (’) as the string delimiter. Specify **QUOTE** to use double quotation marks (") as the string delimiter.	
SYNCPOINT={ONEPHASE | TWOPHASE | NONE}, NOSYNCPOINT	Specifies how commits or rollbacks are to be coordinated among multiple database connections.	
	NONE	Specifies that no Transaction Manager (TM) is to be used to perform a two-phase commit, and does not enforce single updater, multiple reader. A COMMIT is sent to each participating database. The application is responsible for recovery if any of the commits fail.
	ONEPHASE	Specifies that no TM is to be used to perform a two-phase commit. A one-phase commit is to be used to commit the work done by each database in multiple database transactions.
	TWOPHASE	Specifies that the TM is required to coordinate two-phase commits among those databases that support this protocol.

<table>
<tr><th>Option</th><th colspan="2">Description</th></tr>
<tr><td>SYNTAX</td><td colspan="2">A synonym for directive SQLERROR=CHECK.</td></tr>
<tr><td>TEXT=label, NOTEXT</td><td colspan="2">The description of a package. Maximum length is 255 characters. The default value is blanks. This DRDA precompile/bind option is not supported by DB2.</td></tr>
<tr><td>TRANSFORM-GROUP=identifier.
NOTRANSFORM-GROUP</td><td colspan="2">Specifies the transform group name to be used for static SQL. This is a SQL identifier up to 18 characters in length. This DRDA precompile/bind option is not supported by DB2.</td></tr>
<tr><td rowspan="3">VALIDATE={RUN | BIND},
NOVALIDATE</td><td colspan="2">Determines when the database manager checks for authorization errors and object not found errors. The package owner authorization ID is used for validity checking.</td></tr>
<tr><td><b>BIND</b></td><td>Validation is performed at precompile/bind time. If all objects do not exist, or all authority is not held, error messages are produced. If sqlerror continue is specified, a package/bind file is produced despite the error message, but the statements in error are not executable.</td></tr>
<tr><td><b>RUN</b></td><td>Validation is attempted at bind time. If all objects exist, and all authority is held, no further checking is performed at execution time.
If all objects do not exist, or all authority is not held at precompile/bind time, warning messages are produced, and the package is successfully bound, regardless of the sqlerror continue option setting. However, authority checking and existence checking for SQL statements that failed these checks during the precompile/bind process may be redone at execution time.</td></tr>
<tr><td>VERSION=version-id,
NOVERSION</td><td colspan="2">Defines the version identifier for a package. The version identifier is any alphanumeric value, $, #, @, _, -, or ., up to 254 characters in length. This DRDA precompile option is not supported by DB2.</td></tr>
</table>

9.7 Error Codes

Error conditions are returned at compilation time as a number and explanation. Further details of these messages are given in the documentation supplied with your database system. Messages referencing host variables show slightly modified names: hyphens are shown as underscores (_), and there are up to three additional characters at the end of the name which can be ignored. These changes are side effects of modifications made by the DB2 ECM to the SQL code.

Error conditions at run time are indicated by non-zero values in SQLCODE. Explanatory text is placed in the MFSQLMESSAGETEXT data item if defined; see the section *SQL Communications Area* for further details about this data item.

For example:

```
801-S
**    External Compiler Module message
**    SQ0100 SQL1032N  No start database manager command was
issued.
**    SQLSTATE=57019
```

9.8 Creating Debug Files

If an error occurs when compiling a program that requires technical support, your support representative might ask you to provide additional debug files to help in determining the cause of the problem. The support representative might ask you to provide one or more of three debug files, to recreate the problem, in addition to source and data files. You might want to specifiy some of these directives to help in your own debugging efforts. The directives are:

Directive	**File created**	**Information within file**
CHKECM(CTRACE)	ecmtrace.txt	This file contains pseudo COBOL code that shows the code generated to replace the EXEC SQL statements. This file is equivalent to output out of the IBM DB2 COBOL precompiler.

Directive	File created	Information within file
CHKECM(TRACE)	ecmtrace.txt	This file contains detailed information as to what information is passed between the DB2 ECM and the Compiler. If an error occurs that generates invalid syntax, this file will be needed to help isolate where the problem occurred.
DB2(CTRACE)	sqltrace.txt	This file contains a detailed list of information passed to IBM Precompiler Services, and the results. This file is very useful if an error might involve a bug in the DB2 system software as well as the DB2 ECM.

9.9 Linking

To link an application:

1. Open the Net Express project and set the **Type of Build** to **Generic Release Build**.
2. Right click on the **.exe** or **.dll** file.
3. Select **Build Settings ...** and then click on the **Link** tab.
4. Set the **Category** to **Advanced**.
5. In the **Link with these LIB's** edit box enter:

   ```
   db2api.lib
   ```

9.10 Binding

If you use the NOACCESS option of the DB2 Compiler directive or intend to execute the application on a machine other than the one it was compiled on, bind the application to a particular database before execution. In this case, you should use the BIND option to create a bind file that can then be used to bind the program to the database using the DB2 BIND command. For details on doing this, see the documentation supplied with your SQL system.

9.11 Publishing your DB2 Applications on UNIX

If you want to publish your DB2 application on UNIX you must:

1 Configure your login environment to set the COBOPT environment variable to the name of the database parameter file:

```
COBOPT=/usr/lpp/db2_vv_rr_mmmm/lib/db2mkrts.args
export COBOPT
```

where the values of *vv*, *rr* and *mmmm* represent the version, release and modification level of the DB2/6000 system installed on your machine.

2 Edit the file named in the COBOPT environment variable by removing the -vdd option from the initial line of the file.

3 From the directory $COBDIR/src/sql enter:

```
mkrts sqlinit.o
cp rts32 $COBDIR
```

Database Access

10 SQL Option for DB2

This chapter explains how you can use Net Express to create and maintain DB2 applications on your PC.

10.1 Overview

With SQL Option, you can compile, debug and run programs that contain Embedded SQL statements from the Net Express IDE without requiring access to a mainframe or LAN-based database system.

SQL Option allows the EBCDIC environment on the PC to be fully compatible with your mainframe's EBCDIC environment. Refer to the section SQL Option NLS Environment for details

SQL Option uses the same technology as the popular XDB database system. An XDB database behaves exactly like a mainframe DB2 database but it runs on the PC. A personal XDB Server is installed on your PC, providing you with a completely self-contained development and test environment for DB2 applications.

If you are developing in a workgrouping environment, your system administrator can configure SQL Option so that you can access one or more shared, LAN-based XDB Servers in addition to your personal XDB Server. This two-tier configuration provides you with several further options for testing your application. For example, you can:

- Maintain a complete copy or a subset of your mainframe test data on a shared XDB Server. Members of your workgroup can test against this directly or import the necessary tables into their personal XDB Server in situations where data needs to be isolated from other users.
- Duplicate test data on the shared XDB Server and workgroup members' personal XDB Servers. The shared XDB Server can then be used for integration testing, with workgroup members performing development and unit tests against their personal XDB Servers.

- Distribute data across a shared XDB Server and personal XDB Servers. For example, larger, more static tables can be left on the shared XDB Server, with smaller, more volatile tables being exported to the PC.

In addition, with appropriate configuration, you can access mainframe DB2 databases seamlessly with SQL Option's connectivity features. With this third tier configured, you have access to the full range of development and test options, giving you complete freedom to devise an effective workflow customized to your environment. For example, you can:

- Use the mainframe DB2 test environment for all testing
- Use the mainframe for integration testing, with workgroup members performing development and unit tests against either their personal XDB Servers or a shared XDB Server
- Distribute test data across all three tiers, keeping very large tables on the mainframe and providing others on a shared XDB Server. Workgroup members can then download specific data to their personal XDB Server where necessary for isolation purposes.

10.2 SQL Option Components

SQL Option for DB2 consists of an embedded SQL precompiler and a number of configuration tools and graphical data utilities:

- XDB Server
- Server Configuration utility
- Server administration options
- SQL Wizard
- Migrate utility
- Execute SQL option
- Declaration Generator utility
- Options utility

- Bind utility
- Gateway Profile utility
- SQL Option Preprocessor

You can access all these components except for the Options utility on the **Tools** menu of the Net Express IDE. Help for SQL Option tools is available from the components or on the IDE **Help** menu. From this menu you can also access Error Messages help and a full SQL Reference.

You can find the Options utility by selecting **SQL** on the **Options** menu.

10.3 XDB Server

XDB Server is the server component that performs all database operations and emulates a DB2 system. A personal XDB Server is installed on your machine when you install SQL Option for DB2. This local copy of XDB Server is the default server. It must be running before you can debug or run client applications against a local XDB database. You can start it manually on the **Tools** menu or you can configure your project to start it automatically (if it isn't already running) when the project is loaded.

If you want to access mainframe DB2 data directly, your system administrator must first install and configure a number of SQL connectivity components. Once this is done, your system administrator can advise you how to connect to mainframe data. Detailed information on the DB2 Link feature is available by clicking **SQL For DB2 Help** > **Link To DB2** on the IDE **Help** menu.

10.3.1 Server Configuration Utility

The Server Configuration utility enables you to change the configuration of your personal XDB Server. For example, you can change the following:

- The XDB Server name
- Whether XDB Server security is enabled or not

- Which protocol is used to access the XDB Server
- Whether other people can use the XDB Server
- The amount of system resources available to XDB Server

10.3.2 Server Administration Options

The **SQL for DB2** option on the Net Express IDE **Tools** menu enables you to control your connection to an XDB Server.

You can use **Start Server** to run your personal XDB Server.

Log On enables you to log on interactively to an XDB Server when client security is switched on. Client security is controlled from the **Security** tab of the Options utility. Client security must be switched on to access an XDB Server that has security switched on.

You would typically use **Log On** if your XDB Server has security switched on and you want to use a number of the SQL Option utilities without having to log on separately for each one. You do not need to log on explicitly if your XDB Server does not have security switched on.

Log Off enables you to undo a previous log on. If, for example, you log on to an XDB Server that has security switched on and then log off, the next time you try to access the XDB Server, you have to enter an AuthID and password to gain access. You do not need to log off explicitly if your XDB Server does not have security switched on.

10.4 SQL Wizard

SQL Wizard is a graphical utility that makes it easy to create, maintain and query XDB and DB2 databases. You can use SQL Wizard to:

- Manage system security and priorities
- Manage locations, tables and queries
- Create and run queries
- Enter data directly into a table

- Import and export data
- Run batch scripts

10.4.1 Managing System Security and Priorities

SQL Option is provided with the security and administration features you would expect of a database system. Depending on how you intend to use it, however, you might not need to enable the security features. For this reason, your personal XDB Server and its client tools and utilities are initally installed with security switched off.

Security is controlled separately at the server and at the client. If you switch security on for an XDB Server, it can only be accessed by clients that also have security switched on. (In this context, a client is one of the XDB configuration tools or graphical data utilities. The client applications that you develop are expected to handle authorization in the usual way by means of CONNECT statements). If you are developing a DB2 application that handles passwords, you may want to work with a test environment that has security switched on to enable user authentication.

Use the **Admin** menu of SQL Wizard to manage system security. You can control security at three levels:

- All users
- Named groups of users
- Individual users

When security is switched off for an XDB Server, all users are effectively superusers because:

- No passwords are required
- All users share a common AuthID, specified on the **Connect** tab of the Options utility
- All users have access to both client and server utilities

Server security status is set using the Server Configuration utility on the IDE **Tools** menu. Once security has been enabled, users must log on with a valid AuthID and, if required, password. The user's actual AuthID replaces the shared one, which means that a user cannot access a database unless they have been granted the appropriate access privileges by the database owner using GRANT and REVOKE statements.

An AuthID that has been assigned superuser status can change user, group and priority settings that affect all databases on an XDB Server. Initially an AuthID called INSTALL is set as the superuser. If you log on with this AuthID, you can create further superusers as required by clicking **Users** on the **Admin** menu and giving each user the appropriate status.

Ordinary users can use the **Admin** menu only to change their password. Ordinary users who want to change access privileges for specific databases and tables that they own should click **New > SQL** on the **File** menu and use the SQL statements GRANT and REVOKE to change access privileges. The AuthID of the creator of a table is deemed to be its owner.

As well as managing system security, a superuser can assign priorities to individual users or groups of users. Priorities control how much processing resource is available to the user or group, depending on criteria set by the superuser.

Note: When you install SQL Option, the default AuthID is set to TUTORIAL. This AuthID has user privileges and is not authorized to access system tables. This means that, if you switch security on for your personal XDB Server, you cannot use this AuthID to log on to the client utilities.

You should log on using the default superuser AuthID, INSTALL. You can then either set the TUTORIAL AuthID to have superuser status or you can grant it suitable authority on system tables, as appropriate. The INSTALL superuser AuthID has no password assigned initially: you should allocate one to it as soon as possible after switching XDB Server security on.

10.4.2 Managing Locations, Tables and Queries

SQL Wizard’s Catalog Browser window provides an easy way to manage locations, tables and queries, including XDB Server system tables (providing your AuthID has suitable authority). You can open it by clicking any entry on the **View** menu of SQL Wizard or by clicking on the toolbar. The Catalog Browser has three tabs: **Locations**, **Table** and **Query**.

If you are a superuser, you can use the **Locations** tab to manage locations. A superuser can create, alter and drop locations, as well as set the currently active location. Other users can only view a list of available locations.

The **Table** tab displays the hierarchy of locations, AuthIDs, tables and columns. From this page you can create, open, alter and drop tables, as well as view table definitions and indexes, primary keys, foreign keys, views, aliases and synonyms. Note that only a superuser can edit system tables.

The **Query** tab displays all stored queries accessible to the current server under the Location, AuthID and Query hierarchy. From this page you can create, open, run and delete queries.

10.4.3 Creating and Running SQL Queries

You can create SQL queries either by entering SQL statements directly into an SQL window (click SQL on the SQL Wizard toolbar), or by using a prompted query in the Query Design window (click on the toolbar). You do not need to understand SQL to create a prompted query. The Query Design window is split into two areas:

- A schema view that shows the tables involved in your query and any relationships between them. This area of the window is called the table display area.

- A query-by-example view, which shows the columns resulting from your query with any filters that you have applied to them. This area of the window is called the query conditions grid.

When you create a prompted query, you first select the table and columns. Then you apply conditions and sort criteria to the columns to achieve the final result set. Further choices are available from the menus, including joins, the creation of computed columns and the use of built-in functions such as minimum, maximum and average.

As you create a prompted query, the equivalent SQL statements are built up by SQL Wizard. To see them, click SQL. You can edit the SQL statements directly but the changes you make do not appear in the Query Design window. To go back to the Query Design window, click ▣. It's a good idea to save your prompted query before you try editing it in the SQL window. You can run your query and see the results at any time by clicking RUN.

10.4.4 Entering Data Directly into a Table

SQL Wizard provides both a spreadsheet-like table view and a form view that enable you to enter and edit data directly. Use the table view to see many records at once and the form view to see one record at a time.

You must have appropriate access privileges to edit tables. Only a superuser can edit system tables. Ordinary users can edit tables created using either their own AuthID or a GroupID to which they belong. They can also edit tables to which they have specifically been given access using the GRANT statement. In all cases, you must select **Allow Editing** on the **Record** menu before editing is enabled.

You can open or create a table at any time from the Catalog Browser. See the section *Managing Locations, Tables and Queries* for more information. The Table view opens automatically to show the result of a query.

10.4.5 Importing and Exporting Data

SQL Option can import and export data in a variety of formats. You can specify import and export settings by clicking **New > Import** or **New > Export** on the **File** menu of SQL Wizard. You can optionally save these settings in an **.imp** or **.exp** file that you can subsequently run as a batch file for routine data conversions.

Use SQL Option's import capabilities to run programs against test data prepared in another file format. For example, you could export suitable test data held in a Microsoft Access database, or a Microsoft Excel or Lotus 1-2-3 spreadsheet, to delimited or fixed-field ASCII files. You could then import these files into XDB Server.

You can import data into an XDB database from any of the following sources:

- Free ASCII files (Delimited)

  Imports data from delimited (free-format) ASCII files.

- Fixed-format ASCII files (Columnar)

  Imports data from columnar (fixed-format) ASCII files.

- dBASE files

  Imports data from dBASE II, III and III+ files. dBASE IV files with the same format as dBASE III files can be imported using the dBASE III option.

- DBMAUI

  Imports data downloaded from IBM's DB2 using IBM's DBMAUI facilities.

- DSNTIAUL

  Imports data downloaded from IBM's DB2 using IBM's DSNTIAUL facilities.

Use SQL Option's export capabilities to transfer data from an XDB database to a mainframe DB2 database if you do not have gateway connectivity, or to create the DDL commands required to create a table.

You can export data from an XDB database into any of the following targets:

- Free ASCII files (Delimited)

  Exports data to delimited (free-format) ASCII files.

- Fixed width ASCII files (Columnar)

  Exports data to columnar (fixed-format) ASCII files.

- dBASE files

  Exports data to dBASE II and dBASE III files.

- DBMAUI

  Exports data to DBMAUI files. DBMAUI files can be read into DB2 systems.

- DSNTIAUL

  Exports data to DSNTIAUL files. DSNTIAUL files can be read into DB2 systems.

- WordPerfect Mail Merge file

  Exports data to WordPerfect 4.2 and 5.0 secondary merge files.

- SQL statements

  Generates DDL commands that can be used to re-create a table, insert data into it and create indexes on the table. COMMENT, WHERE and SYNONYM statements can be included.

10.4.5.1 Import/Export NLS Considerations

You need to be aware of character set conversion issues when importing or exporting data, especially if:

- Your data contains diacritical (accented) or other characters that lie outside the first half of the ASCII character set
- Your data was created using a variety of EBCDIC code pages

If you are importing or exporting EBCDIC data using SQL Wizard, we recommend using the DSNTIAUL data format as this enables you to specify an appropriate EBCDIC to ANSI or ANSI to EBCDIC conversion.

You specify the appropriate conversion by picking an entry in the code page window. For example, suppose you want to export a Swedish EBCDIC table. During the export, you would pick the ANSI to EBCDIC translation **1252 - 278**. Or, to import a Spanish EBCDIC table, you would pick the EBCDIC to ANSI translation **284 - 1252**.

10.4.6 Running Batch Scripts

You can use SQL Wizard to run the following kinds of batch scripts:

- Prompted queries
- SQL scripts, including DDL and DML
- Import files
- Export files

To run a batch script, click **Run Batch** on the **File** menu in SQL Wizard.

Several other SQL Option utilities can create batch files; you can also run these from the appropriate utility. For example:

- Use the Migrate utility to create and run **.mig** files
- Use the Declaration Generator to create and run **.dge** files

10.5 Migrate Utility

If your system administrator has configured the mainframe connectivity facilities of SQL Option, you can use the Migrate utility to import or export data between XDB locations, or between an XDB location and a DB2 subsystem.

The Migrate utility simplifies the process of copying data from one location to another by eliminating intermediate file transfers and the hand-editing of indexes and foreign keys. Using the Migrate utility, you can copy keys, indexes and tables in a single step. You can also copy a subset of a table (selected columns or rows) by extracting data with a SELECT statement.

You can use the Migrate utility’s referential-integrity feature to detect data dependencies automatically and copy tables in the correct order. You can also produce a preliminary "impact analysis" report, which describes the effect of a migration before it is performed. If you copy certain tables regularly, you can save the specification in a **.mig** file. You can then run this file in batch mode, enabling you to perform a migration without respecifying it manually.

For more information about the Migrate utility, click **SQL For DB2 Help > Migrate** on the IDE **Help** menu.

10.6 Execute SQL Option

You can use **Execute SQL** to execute an SQL file that is part of the currently open project. To use it, click on the file in the Project Workspace then click **Execute SQL** on the IDE **Tools** menu. The file must have an **.sql** extension.

10.7 Declaration Generator Utility

In order to access DB2 data from an application program, you need to create host variables that are common data items between the SQL statements in your query and in your program. A data item declaration in the program must have the same name as that used in your SQL statement and must conform to the relevant DB2 data type for the column concerned.

The Declaration Generator utility automates the process of creating copybooks to declare host variables. You specify the table for which you want to generate a copybook and the Declaration Generator creates a copybook with data declarations that match the data-names and types used in the XDB system. The Declaration Generator can be used interactively or in batch mode and can create a separate copybook for each table or one copybook containing declarations for many tables.

To open the Declaration Generator, click **SQL For DB2 > Declaration Generator** on the **Tools** menu.

10.8 Options Utility

Use the Options utility for configuring SQL Option for DB2 and its connectivity with XDB and DB2 database servers. The settings that you can change fall into the following categories:

- Connect

  For configuring the connection between a client application (for example, SQL Wizard or a program that you have developed) and the XDB or DB2 database that it uses.

- Paths

  For specifying where SQL Wizard and other SQL Option files are to be found or written.

- Format

  For specifying formats for dates, times and numeric fields.

- Multi-user

  For specifying options for a multi-user XDB Server in a networked environment. For example, you can control isolation levels and autocommit.

- Browser

  For specifying which database objects are shown in the Catalog Browser window.

- Query

  For specifying default options for working with queries. For example, you can specify whether unqualified table-names can be used and whether Cartesian products are permitted. These options can be overridden for a particular query when you are editing the query.

- Query run

  For specifying options for running queries with SQL Wizard. For example, you can control whether the result is displayed in table or form view and limit the number of rows returned.

- SQL

  For specifying defaults for various SQL operational parameters, including compatibility and sort sequence. For example, you can set DB2 or SQL/DS compatibility and whether to use ASCII or EBCDIC sort sequences. You can also set the escape character and SysAuthID.

- Security

  For specifying security options. You can switch on client security (required if you want to use an XDB Server that has security switched on), as well as controlling the enforcement criteria for user passwords.

The Options utility has a **Summary** page that enables you to obtain a list of all the current configuration settings. You cannot make changes to any of the options from the **Summary** tab but you can print it for reference purposes.

10.9 Bind Utility

The Bind utility enables you to create static SQL packages in a remote DB2 location accessed using DRDA, providing that the target location is registered appropriately at the DB2 Link gateway and that your AuthID has Bind authority at the target location.

The Bind utility creates a static SQL package on the remote system by processing database request modules (DBRM) stored in **.dbr** files that are created by the SQL precompiler. The precompiler creates a separate **.dbr** file for each program, with a separate DBRM entry in the file for each Embedded SQL statement in the program.

You do not need to bind your application if you use the default SQL precompiler settings, as these enable Embedded SQL statements in your programs to be run dynamically against the remote system. Running Embedded SQL dynamically is useful in situations where you want to access remote DB2 data but do not want to keep rebinding your application as it changes; for example, while debugging the application. You can optionally change the precompiler settings to produce static SQL database request modules to bind the application for deployment purposes. For more information on the relevant directives (DBRM, LOCATION, COLLECTION-ID and AUTOBIND), click **SQL For DB2 Help >**

COBOL Precompiler > SQL Option Preprocessor > SQL Option Preprocessor Directives on the IDE **Help** menu.

A package at a location is identified by a collection identifier, a program identifier, a version label and a consistency token. Typically, you would use the collection identifier as a way of grouping the packages used in a single application consisting of one or more programs. The precompiler creates a program identifier for a package from the root of the program filename. You can specify a version label or accept the default value (01). You do not have to specify a version label, because the current package can be identified from its consistency token, generated from a timestamp at the start of precompilation. Note that each time you recompile a DB2 application, the timestamp changes and you must rebind the application to the DB2 host location if you are using static SQL packages. An alternative approach to avoid having to rebind your application each time you compile it is to debug against data stored in an EBCDIC location on your personal XDB Server. You can then test it against mainframe data when you have achieved a satisfactory level of stability.

The Bind utility enables you to specify a number of options for an application, including a version number and the isolation level of the application. These options can be stored in a file for future use.

For more information about the Bind utility, click **SQL For DB2 Help > Bind** on the IDE **Help** menu.

Note: The Bind utility does not appear on the **SQL For DB2** submenu of the IDE **Tools** menu by default but you can add it to a submenu of the **Tools** menu yourself. See *Bind* in the online help index for more information.

10.10 Gateway Profile Utility

Before you can run any application that accesses the mainframe, you must configure the XDB Link using the Gateway Profile utility. This utility:

- Defines connections between your workstation and the mainframe data source
- Configures settings needed to translate data correctly and to control client conversations.

To open the Gateway Profile utility, click **SQL For DB2 > XDB Link** on the **Options** menu. For more information about this utility, click **SQL For DB2 Help > XDB Link** on the IDE **Help** menu.

10.11 SQL Option Preprocessor

The SQL Precompiler allows COBOL programmers to develop applications containing embedded SQL and test them against a full-featured, relational database system.

There are certain requirements your COBOL programs must meet before they can be compiled and debugged. Chapters 1 through 6 of the ***Data Access*** guide and the ***SQL Option Preprocessor User's Guide*** explain what these requirements are.

10.11.1 SQL Communications Area (SQLCA)

Every COBOL program containing embedded SQL must have an SQL Communications Area (SQLCA) or the field SQLCODE defined in its working storage section. This definition is normally accomplished by including the SQLCA copybook provided with Net Express. A complete description of the SQLCA structure is provided in the ***SQL Option SQL Reference***.

10.11.2 SQL Descriptor Area (SQLDA)

COBOL programs that include dynamic SQL must have an SQL Descriptor Area (SQLDA) defined. This definition is usually accomplished by including the SQLDA copybook provided with Net Express. However, the SQLDA copybook included with Net Express may not match the structure you normally use, in which case you may need to create an alternate copybook.

The SQLDA holds information about dynamic SQL queries, and is required for allocating the proper amount of space for the query.

For more information about dynamic SQL see the chapter *Dynamic SQL* . For a complete description of the SQLDA structure, see the ***SQL Option SQL Reference***.

10.11.3 Support for Object Oriented COBOL Syntax

The SQL Option preprocessor has been enhanced to work with Object Oriented COBOL syntax (OO programs). There is, however, one restriction that you should be aware of:

- If you use an EXEC SQL WHENEVER statement within a METHOD, any additional METHODs coded in the same CLASS that have SQL statements in them need to have the section that is referenced in the preceding WHENEVER statement defined. Not doing this results in a compilation error indicating that the section has not been defined. You can get around this restriction by defining another EXEC SQL WHENEVER statement.

10.11.4 Security

SQL Option security measures control access to the system and protect data stored in table objects. If the security option is turned on at both the server and on the client, you will be prompted for a user ID and password when you compile using the **VALIDATE** preprocessor directive or execute a program containing embedded SQL. For more information on setting up security and authority, see the following SQL Option manuals:

- ***Administrator's Guide***
- ***SQL Wizard User's Guide***
- ***Options User's Guide***

10.11.5 DSNTIAR Facility

The SQL Option DSNTIAR facility converts an SQL return code into a character string error message (similar to the IBM DSNTIAR facility on the mainframe). See the ***SQL Option Preprocessor User's Guide*** for details on how to invoke this facility.

10.11.6 Migration Considerations

In this version of Net Express, the SQL Option Preprocessor has changed from using the old type of preprocessor architecture to use a new type of preprocessor interface called an External Compiler Module (ECM) interface, which is more tightly integrated with the Net Express Compiler and Debugger.

The main difference is that the new preprocessor only gets passed statements that begin with EXEC SQL. The old preprocessor had to look at every line of code in a program.

Another difference is the number of preprocessor directives have been reduced because of tighter integration with the compiler and the fact that programs being compiled target mainframe DB2 compatibility by default.

10.11.6.1 Invoking the Preprocessor

If you are migrating from either the Classic Workbench product or Mainframe Express version 1.1, the syntax for invoking the SQL Option Preprocessor has changed from:

```
p($xdbdir\xdb) any additional preprocessor directives end-p
```

to:

```
XDB(any additional preprocessor directives)
```

For example, the following command to compile program TEST1 with preprocessor directives VALIDATE and FILLSYSCAT changes from:

```
Cobol test1 p($xdbdir\xdb) validate fillsyscat endp gnt;
```

to:

```
Cobol test1 xdb(validate fillsyscat) gnt;
```

This change will affect all users who compile programs using batch files.

10.11.6.2 Directives in Comments

With early versions of XDB software, you could also place precompiler directives in comments by coding $$XDB. This is still supported but you need to specify the COMPILER directive **DIRECTIVES-IN-COMMENTS** for them to be picked up by the new SQL Option preprocessor.

10.11.6.3 Ambiguous References

If you define a COBOL variable with the same name more than once, and do not fully qualify the host variable name, the new preprocessor will flag the ambiguous reference with the following message:

```
SQ0408S <variable-name> is non-unique and should be qualified
```

10.11.6.4 Debug Files

The old preprocessor had a directive DEBUG which generated a text file XDB.$$$ that listed the changes to convert EXEC SQL statements to SQL Option CALL statements. This file was very useful when trying to pinpoint problems. This preprocessor directive no longer does this. Instead, the ECM interface uses the following COMPILER and preprocessor directives to produce debug files that you may be asked to produce to aid in resolving problems:

Directive	File created	Function
XDB(CTRACE)	sqltrace.txt	Before/after information about the SQL statement and any tasks the EXEC SQL statement should generate, as well as SQLCA information. Precompiler errors are returned as a positive SQLCODE. SQL Option server errors (when VALIDATE is used) are returned in SQLCODE field unchanged from what server set.
CHKECM(CTRACE)	ecmtrace.txt	COBOL code generated by ECM to replace EXEC SQL calls. Major difference in this file from the old DEBUG file is that you need to cut/paste statements if you want to execute code into a COBOL file and this file only includes EXEC SQL statements plus variables generated by ECM.
CHKECM(TRACE)	ecmtrace.txt	Very detailed file that lists all information that flows from the ECM to/from the Compiler. If the Compiler generates an error message indicating ECM generated the problem, look for literal ECM-FATAL in text file. This will usually pinpoint statement that produced the error.

In the example given earlier, to compile the same program and generate all three of the debug files, the command line entry would be:

```
Cobol test1 xdb(validate fillsyscat ctrace) chkecm(ctrace)
    omf(gnt);
```

10.11.6.5 Maximum number of SQL statements

The new preprocessor is a one pass precompiler. The old preprocessor was also a one pass precompiler, but it could read ahead. This allowed the old preprocessor to define variables for each EXEC SQL statement in WORKING-STORAGE since it knew how many EXEC SQL statements were in the program. The new preprocessor always pre-allocates storage for 750 EXEC SQL statements. If your program gets the following error:

SQ0299s Internal EXEC SQL statement table overflow - use directive MAXSQL to override default.

use the preprocessor directive MAXSQL to increase the maximum number of SQL statements the program can handle.

10.11.6.6 Setting Additional Directives under IDE

When you create a new project or add a new program to an existing project, you can set SQL Option preprocessor directives by clicking **Build Settings** for that program, clicking the **Compile** tab, and then clicking **Advanced**. Select **XDB** from the ESQL Preprocessor drop-down list and then select the directives you want to add from the **Directives** drop-down list. A short description of the directive option is displayed.

10.11.7 XDB Directive

The compiler directive **XDB** has the following options:

AUTHID
AUTOBIND
AUTOCLOSE
COLLECTION-ID
CONCAT
CTRACE
DB2
DB2CLOSE
DBERROR
DBRM
DECLARE
DIRECTIVES
FILLSYSCAT
GENSQLCA
HYPHEN-IN-CURSOR
LOCATION
MAXSQL
NEVERCLOSE
NOT
PKGSET
SAVE-RETURN-CODE
SQLDS
STRICT-DB2
VALIDATE
VALIDATE-ERR_LVL
VALIDATE-LOGIN
XDBFUNCS

The ***SQL Option Preprocessor User's Guide*** describes in detail the syntax and options for each directive and which directives are obsolete. Some options, such as DB2, are a default value. Most options have an alternate (or negative) syntax whereby the condition established by the option can be turned off. The default syntax for each option is underlined. SQL Option directive options are not case sensitive.

10.11.8 Error Messages

All error messages generated by the SQL Option preprocessor have the format of SQnnnnl where "nnnn" is the error message number and "l" is the severity.

The ***SQL Option Preprocessor User's Guide*** describes each error message and possible causes.

10.11.9 Linking

To link an application:

1. Open the Net Express project and set the **Type of Build** to **Generic Release Build**
2. Right click on the **.exe** or **.dll** file

3 If you get an unresolved symbol _XDBINTRF error, try doing this:

a Select **Build Settings ...** and then click the **Link** tab

b Set the **Category** to **Advanced**

c In the **Link with these LIB's** edit box enter:

```
xdbintrf.lib
```

d Then try re-linking the object

e If the problem persists, check to make sure the LIB environment variable has the SQL Option directory included. You can determine this by selecting the **Project** menu, clicking **Properties**, clicking **IDE** and then clicking **Import** . Scroll down the list box displayed until you find the **LIB** environment variable.

10.11.10 Distributing Your Application

If you distribute your application to other machines using **.exe** or **.dll** files that include SQL Option access logic, you may need to include the following **.dll** files:

- xdbintrf.dll
- xdbcrun.dll

10.12 SQL Option NLS Environment

SQL Option allows you to create XDB locations that emulate all the various mainframe EBCDIC code pages. When you access these locations with SQL Wizard or a COBOL program, your data in these locations appears identical to the data on the mainframe.

The five steps necessary to ensure complete compatibility with the mainframe and/or Net Express' IDE are as follows:

1 Decide which EBCDIC code page you want to emulate.

Typically, you would use the mainframe's code page. (If you do not know what it is, you may be able to use XDB Link to find out.) If you do not have a mainframe, you can choose whichever code page you prefer.

2 Create a local EBCDIC location for this code page.

Use SQL Wizard's **Create Location** dialog box. Pick the EBCDIC code page you want in the **Sort Sequence** box. For more information, click **SQL For DB2 Help** > **SQLWizard** on the IDE **Help** menu.

3 Ensure that your project's national language support (NLS) setting is the same as your EBCDIC location.

When you compile a COBOL program with Net Express, it can be associated with a particular EBCDIC code page. For information on how to do this, click **Help Topics** on the **Help** menu, click the **Contents** tab, and select **Development Environment**, **Working with Data Files**, **Configurable Codesets**, **How to**.

4 If you have XDB Link, configure it to access the mainframe.

Use the Gateway Profile utility to define your DB/2 locations to the PC software, then specify the single Workstation Code Page as 911 (in the local Workstation Configuration). Next, use SQL Wizard to access the DB/2 location on the mainframe. A file named **cpg_info.txt** is created in your **mfsql\bin** folder, identifying the EBCDIC code page of the DB/2 location on the mainframe.

The following table defines this EBCDIC code page number:

Net Express NLS Setting	Code Page Number
31 Dutch	37
33 French	297
34 Spanish	284
39 Italian	280
43 Austrian German	273
44 English (UK)	285
45 Danish	277
46 Swedish	278
47 Norwegian	277
49 German	273
351 Portugese	37
358 Finnish	278
437 English (US)	37
500 International	500

5 Populate your local EBCDIC location with data from the mainframe.

If you have XDB Link for the mainframe, there are two ways to do this: using Migrate or using SQL Wizard.

- To use Migrate, all you need to do is point to the DB/2 location on your mainframe as the source location, and point to the local EBCDIC location as the destination location.
- To use SQL Wizard, first export your mainframe's data in DSNTIAUL format, then import it into SQL Wizard. When you use the DSNTIAUL format, you need to specify a code page translation for both exports and imports: for exports, use the PC's ANSI code page to EBCDIC code page translation; for imports, use the reverse translation. For example, suppose you want to extract from a Swedish EBCDIC DB/2 location. For the export, you would pick the ANSI to EBCDIC translation **1252 - 278**, for the import, **278 - 1252**.

If you don't have XDB Link for the mainframe, extract the data from the mainframe into a file with DSNTIAUL format, then import the data into your local EBCDIC location using SQL Wizard's import function, citing the appropriate EBCDIC to ANSI code page translation.

XDB EBCDIC databases are stored using the default OEM character set for your machine (typically, code page 437 in the United States, or code page 850 almost everywhere else). The names of database objects such as tables and indexes are also validated using this character set. You can optionally specify that database object-names should be validated using any of the following character sets if, for example, you want to use accented characters in the names:

MS-DOS Code Page	Character Set
437	US Latin 1
850	International Latin 1
857	Turkish
863	French Canada
865	Nordic
860	Portuguese

To specify that an XDB Server should use one of the supported code pages for name validation, you must add two lines to the **xdb.ini** file in your **\mfuser\config** folder:

```
[SERVER]
XDBCP=codepage
```

where *codepage* is the three-digit number of the code page you require.

Where a location has been created as an EBCDIC location (for example, the default location, MAINTAIN), data is automatically converted to EBCDIC before it is passed to the client application. Conversion between ASCII data on the PC and EBCDIC data on the mainframe is handled by the DB2 Link gateway if you are using the Migrate utility. If you are using the data import and export facilities in SQL Wizard, you can control the conversion used by choosing the DSNTIAUL data format and selecting an appropriate code page conversion. See the section *Import/Export NLS Considerations* for more information.

10.13 Using Existing XDB Data

If you already use XDB databases, you might be able to access them using SQL Option for DB2, providing that the locations in which they were created are compatible.

10.14 Tips

- Use the Query Design window in SQL Wizard when you want to add new DML commands to your program but are not sure of the correct syntax. Create your query and run it until you are happy that results are correct. Then open the SQL window and copy the SQL statements into your program, between the EXEC SQL and the end-of-procedure statements.
- If you are writing a program that needs to create a table and you have a test table with the correct structure, use SQL Wizard's Export capabilities to generate the DDL commands for your program.
- If you create date columns in a test table using the WITH DEFAULT constraint, you can use the Date Warp feature of the IDE to change the system date on a row-by-row basis, without having to restart the XDB Server.
- SQL Option automatically commits changes to a database when the end of a program procedure is reached with no errors. If an error occurs or if you stop debugging prior to reaching the end of a procedure, a rollback is performed instead. You do not need to use the LOGOFF directive when running a COBOL application.

Database Access

11 Stored Procedures

This chapter introduces stored procedures, and describes how they work under OpenESQL and DB2.

A stored procedure is a compiled program that can execute SQL statements.

You need to use stored procedures if your application program operates in a client/server environment, and if either of the following two problems apply:

- The application accesses host variables for which you want to guarantee security and integrity.
- The application executes a series of SQL statements, creating many network send and receive operations, which significantly increase CPU and elapsed time costs.

11.1 OpenESQL Stored Procedures

OpenESQL supports two statements that are used with stored procedures:

- CALL

  Provides generic support for ODBC stored procedure calls.
- EXECSP

  Provides backwards compatibility with the Micro Focus Embedded SQL Toolkit for Microsoft SQL Server.

A stored procedure can:

- Accept input parameters
- Return output parameters
- Accept and return input/output parameters

- Use positional or keyword parameters
- Return a result
- Return result sets
- Be called with parameter arrays

Note: The features provided by different database vendors vary considerably, and any given vendor will offer only a subset of the features listed above. For this reason, stored procedure calls are much less portable between data sources than other OpenESQL statements.

When a stored procedure is called, any parameters are passed as a comma separated list, optionally enclosed in parentheses. A parameter can be a host variable or a literal, or the keyword CURSOR. The keyword CURSOR causes the parameter to be unbound, and should only be used with Oracle 8 stored procedures which return result sets.

If the parameter is a host variable it can be followed by one of the following words, which indicate the parameter type: IN, INPUT, INOUT, OUT, OUTPUT. If no parameter type is specified, INPUT is assumed.

Host variable parameters can be passed as keyword parameters, by preceding the host variable with the formal parameter name and an equals sign:

```
EXEC SQL CALL myProc (keyWordParam = :hostVar) END-EXEC
```

For maximum portability, literal parameters should be avoided and only host variable parameters should be used, and a given call should use either only normal, positional parameters or keyword parameters, but not both. Some servers support a mixture, but keyword parameters should occur after all positional parameters. Keyword parameters are useful as an aid to readability and where the server supports default parameter values and optional parameters.

If a stored procedure call returns a result set, it must be used in a cursor declaration, thus:

```
EXEC SQL
    DECLARE cursorName CURSOR FOR storedProcecureCall
```

The stored procedure is then called by OPENing the cursor and FETCHing result set rows, like any other type of cursor.

Currently OpenESQL supports only a single result set.

ODBC parameters differ fom Oracle array parameters. The effect of using a parameter array is the same as repeating the statement for each element of the array. On a stored procedure call, if one parameter is passed as an array, then all parameters must be arrays with the same number of elements. The stored procedure will "see" one call for each "row" of parameters. The number of rows passed can be limited to less than the full array size by preceding the call with the phrase **FOR :hvar** where **:hvar** is an integer host variable containing a count of the numer of rows to be passed.

Note: The Net Express online help provides the structure and examples for both the CALL and EXECSP statements. (Click **Help Topics** on the **Help** menu. Then, on the **Index** tab, click **CALL** or click **EXECSP**.)

11.2 DB2 Stored Procedures

As we have seen above, a stored procedure is a compiled program that can execute SQL statements. Stored procedures are stored at a local or remote DB2 Universal Database Server. Local DB2 Universal Database Server applications or remote DRDA applications can use the SQL statement CALL to invoke a stored procedure.

Use stored procedures to combine many of your application’s SQL statements into a single message to the DB2 subsystem or DB2 Universal Database Server, reducing network traffic to a single send and receive operation for a series of SQL statements.

Note: If you are running your stored procedure on DB2 for OS/390, through the DB2 Connect product, see your IBM DB2 documentation for other steps necessary to run the stored procedure.

11.2.1 Working with Stored Procedures

To get a stored procedure up and running:

1. Either add the Net Express executable directory (**base\bin**) to the PATH statement

   orcopy the COBOL run-time **.dll** (**cblrtss.dll** if using single threaded run-time) to the directory from where the stored procedure will be executed. You need to do this for DB2 to be able to execute a COBOL stored procedure.

2. Code and prepare a stored procedure. See the section Writing and Preparing Stored Procedures for instructions.

3. Code and prepare an application that calls the stored procedure. An SQL statement, CALL, in that application must use the same parameter list and linkage convention as the stored procedure that it invokes. See the section Writing and Preparing Applications to Use Stored Procedures for instructions.

4. Define your stored procedure to the DB2 Universal Database Server by issuing a CREATE PROCEDURE command, which will place a row in the appropriate system table(s). See the section Defining Stored Procedures under DB2 Universal Database for additional details.

5. Compile your stored procedure. See the section Compiling Stored Procedures under DB2 Universal Database for additional details.

6. Debug and test your stored procedure. See the section Debugging Stored Procedures under DB2 Universal Database for additional details.

11.2.2 Writing and Preparing Stored Procedures

A stored procedure is an application program that runs in the DB2 Universal Database Server’s address space. It can contain most statements that an application program normally contains. It can consist of more than one program. Your stored procedure can call other programs as well as nested stored procedures but there are restrictions. See the ***DB2 Universal Database Application Development Guide*** for details.

The application program that invokes the stored procedure can be in any language that supports SQL statements. You can write a stored procedure using many languages such as C, JAVA, COBOL or now SQL Procedure Language which is consistent with the Persistent Stored Module definition of the ANSI SQL99 standard.

The store procedure can be written in one language, for example JAVA, with the client written in another language, for example COBOL. When the languages differ, DB2 transparently passes the values between the client and the stored procedure so each program gets the values in the expected format defined by the CREATE PROCEDURE statement.

11.2.2.1 Features of a Stored Procedure

A stored procedure is similar to any other SQL application.

The following restrictions apply to stored procedures:

- A C or COBOL stored procedure must be a dynamic link library (DLL) when built to run under Windows server.
- To use SQL Procedure Language, you need to have a C compiler installed since the SQL statements generate a C program.
- The following SQL statements are not allowed in a stored procedure:
    - CONNECT
    - DISCONNECT
    - SET CONNECTION

See the ***DB2 Universal Database Application Development Guide*** for a complete list.

11.2.2.2 Preparing Stored Procedures

The following are tasks which must be completed before a stored procedure can be run on an DB2 Universal Database Server:

1. Prepare the application according to your embedded SQL documentation for creating stored procedure.

2 Define the stored procedure to the DB2 Universal Database Server. See Defining Stored Procedures under DB2 Universal Database.

3 Place the stored procedure DLL or JAVA routine on the DB2 Universal Database Server machine in a location specified in the CREATE PROCEDURE. If not specified, the default location is the **sqllib\function** subdirectory where DB2 Universal Database is installed. See the ***DB2 Universal Database Application Development Guide*** and ***DB2 Universal Database SQL Reference*** for additional options.

11.2.2.3 How an Application Works with a Stored Procedure

A typical stored procedure contains two or more SQL statements, and some manipulative or logical processing. In this example, your application, CALLSTPR, runs on a workstation client and calls a stored procedure, GETEMPSVR. The following process occurs:

1 The workstation application establishes a connection to the DB2 Universal Database Server.

2 The SQL statement CALL tells the DB2 Universal Database Server that the application is going to run the stored procedure, GETEMPSVR. The calling application provides the necessary parameters.

3 The DB2 Universal Database Server searches the system tables for rows associated with stored procedure GETEMPSVR.

4 The DB2 Universal Database Server passes information about the request to the stored procedure.

5 The stored procedure executes SQL statements.

6 The stored procedure assigns values to the output parameters and then exits.

7 Control returns to the calling application, which receives the output parameters.

The application can call more stored procedures or it can execute more SQL statements. The application designer determines whether to COMMIT work in the stored procedure that runs on the server, or in the client as a single transaction.

11.2.3 Writing and Preparing Applications to Use Stored Procedures

To invoke a stored procedure and to pass a list of parameters to the procedure, use the SQL statement CALL. Your application program can call several stored procedures.

After connecting to its server, an application can mix calls to stored procedures with SQL statements sent to the server.

11.2.3.1 Executing the SQL Statement CALL

Use the CALL statement to execute each series of SQL statements in your application.

11.2.3.1.1 Example 1:

To call the stored procedure described in How an Application Works with a Stored Procedure, your application might use this statement:

```
EXEC SQL
 CALL GETEMPSVR (:V1, :V2)
END-EXEC
```

If you use host variables in the CALL statement, you must declare them before using them.

11.2.3.1.2 Example 2:

The example above is based on the assumption that none of the input parameters can have null values. To allow null values, code a statement like this:

```
EXEC SQL
 CALL GETEMPSVR (:V1 :IV1, :V2 :IV2)
END-EXEC
```

where :IV1 and :IV2 are indicator variables for the parameters.

11.2.3.1.3 Example 3:

To pass integer or character string constants or the null value to the stored procedure, code a statement like this:

```
EXEC SQL
 CALL GETEMPSVR (2, NULL)
END-EXEC
```

11.2.3.1.4 Example 4:

To use a host variable for the name of the stored procedure, code a statement like this:

```
EXEC SQL
 CALL :PROCNAME (:V1, :V2)
END-EXEC
```

11.2.3.1.5 Example 5:

Assume that the stored procedure name is GETEMPSVR. The host variable PROCNAME is a character variable of length 254 or less that contains the value GETEMPSVR. You should use this technique if you do not know in advance the name of the stored procedure, but you do know the parameter list convention.

To pass your parameters in a single structure, rather than as separate host variables, code a statement like this:

```
EXEC SQL
 CALL GETEMPSVR USING DESCRIPTOR :ADDVALS
END-EXEC
```

where ADDVALS is the name of an SQLDA.

11.2.3.1.6 Example 6:

To use a host variable name for the stored procedure with an SQLDA, code a statement like this:

```
EXEC SQL
 CALL :PROCNAME USING DESCRIPTOR :ADDVALS
END-EXEC
```

This form gives you extra flexibility because you can use the same CALL statement to invoke different stored procedures with different parameter lists.

Your client program must assign a stored procedure name to the host variable PROCNAME and load the SQLDA ADDVALS with the parameter information before making the SQL CALL statement.

Each of the above CALL statement examples uses an SQLDA. If you do not explicitly provide an SQLDA, the precompiler generates the SQLDA based on the variables in the parameter list.

You can execute the CALL statement only from an application program. You cannot use the CALL statement dynamically.

11.2.3.2 Parameter Conventions

When an application executes the CALL statement, the DB2 Universal Database Server builds a parameter list for the stored procedure, using the parameters and values provided in the statement. The DB2 Universal Database Server obtains information about parameters from the system tables. See the section Defining Stored Procedures under DB2 Universal Database for more information. Parameters are defined as one of these types:

- IN

  Input-only parameters, which provide values to the stored procedure.

- OUT

  Output-only parameters, which return values from the stored procedure to the calling program.

- INOUT

  Input/output parameters, which provide values to or return values from the stored procedure.

If a stored procedure fails to set one or more of the output-only parameters, the DB2 Universal Database Server simply returns the output parameters to the calling program, with the values established on entry to the stored procedure. COBOL supports three parameter list conventions. Other languages support other conventions. The parameter list convention is chosen based on the parameter style defined in the CREATE PROCEDURE statement.

Parameter Style	Description
SIMPLE	Use SIMPLE (or GENERAL) to prevent the calling program passing null values for input parameters (IN or INOUT) to the stored procedure. The stored procedure must declare a variable for each parameter passed in the CALL statement.
SIMPLE WITH NULLS	Use SIMPLE WITH NULLS (or GENERAL WITH NULLS) to allow the calling program to supply a null value for any parameter passed to the stored procedure. The following rules apply: • An indicator variable must follow each parameter in the calling program's CALL statement, using one of the following two forms: • host variable :indicator variable or: • host variable INDICATOR :indicator variable • The stored procedure must declare a variable for each parameter passed in the CALL statement. • The stored procedure must declare a null indicator structure containing an indicator variable for each parameter passed in the CALL statement. • On entry, the stored procedure must examine all indicator variables associated with input parameters to determine which parameters contain null values. • On exit, the stored procedure must assign values to all indicator variables associated with output variables. An indicator variable for an output variable that returns a null value to the caller must be assigned a negative number. Otherwise, the indicator variable must be assigned the value zero (0).

Parameter Style	Description
DB2SQL	In addition to the parameters on the CALL statement, the following arguments are passed to the stored procedure: • a NULL indicator for each input parameter on the CALL statement. • the SQLSTATE to be returned to DB2. • the qualified name of the stored procedure. • the specific name of the stored procedure. • the SQL diagnostic string to be returned to DB2.

11.2.3.3 Using Indicator Variables to Speed Processing

If any of your output parameters require a lot of storage, do not pass the entire storage areas to your stored procedure, but declare an indicator variable for every large output parameter in your SQL statement CALL. Indicator variables are used in the calling program to pass only a two-byte area to the stored procedure and to receive the entire area from the stored procedure.

Note: If you are using the SIMPLE WITH NULLS linkage convention, you must declare indicator variables for all of your parameters, so you do not need to declare another indicator variable for the large output parameters.

Assign a negative value to each indicator value associated with a large output variable. Then include the indicator variables in the CALL statement. This technique can be used whether the stored procedure linkage is SIMPLE or SIMPLE WITH NULLS.

For example, suppose that a stored procedure, STPROC2 defined with the SIMPLE linkage convention takes one integer input parameter and one character output parameter of length 5000. It is wasteful to pass the 5000 byte storage area to the stored procedure. Instead, a COBOL program containing these statements passes only two bytes to the stored procedure for the output variable and receives all 5000 bytes from the stored procedure:

```
INNUM  PIC S9(9) COMP
OUTCHAR PIC X(5000)
IND  PIC S9(4) COMP
.
.
.
MOVE -1                      TO IND
EXEC SQL CALL STPROC2(:INNUM, :OUTCHAR :IND)  END-EXEC
```

11.2.3.4 Declaring Data Types for Passed Parameters

A stored procedure must declare each parameter passed to it[1] . In addition, the PARMLIST column in the DECLARE PROCEDURE must contain a compatible SQL data type declaration for each parameter. For PARMLIST string and corresponding language declarations, see the SQL data types table in the Net Express ***Database Access*** manual.

For example:

```
CREATE PROCEDURE GETEMPSVR
 (IN  EMPNO CHAR(6),
  INOUT SQLCD    INT ,
  OUT FIRSTNME CHAR(12),
  OUT LASTNAME CHAR(12),
  OUT HIREDATE CHAR(10),
  OUT SALARY   DEC(9,2) )
  LANGUAGE COBOL
  EXTERNAL NAME 'GETEMPSVR!GETEMPSVR'
  PARAMETER STYLE DB2SQL;
```

11.2.3.5 Limitations

IBM has not implemented the same support for all SQL syntax related to stored procedures in every supported language. For example, you cannot create COBOL stored procedures that use result sets or the EXEC SQL syntax that supports that functionality with the workstation version of DB2 Universal Database. This might change in the future. See the ***DB2 Universal Database SQL Reference*** and the ***DB2 Application Development Guide*** for details of which functions are supported and by what language.

There is no support for structure, array, or vector parameters using the DB2 native precompiler. However, there is much more flexibility when using the OpenESQL precompiler and a ODBC connection. See the Net Express ***Database Access*** manual for more details.

11.2.4 Defining Stored Procedures under DB2 Universal Database

A stored procedure is unusable until it is defined[1] - use the CREATE PROCEDURE command to do this. You can either use the DB2 command prompt or place the command in a program and compile and run it. If you use the DB2 command prompt, you first connect to the DB2 Universal Database Server where the stored procedure will be executed.

For example:

```
C:> db2 connect to sample
```

You can type in the command at the DB2 command prompt making sure you include continuation characters and command delimiters, or you can place the CREATE PROCEDURE in an ANSI text file. For example, if we placed the previous command in text file **creproc.sql**, the command that you would enter would be:

```
C:> db2 -td; -vf creproc.sql
```

where:

- the "-td" option indicates the next character is the delimiter to end the command. In our example, it is a semicolon (;).
- the "-vf" option indicates that the next token is the file to process that contains the SQL command script.

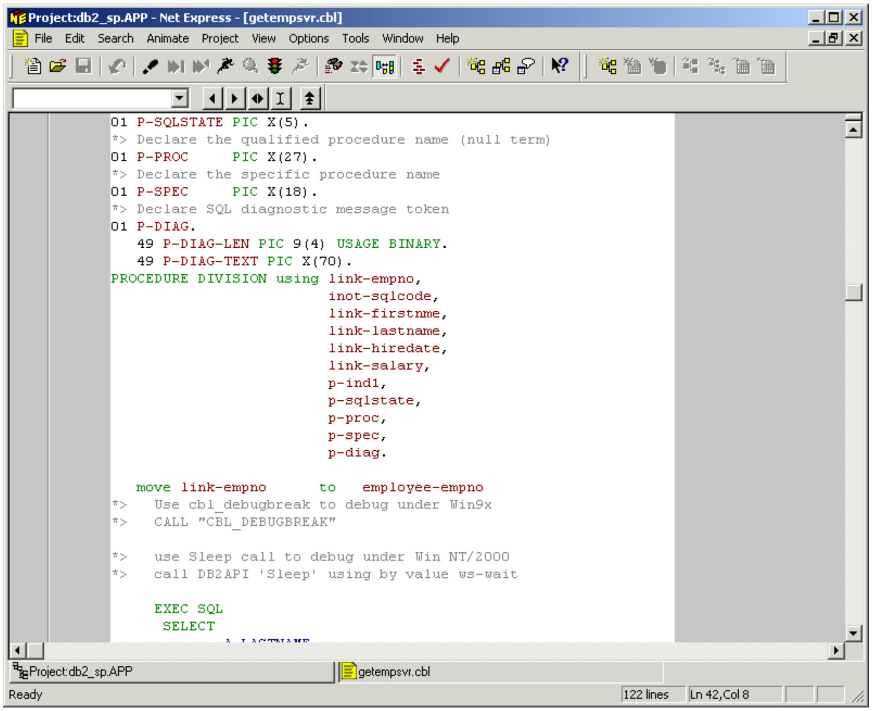
The create procedure statement must uniquely identify a stored procedure. If you want to change the stored procedure to either add or drop parameters or change functionality, you must use the DROP PROCEDURE command and then re-add it with the CREATE PROCEDURE command.

[1] When DB2/2 was originally developed, DB2/2 did not support the CREATE PROCEDURE function, and it is possible to write COBOL stored procedures without doing a CREATE PROCEDURE. Examples of this method and the parmlist that is required are included in the DB2 Universal Database Application Development Client.

11.2.5 Compiling Stored Procedures under DB2 Universal Database

To compile a COBOL stored procedure using Net Express and DB2 Universal Database, follow the steps below:

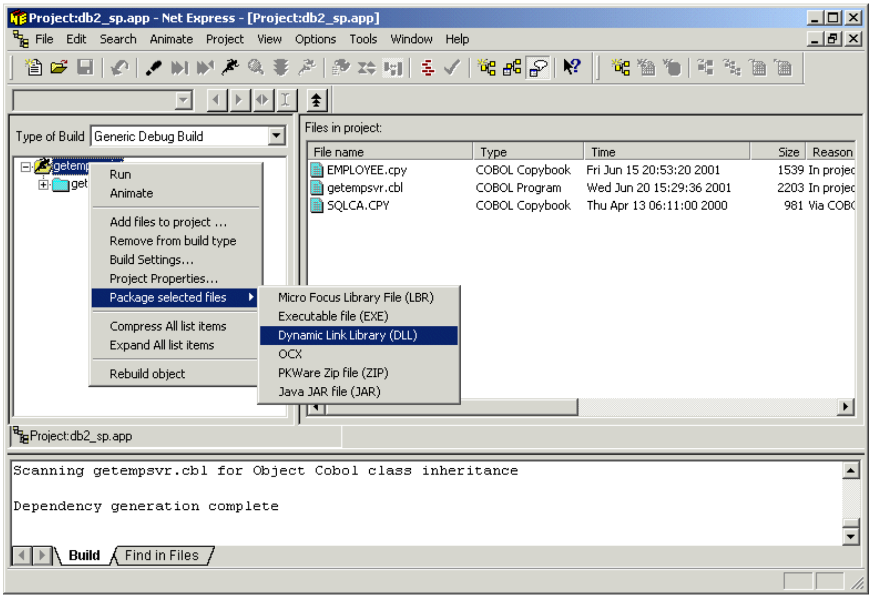
1 Compile the program which is to be used as a stored procedure with the DB2 directive, just like any DB2 Universal Database program. This can be done by adding a $SET statement to your program. See the Net Express ***Database Access*** manual for more details on DB2 directive options.



Project:db2_sp.APP - Net Express - [getempsvr.cbl]
File Edit Search Animate Project View Options Tools Window Help

```
01 P-SQLSTATE PIC X(5).
*> Declare the qualified procedure name (null term)
01 P-PROC     PIC X(27).
*> Declare the specific procedure name
01 P-SPEC     PIC X(18).
*> Declare SQL diagnostic message token
01 P-DIAG.
   49 P-DIAG-LEN PIC 9(4) USAGE BINARY.
   49 P-DIAG-TEXT PIC X(70).
PROCEDURE DIVISION using link-empno,
                         inot-sqlcode,
                         link-firstnme,
                         link-lastname,
                         link-hiredate,
                         link-salary,
                         p-ind1,
                         p-sqlstate,
                         p-proc,
                         p-spec,
                         p-diag.

   move link-empno      to   employee-empno
*>   Use cbl_debugbreak to debug under Win9x
*>   CALL "CBL_DEBUGBREAK"

*>   use Sleep call to debug under Win NT/2000
*>   call DB2API 'Sleep' using by value ws-wait

     EXEC SQL
      SELECT
```

2 After the program has been added to the Net Express project, select the program and package it as a Dynamic Link Library (**.dll**).



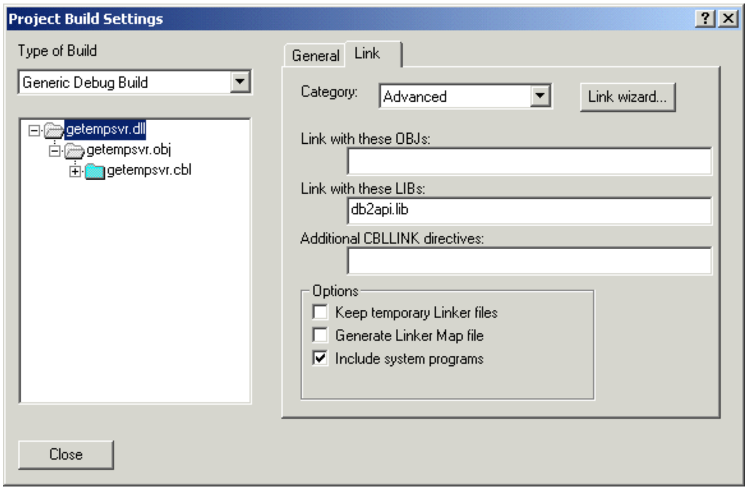
Type of Build: Generic Debug Build

Files in project:

File name	Type	Time	Size	Reason
EMPLOYEE.cpy	COBOL Copybook	Fri Jun 15 20:53:20 2001	1539	In projec
getempsvr.cbl	COBOL Program	Wed Jun 20 15:29:36 2001	2203	In projec
SQLCA.CPY	COBOL Copybook	Thu Apr 13 06:11:00 2000	981	Via COB(

```
Scanning getempsvr.cbl for Object Cobol class inheritance

Dependency generation complete
```

3 In the build setting for the program, select the Link tab and select the Advanced category from the drop-down list box. In the "Link with these Libs" entry field, add **db2api.lib** and then click the **Close** button.



Type of Build: Generic Debug Build

Category: Advanced

Link with these OBJs:

Link with these LIBs: db2api.lib

Additional CBLLINK directives:

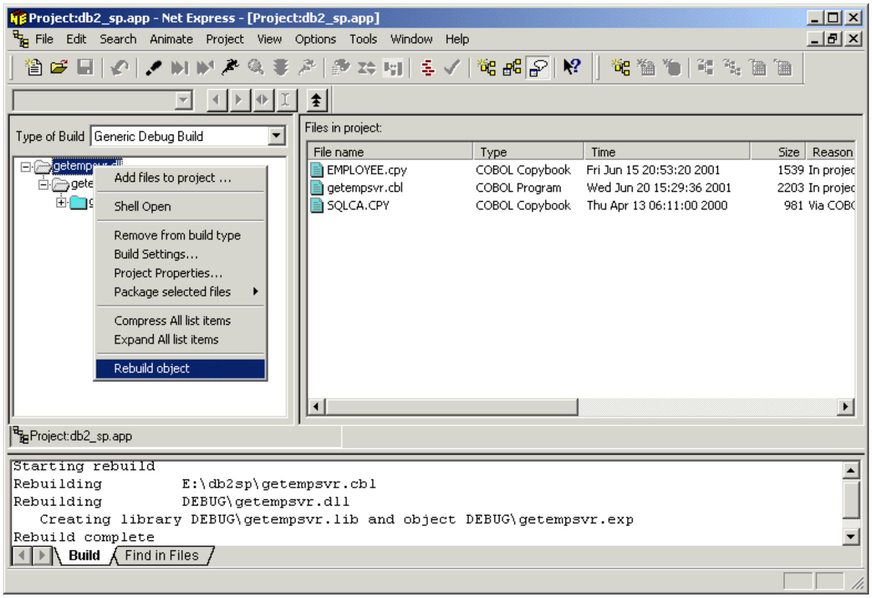
Options

☐ Keep temporary Linker files

☐ Generate Linker Map file

☑ Include system programs

4 To compile and link the stored procedure, you can then select the **.dll** file and select "Rebuild object" from the context menu.

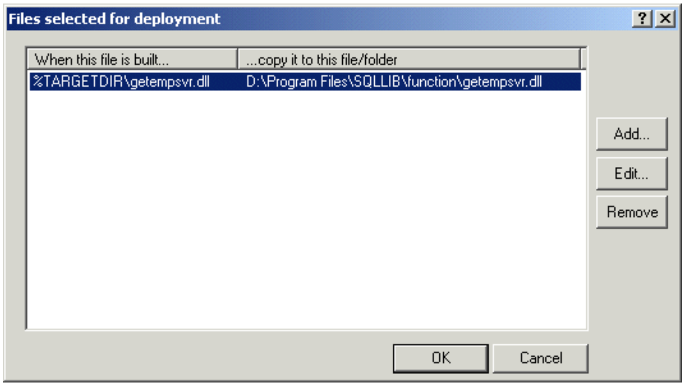


Files in project:

File name	Type	Time	Size	Reason
EMPLOYEE.cpy	COBOL Copybook	Fri Jun 15 20:53:20 2001	1539	In projec
getempsvr.cbl	COBOL Program	Wed Jun 20 15:29:36 2001	2203	In projec
SQLCA.CPY	COBOL Copybook	Thu Apr 13 06:11:00 2000	981	Via COBC

```
Starting rebuild
Rebuilding          E:\db2sp\getempsvr.cbl
Rebuilding          DEBUG\getempsvr.dll
   Creating library DEBUG\getempsvr.lib and object DEBUG\getempsvr.exp
Rebuild complete
```

5 Depending upon how you defined your CREATE PROCEDURE, you are now ready to either test your stored procedure or copy it to the **sqllib\function** subdirectory. From the "Rebuild object" command, select "Deployment" from the **Project** menu and select the file and specify where to copy it to.



When this file is built...	...copy it to this file/folder
%TARGETDIR\getempsvr.dll	D:\Program Files\SQLLIB\function\getempsvr.dll

11.2.6 Debugging Stored Procedures under DB2 Universal Database

Because debugging a stored procedure by its very nature implies stopping the server and stepping through the server code, it is imperative that the programmer debugging the stored procedure code is the only person using the database server. You should therefore debug DB2 Universal Database stored procedures using a database on your own workstation if possible.

You can test a COBOL stored procedure without coding a client program by using:

- the IBM Stored Procedure Builder to run a stored procedure. You connect to the database where the stored procedure is located, and then select the stored procedure you want to run. The Store Procedure Builder then prompts you for INPUT and INOUT values. The results and any errors are then displayed.
- the DB2 command prompt.

If you don't get the expected results, you might want to debug the COBOL stored procedure. If you are running Net Express under Windows 9x, you can just add a call to CBL_DEBUGBREAK to your stored procedure and the Net Express debugger will display when the statement is executed.

If you are running under Windows NT / 2000, DB2 stored procedures are executed under the db2dari process which means that a call to CBL_DEBUGBREAK will not work. You need to compile the stored procedure with a "sleep" function. To do this, define a variable in Working-Storage that will define how long to put the variable to sleep.

For example:

```
01  ws-wait                 pic 9(8) comp-5 value 30000.
```

Then in the program add this statement so that it executes before you want to start animating:

```
call DB2API ’Sleep’ using by value ws-wait
```

where DB2API is defined as call-convention 74 in a Special Names paragraph.

To animate the stored procedure:

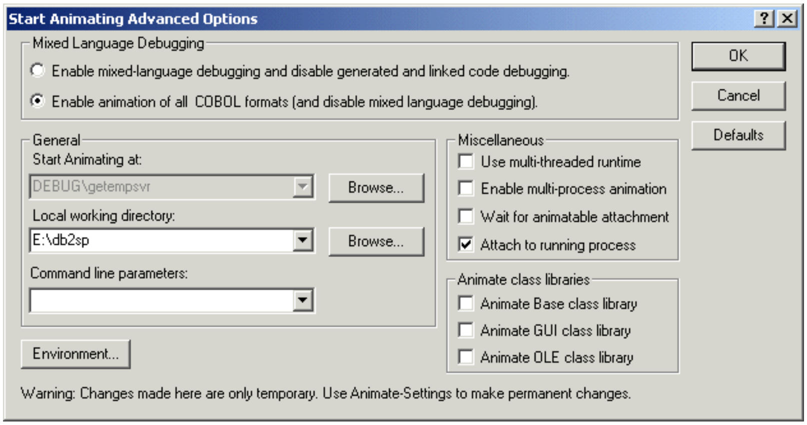
1 Invoke the stored procedure by using the IBM Stored Procedure Builder or running a client program.

2 Start Net Express and click the **Step** button to display the Start Animating window.

You must have Administrator authority to attach a debugger to a running process.

3 Click the **Options** button.

4 Check the "attach to running process" check box and then click the **OK** button.



Start Animating Advanced Options

Mixed Language Debugging

○ Enable mixed-language debugging and disable generated and linked code debugging.

● Enable animation of all COBOL formats (and disable mixed language debugging).

General

Start Animating at: DEBUG\getempsvr

Local working directory: E:\db2sp

Command line parameters:

Miscellaneous

☐ Use multi-threaded runtime

☐ Enable multi-process animation

☐ Wait for animatable attachment

☑ Attach to running process

Animate class libraries

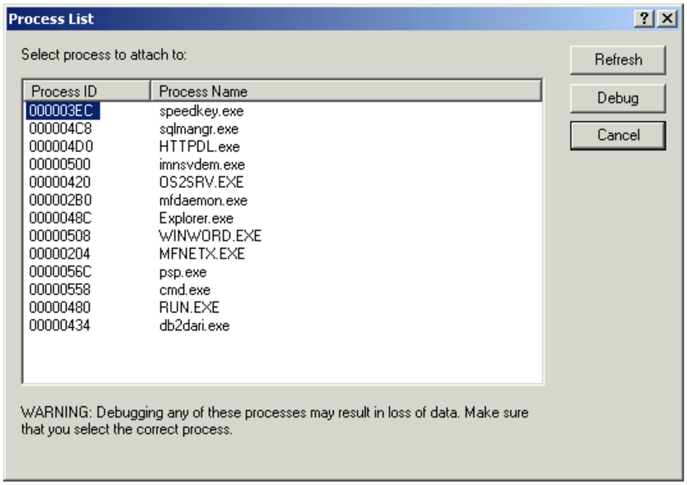
☐ Animate Base class library

☐ Animate GUI class library

☐ Animate OLE class library

Warning: Changes made here are only temporary. Use Animate-Settings to make permanent changes.

5 Select the process associated with **db2dari.exe** and then click the **Debug** button. If the db2dari process is not listed yet, click the **Refresh** button.



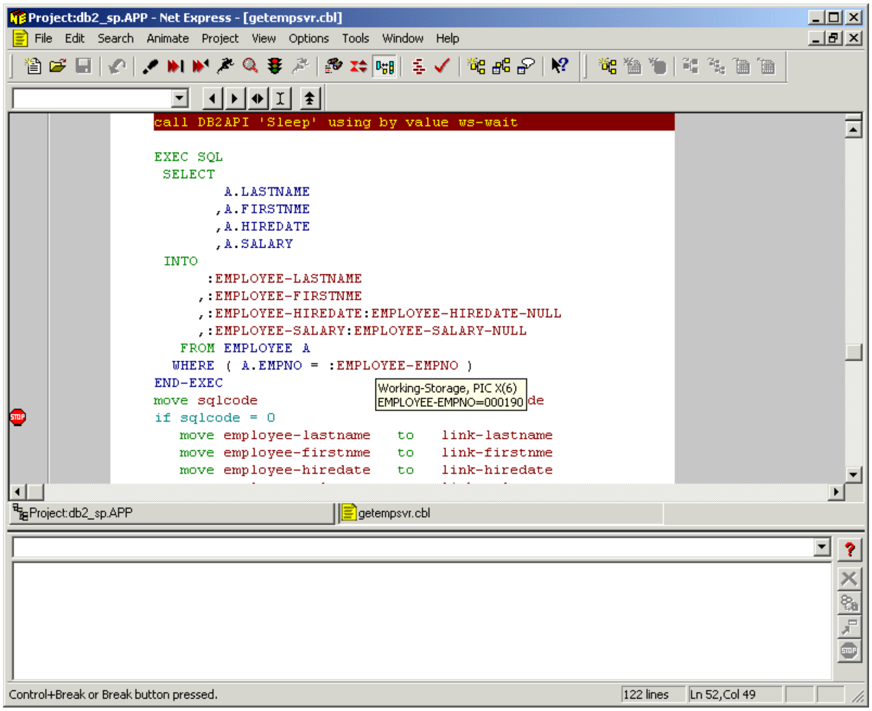
Select process to attach to:

Process ID	Process Name
000003EC	speedkey.exe
000004C8	sqlmangr.exe
000004D0	HTTPDL.exe
00000500	imnsvdem.exe
00000420	OS2SRV.EXE
000002B0	mfdaemon.exe
0000048C	Explorer.exe
00000508	WINWORD.EXE
00000204	MFNETX.EXE
0000056C	psp.exe
00000558	cmd.exe
00000480	RUN.EXE
00000434	db2dari.exe

WARNING: Debugging any of these processes may result in loss of data. Make sure that you select the correct process.

6 Click the **Break** button. The debugger is displayed.

7 Set a break point on the COBOL statement you want to start animating and then click the **Run** button. Modify the value of the wait time to "1" so that you do not have to wait for the sleep timer to expire.


Project:db2_sp.APP - Net Express - [getempsvr.cbl]
File Edit Search Animate Project View Options Tools Window Help
Project:db2_sp.APP
getempsvr.cbl
122 lines
Ln 52,Col 49

```
call DB2API 'Sleep' using by value ws-wait

EXEC SQL
 SELECT
        A.LASTNAME
       ,A.FIRSTNME
       ,A.HIREDATE
       ,A.SALARY
 INTO
      :EMPLOYEE-LASTNAME
     ,:EMPLOYEE-FIRSTNME
     ,:EMPLOYEE-HIREDATE:EMPLOYEE-HIREDATE-NULL
     ,:EMPLOYEE-SALARY:EMPLOYEE-SALARY-NULL
   FROM EMPLOYEE A
  WHERE ( A.EMPNO = :EMPLOYEE-EMPNO )
END-EXEC
move sqlcode                  de
if sqlcode = 0
   move employee-lastname   to   link-lastname
   move employee-firstnme   to   link-firstnme
   move employee-hiredate   to   link-hiredate
```

```
Working-Storage, PIC X(6)
EMPLOYEE-EMPNO=000190
```

Control+Break or Break button pressed.

When you have completed debugging the stored procedure, just exit from the Animator.

Tip: If the stored procedure is not working as expected, make sure the parameters passed to the stored procedure are getting passed in the expected format by examining each parameter in the Linkage Section.

Part 4: COBSQL

This part contains the following chapters:

- Chapter 11, “COBSQL”

12 COBSQL

COBSQL is an integrated preprocessor designed to work with COBOL precompilers supplied by relational database vendors. It is intended for use with:

- Oracle Pro*COBOL Version 1.8
- Oracle Pro*COBOL Version 8.04
- Informix Embedded SQL/COBOL Version 9.x
- Sybase Open Client Embedded SQL/COBOL Version 11.5

You should use COBSQL if you are already using either of these precompilers with an earlier version of a Micro Focus COBOL product and want to migrate your application(s) to Net Express, or if you are creating applications that will be deployed on UNIX platforms and need to access either Oracle or Sybase relational databases.

For any other type of embedded SQL application development, we recommend that you use OpenESQL.

Note: The Oracle Version 1.8 precompiler does not support nested programs. COBSQL does not support Object Oriented COBOL syntax (OO COBOL). If you want to use OO COBOL, therefore, you must use OpenESQL.

11.1 Overview

You can access the SQL functions offered by the Oracle, Sybase or Informix Database Management System (DBMS) by embedding SQL statements within your COBOL program in the form:

```
EXEC SQL
   SQL statement
END-EXEC
```

and then using the Oracle, Sybase or Informix precompiler to process the embedded SQL before passing the program to the COBOL Compiler. The database precompiler replaces embedded SQL statements with the appropriate calls to database services. Other additions are made to the source code to bind COBOL host variables to the SQL variable names known to the database system.

The advantage of embedding SQL in this way is that you do not need to know the format of individual database routine calls. The disadvantage is that the source code that you see when you animate your program is that output by the precompiler and not the original embedded SQL. You can overcome this disadvantage by using COBSQL.

COBSQL provides an integrated interface between Micro Focus COBOL and the third-party standalone precompiler, enabling you to animate a program containing EXEC SQL statements and display your original source code rather than the code produced by the precompiler.

This chapter shows you how you can use COBSQL in conjunction with either the Oracle, Sybase or Informix precompiler to compile and animate your programs.

11.2 Operation

To use COBSQL, specify the PREPROCESS"COBSQL" Compiler directive when you compile your program. All directives following it are passed from the Compiler to COBSQL. You can specify Compiler directives by using $SET statements in your program or via the Net Express **Build Settings** screen.

To terminate the directives to be passed to COBSQL, you must use the ENDP COBOL directive. You can do this by adding the following line either to the project settings or to the end of the Net Express Build settings:

```
Preprocess(Cobsql) csqltype=database_product end-c
    comp5=yes endp;
```

where *database_product* is one of Oracle, Sybase or Informix. For example, for Oracle:

```
Preprocess(Cobsql) csqltype=oracle end-c comp5=yes endp;
```

Note: Net Express ignores any directives placed after the semi-colon (;) in the Build settings. Therefore, if you add the above line to the Build settings, you must position the line at the end of the settings.

Since the system sets extra default Cobol directives, the above line is required when putting directives into the Build Settings dialog from within Net Express.

For both project settings and Net Express Build settings, END-C and ENDP have the following effect:

- Directives placed before END-C pass to COBSQL
- Directives placed between END-C and ENDP pass via COBSQL to the precompiler
- Directives placed after ENDP pass to the COBOL compiler. Therefore, without the ENDP directive, compiler directives continue to pass to COBSQL rather than to the COBOL compiler.

11.2.1 Specifying Directives

You specify directives to COBSQL as if they were Compiler directives, but you must put them after the directive PREPROCESS"COBSQL".

It is also possible to add the Cobsql directives to the standard Net Express directives file **cobol.dir**.

Notes:

- Each line in the **cobol.dir** file may contain one or more compiler directives
- Avoid splitting a compiler directive across multiple lines in the **cobol.dir** file
- When the compiler encounters either P(COBSQL) or PREPROCESS(COBSQL) in the **cobol.dir** file, the compiler passes the rest of the line to the pre-processor until it reaches an ENDP
- The compiler treats the pre-process statement and all the options that follow, up to the ENDP, as a single compiler directive.

Therefore, the pre-process statement and all the options must all appear on a single line in the **cobol.dir** file.

Alternatively, you can put COBSQL and precompiler directives in a file, **cobsql.dir**. This file should reside either in the current directory or in a directory specified in $COBDIR. COBSQL searches the current directory and then along the COBDIR path for a **cobsql.dir** file. Once COBSQL finds a **cobsql.dir** file, it stops searching. So, if you have a **cobsql.dir** file in the current directory, the COBDIR path is not searched.

Notes:

- Each line in the file **cobsql.dir** can contain one or more COBSQL directives
- Since COBOL does not read the file **cobsql.dir**, avoid putting COBOL compiler directives into the file
- Avoid splitting a COBSQL directive across multiple lines in the file **cobsql.dir**
- When COBSQL encounters either END-C, END or END-COBSQL in the **cobsql.dir** file, the rest of the line passes to the database precompiler
- The options to be passed to the database precompiler must all appear on a single line in the file **cobsql.dir**

COBSQL processes **cobsql.dir** first and then any directives specified via the **Build Settings** screen.

A number of the directives can be reversed by placing NO in front of them, for example, DISPLAY can be reversed using NODISPLAY. All the directives in the lists below that can have NO placed in front of them are marked with an asterisk. By default, the NO version of a directive is set.

You can specify shortened versions of some of the directives. If applicable, the shortened version of a directive is shown in the lists below, immediately after the full length version.

Some directives can be passed to COBSQL by the COBOL Compiler (see the section *COBOL Directives*), removing the need to specify common directives more than once. Directives that can be retrieved from the COBOL Compiler are processed before COBSQL directives.

For example, in the following command line:

```
cobol testprog p(cobsql) csqlt=ora makesyn end-c
    comp5=yes mode=ansi endp omf(gnt) list();
```

- The COBSQL directives, terminated by `end-c`, are `csqlt=ora` and `makesyn`
- The precompiler directives (in this case, Pro*COBOL), terminated by `endp`, are `comp5=yes` and `mode=ansi`
- The Compiler directives are `omf(gnt)` and `list()`

11.2.2 COBSQL Directives

The following is a list of the COBSQL directives:

Directive	**Description**
COBSQLTYPE CSQLT	Specifies which precompiler to use (ORACLE, SYBASE or INFORMIX-NEW); for example, COBSQLTYPE=ORACLE .
CSTART* CST	Forces COBSQL to load the database support modules at execution time
CSTOP* CSP	Forces COBSQL to load the stop run module that performs a rollback if the application terminates abnormally
DEBUGFILE* DEB	Creates a debug (**.deb**) file
DISPLAY* DIS	Displays precompiler statistics. Should only be used when initially verifying that COBSQL is correctly calling the standalone precompiler.

END-COBSQL
END-C
END
	Signals the end of COBSQL directives; remaining directives, if any, are passed to the precompiler

KEEPCBL	Saves precompiled source file (**.cbl**)

Directive	Description
MAKESYN	Converts all COMP host variables to COMP-5 host variables. The default situation, if MAKESYN is not set, is that all variables (not just host variables) are converted from COMP to COMP-5.
NOMAKESYN	No conversion of COMP-5 variables or host variables is carried out
SQLDEBUG	Creates a number of files that can be used by Micro Focus to debug COBSQL. These files include the output file from the precompiler (normally this has a **.cbl** extension), the listing file produced by the precompiler (this has a **.lis** extension), plus a COBSQL debug file which has a **.sdb** extension. SQLDEBUG will also turn on KEEPCBL and TRACE.
TRACE*	Creates a trace file (**.trc**)
VERBOSE	Displays all precompiler messages and gives status updates as the program is processed. You should only use this when initially verifying that COBSQL is calling the standalone precompiler correctly.

11.2.3 COBOL Directives

The following is a list of the COBOL directives:

Directive	Description
BELL*	Controls whether COBSQL sounds the bell when an error occurs.
BRIEF*	Controls whether COBSQL shows SQL error text as well as the error number.
CONFIRM*	Displays accepted/rejected COBSQL directives.
LIST*	Saves the precompiler listing file (**.lis**).
WARNING*	Determines the lowest severity of SQL errors to report.

11.3 Building COBSQL Applications

It is beyond the scope of this document to list the database support modules that are required when shipping on a COBSQL application. It is assumed that the end-user machine already has all the required support modules installed and that it is correctly configured to communicate with the database server.

When linking COBSQL applications, use the import library **csqlsupp.lib**. This resolves the calls inserted by COBSQL to the COBSQL init and stop run modules. The calls are actually handled by the module **csqlsupp.dll** which needs to be shipped with the application. This general support module is required for both Oracle and Sybase applications.

Specify the library **csqlsupp.lib** as one of the libraries to use when linking the application. The library **csqlsupp.lib** resides in the directory **Net Express\base\lib**. The module **csqlsupp.dll** resides in the directory **Net Express\base\bin**.

If you split the application into a main exe and a number of sub DLL's, then you only need to link **csqlsupp.lib** into the modules that you have compiled with the COBSQL directives CSTART or CSTOP.

If you compile all programs with CSTART or CSTOP, then you must link **csqlsupp.lib** into all the modules. Linking **csqlsupp.lib** to each module causes each module to be slightly larger than required. Only one version of **csqlsupp.dll** is loaded when the application runs.

If you compile the main program only with CSTART and CSTOP, then you need to link in the library **csqlsupp.lib** with the main program only. Any program that you compile with either the CSTART or the CSTOP COBSQL directive, you then need to link **csqlsupp.lib** with the module for that program.

11.4 Using the CP Preprocessor to Expand Copyfiles

The complete set of methods used within COBOL to manipulate copyfiles is not available with database precompilers and COBSQL itself cannot handle included copyfiles. These problems can be overcome, however, by using the Micro Focus Copyfile Preprocessor (CP).

CP is a preprocessor that has been written to provide other preprocessors, such as COBSQL, with a mechanism for handling copyfiles. CP follows the same rules as the COBOL Compiler for handling copyfiles so any copyfile-related Compiler directives are automatically picked up and copyfiles are searched for using the COBCPY environment variable. CP will also expand:

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

statements. For more information on CP, refer to the Net Express online help (look under "CP" in the help file index).

Oracle uses **.pco** and **.cob** extensions, Sybase uses **.pco** and **.cbl** extensions and Informix uses **.eco**, **.cob** and **.mf2** extensions.

Oracle and Sybase: For CP to resolve copyfiles and include statements correctly, use the following COBOL Compiler directives for Sybase and Oracle:

```
copyext (pco,cbl,cpy,cob) osext(pco)
```

Informix: For Informix, use:

```
copyext (eco,mf2,cbl,cpy,cob) osext(eco)
```

COBSQL can call CP to expand copyfiles before the database precompiler is invoked. This means that all the copy-related commands are already resolved so that it appears to the database precompiler that a single source file is being used.

The other advantage of using CP is that it makes copyfiles visible when animating.

When CP sees an `INCLUDE SQLCA` statement, it does the following:

- Searches for a file called **sqlca.*ext*** in the current directory where ***ext*** is any copyfile extension as set up by the OSEXT and COPYEXT Compiler directives. By default these are **.cbl** and **.cpy**.
- Searches for **sqlca.*ext*** along the COBCPY path.
- An example **sqlca.cpy** file is provided in the **source** directory under your Net Express base installation directory. If the SQLCA file supplied by the database vendor is not located on the COBCPY path, this Micro Focus demonstration version of the SQLCA will be used.

Note: Using the file **sqlca.cpy** can result in errors when the program is run.

You can specify the CP preprocessor's SY directive to prevent CP expanding the SQLCA include file, for example:

```
preprocess"cobsql" preprocess"cp" sy endp
```

You should always use CP's SY directive when processing Sybase code because Sybase expects to expand the SQLCA itself.

As Oracle can produce code with either COMP or COMP-5 variables, it has two sets of copyfiles. The standard **sqlca.cob**, **oraca.cob** and **sqlda.cob** all have COMP data items. The **sqlca5.cob**, **oraca5.cob** and **sqlda5.cob** files have COMP-5 data items. If you are using the comp5=yes Oracle directive, you must set the COBSQL directive MAKESYN to convert the COMP items in the SQLCA to COMP-5.

If CP produces errors when attempting to locate copyfiles, check to make sure that the OSEXT and COPYEXT Compiler directives are set up correctly. COPYEXT should be set first and should include as its first entry the extension used for source files (**.pco** or **.eco**, for example).

If these are set correctly, ensure that the copyfile is either in the current directory or in a directory on the COBCPY path.

When using CP in conjunction with COBSQL, SQL errors inside included copyfiles will be reported correctly. Without CP, the line counts will be wrong, and the error will either go unreported or will appear on the wrong line.

11.5 National Language Support (NLS)

COBSQL error messages can be displayed in different languages depending on the setting of the LANG environment variable. For full details on NLS and how to set the LANG environment variable, look up **NLS** and **LANG** in the help file index.

Most database clients include some NLS capability but their requirements for the setting of the LANG environment variable differ from those of this COBOL system. We recommend, therefore, that you use the alternative environment variable, COBLANG.

The setting of COBLANG only affects this COBOL system, allowing the LANG environment variable to be used by the database client. Note that for COBLANG to work correctly, **mflang*nn*.lbr**, where *nn* is the setting of COBLANG, must be available in the Net Express **\bin** directory. So if COBLANG=05 (UK NLS messages), the file **mflang05.lbr** must be present in **Net Express\Base\Bin**.

The setting of COBLANG only affects the COBSQL error messages; error messages produced by the database precompiler are not translated by COBSQL.

11.6 Examples

The following examples show, for the Oracle, Sybase and Informix precompilers, command lines which can be entered at the Net Express Command Prompt to compile a program using COBSQL.

11.6.1 Oracle

```
cobol sample.pco anim nognt preprocess(cobsql)
   cstart cstop CSQLT=ORA end-c comp5=yes endp;
```

UNIX:

```
cob -a -v -k sample.pco
    -C "p(cobsql) cstop cobsqltype==ORACLE"
```

11.6.2 Sybase

```
cobol example1.pco confirm preprocess(cobsql)
   cstop csp cobsqltype=sybase preprocess(cp) sy endp;
```

UNIX:

```
cob -a -v -P -k example1.pco
    -C "p(cobsql) csp CSQLT==syb"
```

11.6.3 Informix

UNIX:

```
cob -a -k demo1.eco
    -C "p(cobsql) cobsqltype==informix-new"
```

11.7 Troubleshooting

Initially, you should check each of the items outlined below.

- Basic network connectivity

  Forget SQL, and determine whether the client and server are communicating. For TCP/IP, check whether you can ping the server from the workstation and vice versa. If host names don’t work, try raw IP addresses. For PC protocols, try mounting a network drive or sending messages.

- SQL networking software

  Check that the SQL networking software is "talking" correctly to the network software. Many SQL vendors supply a ping utility which will show whether the SQL network is set up correctly.

- Interactive SQL

  If the SQL network is okay, try some interactive SQL. Most vendors supply a simple utility that allows you to enter SQL from the keyboard and view the results. Most vendors also supply a sample database that is useful for this purpose.

- Standalone Precompiler

  Verify that the standalone precompiler works. There may be an icon or a command line for the precompiler. Verify that it can produce COBOL code correctly. It is normal for some sample applications to be supplied with the precompiler.

- Preprocessed application

  Check that a preprocessed application runs okay. Pass the expanded program through the COBOL Compiler and then try to run it.

- COBSQL with minimal directives

  Try COBSQL with minimal directives. Set up a project in Net Express, place the SQLCA copyfile into the directory with the sample program (prior to running the precompiler), and see if this works.

If you continue to have problems, please contact Micro Focus Support. To help Technical Support locate the cause of the problem:

- Use the COBSQL directive SQLDEBUG
- Send a zip file containing the following to Technical Support:
  - The original source
  - The expanded source (as produced by the database precompiler)
  - The trace file (which will have the extension **.trc**)
  - The database list file (which will normally have the extension **.lis**)
  - The command line debug file (which will have the extension **.sdb**)
  - The project file (which has the same name as the project but with the extension **.app**)
  - The COBSQL directive settings used

11.7.1 Common Problem Areas

If you cannot locate the source of the problem, check each of the following:

- Latest versions

  Ensure that you are using the latest version of all the products involved.

- COMP/COMP-5 Conflicts

  Check the vendor's documentation and example applications.

- Configuration

  Check that environment variables, PATH and configuration file settings are set up correctly.

- Directives

  By default, COBSQL does not display the command line it passes to the database precompiler. Setting the SQLDEBUG directive enables the command line to be displayed (you will need to do this if the precompiler gives command line errors). Possible causes of command line errors are that the directives to be passed to the precompiler are incorrect or that the length of the precompiler command line has been exceeded.

- Memory

  COBSQL may display the following error because the database precompiler has terminated unexpectedly:

  ```
  * CSQL-F-021: Precompiler did not complete -- Terminating
  ```

  This may be because the Operating System has run out of memory attempting to execute the database precompiler.

- Missing output files

  COBSQL may display the following errors because it cannot find the precompiler's output file. This may be because the precompiler did not produce an output file. The normal reason for this is that the precompiler hit a fatal error which meant it could not create the output file.

  ```
  * CSQL-E-024: Encountered an I/O on file filename
  * CSQL-E-023: File Status  3 / 5
  ```

where *filename* is the name of the file produced by the database precompiler.

- Premature end of expanded source

  If COBSQL reports the error "Premature end of expanded source" and the precompiler runs correctly, this indicates that COBSQL has not been able to match the original source lines with the lines produced by the database precompiler.

  Another possible reason for COBSQL reporting this error is that the program does not contain any SQL. Generally, if the database precompiler does not come across any SQL it will abort the creation of its output file part way through, causing this error to be displayed.

11.7.2 Oracle Considerations

You can use Oracle Pro*COBOL 1.8 or Oracle Pro*COBOL 8.x. The following sections describe the items to consider for each of these versions.

*11.7.2.1 Oracle Pro*COBOL 1.8 Considerations*

- The following Oracle precompiler options can have a marked effect on the behaviour and the memory requirements of an Oracle application. We recommend that you take time to review your Oracle documentation before changing the setting of these directives.

  ```
  DBMS
  HOLD_CURSOR
  MAXOPENCURSORS
  MODE
  RELEASE_CURSOR
  ```

- Within an Oracle program it is possible to use the ALTER SESSION command to change the OPTIMIZER_GOAL. This can have a profound effect on the performance of the application. Before using this syntax, we recommend that you consider what is the best optimisation for an application in the general case rather than for a particular SQL statement. If you need to change the performance of one particular statement, it is probably better to use HINTS. Consult your Oracle documentation for further information on ALTER SESSION, OPTIMIZER_GOAL and HINTS.

- Oracle provides a method of reducing network access by using arrays within Embedded SQL statements. This enables the application to perform 'batch' SQL commands, in that more than one row can be acted upon in a single SQL statement. See the section *Host Arrays* in the chapter *Host Variables* for more details.

  The use of arrays enables an application, for example, to fetch ten rows at a time instead of one at a time. Oracle supply an example program (normally called **sample3.pco**) that uses an array to fetch multiple rows. Arrays are documented in the ***Pro*Cobol Supplement to the ORACLE Precompilers Guide***.

- Pro*Cobol treats BINARY host variables as though they had been defined with a picture clause of COMP. This means that BINARY items will also be converted to COMP-5 when the Pro*Cobol comp5=yes option is used.

- Pro*Cobol always needs to be supplied with an include directive, unless the SQLCA copyfile is in the current directory, because it needs to know the location of the SQLCA copyfile. If this is not supplied, Pro*Cobol gives an error on *filename*.pco, where *filename* can be any string of eight characters.

- When Pro*Cobol runs, it can generate useful information. When the COBSQL VERBOSE directive is used, COBSQL displays as much of this information as it can.

  To get the maximum information from Pro*Cobol, set the Pro*Cobol directive xref=yes. You can add this directive to the Pro*Cobol configuration file ***$ORACLE_HOME*\PRO*xx*\pcbcfg.cfg** where:

xx	is the Pro*COBOL version (for example, for Oracle 8.0 this is PRO80)
$ORACLE_HOME	is the root directory for the Oracle installation on your machine

- By using the COBSQL DISPLAY directive, the current Pro*Cobol directives can be seen on the screen. This will be followed by the statistical information created by Pro*Cobol.

- If the COBOL LIST directive is used, COBSQL passes any information it has collected from the precompiler to the COBOL checker for inclusion at the the bottom of the COBOL listing.

11.7.3 Oracle Pro*COBOL 8.x Considerations

Support for Pro*COBOL 8.0 has been added to COBSQL which now works correctly with the Pro*COBOL 8.0 4.0 precompiler.

11.7.3.1 Directives

To use COBSQL with Oracle 8, you should use the following directives:

Directive	**Description**
CBL2ORA8 C28	This puts calls into the Oracle 8 specific support modules **ora8prot** and **ora8lib**. Both of these modules are built into **csqlsupp.dll**.
COBSQLTYPE CSQLT	EXEC SQL preprocessor. Use the options ORACLE8 and ORA8 to use Pro*COBOL 8.x with COBSQL.

11.7.3.2 Directives

If you are migrating programs from Pro*COBOL 1.x to 8.x, you should be aware of the following:

- Within the Data Division:
  - You should terminate all SQL statements with a period
  - Only ANSI comments are supported. Comments starting with *> in any column are not supported
- Support for nested programs

  Define all inserted variables as GLOBAL including the data items inserted by COBSQL that support the EBCDIC to ASCII conversions.

- Oracle 8 no longer requires a declare section as this can cause problems with the Oracle COMP to COMP-5 conversion.

  If the Oracle directive DECLARE_SECTION=NO is set (the default), Oracle converts all COMP, BINARY or COMP-4 data items to COMP-5.

  To limit the conversion of items to the declare section, set:

  - DECLARE_SECTION=YES or MODE=ANSI

    or

  - The Pro*COBOL directive COMP5=NO and the COBSQL directive MAKESYN

- Extra data types supported by Pro*COBOL 8.x are:

  - `PACKED-DECIMAL`

    These data items are treated in the same way as `COMP-3` data items.

  - `COMP-4`
  - `PIC 9(4) COMP / COMP-4 / BINARY / COMP-5`
  - `PIC 9(4) USAGE DISPLAY`
  - `PIC s9(4) USAGE DISPLAY SIGN TRAILING`
  - `PIC s9(4) USAGE DISPLAY SIGN TRAILING SEPARATE`
  - `PIC s9(4) USAGE DISPLAY SIGN LEADING`
  - `PIC s9(4) USAGE DISPLAY SIGN LEADING SEPARATE`

    This type was previously supported as the Oracle `DISPLAY` datatype.

11.7.3.3 Micro Focus COBOL

Some Micro Focus COBOL language extensions, data definitions and section headings are rejected by Pro*COBOL 8.x:

- All pointer data types
- 88 level variables
- Constants in a Value clause

- Items defined as external-form
- Local-Storage Section
- Thread-Local-Storage Section

To overcome this you need to put these items into copybooks which are not opened by Pro*COBOL. However, this does not work if you use CP which expands copybooks before Pro*COBOL is invoked. This could cause a problem if you are using **htmlpp** which calls CP to expand copybooks. You must therefore invoke **htmlpp** before COBSQL.

For example, the following compile line works:

```
COBOL PROG P(HTMLPP) PREPROCESS(COBSQL) CSQLT=ORACLE8
```

whereas this line does not:

```
COBOL PROG PREPROCESS(COBSQL) CSQLT=ORACLE8 P(HTMLPP)
```

You must define at least one variable within the Working-Storage Section for Pro*COBOL 8.0.4 to add its variables to the generated **.cbl** file.

11.7.4 Sybase Considerations

- If CP is not used with COBSQL to locate copyfiles, the copyfile to be brought in by the Sybase precompiler needs to be fully qualified. The Sybase precompiler does not search for any extensions.
- If the Sybase precompiler has problems locating the correct language to use to report messages on the client, check that the Sybase file, **locales.dat** is configured correctly.

  If the default setting for the client Operating System has been configured, but Sybase still reports national language support errors, use the LANG environment variable to override the setting in the **locales.dat** file.

For example if the Windows NT client was causing problems and the **locales.dat** file contained the following setting for Windows NT:

```
[NT]
locale = default, us_english, iso_1
locale = enu, us_english, iso_1
locale = fra, french, iso_1
locale = deu, german, iso_1
```

then the LANG setting for English would be:

```
LANG=enu
```

- It can be difficult to identify Sybase error messages. To help COBSQL identify them, you can modify the **esql.loc** file. Use a text editor to edit the **esql.loc** file to change the layout of the messages to the following:

  `SYB-`*`number-type-text`*

  where the parameters are:

  SYB- A string which indicates to COBSQL that this is a modified Sybase error message.

  number- A unique, four digit error number assigned to the Sybase error.

  type- Indicates the severity of the errror; some of the Sybase messages are only warnings rather than normal or fatal errors.

  text The original Sybase error message.

  For example, a typical entry in the **esql.loc** file might be:

  ```
  9 = M_PRECLINE, "Warning(s) during check of query on line
  %1!."
  ```

  and this would be changed to read:

  ```
  9 = M_PRECLINE, "SYB-W-2009 Warning(s) during check of
  query on line %1!."
  ```

  It is recommended that you make a copy of **esql.loc** before altering it. Using the modified version, COBSQL can detect the full range of Sybase error messages.

The location of **esql.loc** is dependent on the language and code page used. This is defined in the **locales.dat** file. If the definition of the default language for the AIX platform was as follows:

```
[aix]
locale = C, us_english, iso_1
locale = En_US, us_english, iso_1
locale = en_US, us_english, iso_1
locale = default, us_english, iso_1
```

The default langauge would be us_english, using the iso_1 code page, so the copy of **esql.loc** that is to be used is:

```
/sybase home/locales/messages/us_english/iso_1/esql.loc
```

where *sybase home* is the directory that the Sybase client is installed into.

For more information on how Sybase uses and locates the different error message files, refer to your Sybase ***Client Reference Manual***.

- To use Sybping with different Sybase servers, each server should have a service defined. The Sybase product sqledit shows you how to enter the required information.
- If the name of the server is not specified when connecting to Sybase, Sybase searches for an environment variable called DSQUERY. This can be set within the environment variable section of a project. Doing this enables different projects to connect to different Sybase servers.
- When the Sybase precompiler runs, it can generate useful information. If the COBSQL directive VERBOSE is set, COBSQL displays as much of this information as it can.
- By using the COBSQL DISPLAY directive, the statistical information created by the Sybase precompiler can be seen on the screen.
- If the COBOL LIST directive is set, COBSQL passes any information it has collected from the precompiler to the COBOL checker for inclusion at the end of the COBOL listing.
- The Sybase precompiler accepts programs that do not contain any SQL statements, and produces an output file. This means that all of the programs in a project can be compiled successfully. However, it also means that the Sybase precompiler inserts all the Working-Storage items and COBOL support code that it would insert for a program that contains Sybase SQL statements into a non-Sybase

program. This makes the program larger, and may effect its performance.

- With Sybase System 11 on Windows NT, the Sybase COBOL precompiler can be supplied separately to the Sybase System 11 Server. If this is the case, care must be taken when installing the Sybase COBOL precompiler as it is supplied with its own version of the Open Client support files and problems can arise if the two products have different versions of these files. In this situation, we strongly recommend that you contact Sybase support before proceeding with the installation of the Sybase System 11 COBOL precompiler.
- You should take particular care if you are installing the Sybase client onto a machine that has the Server installed onto it (or vice versa). Installing the Sybase client only does not cause any problems.

11.7.5 Informix Considerations

- Informix uses the eco, cob and mf2 file extensions. For CP to resolve copybooks and include statements correctly, use the following COBOL Compiler directives:

  ```
  copyext(eco,mf2,cob,cpy,cbl) osext(eco)
  ```

- Under UNIX, because COBSQL invokes the Informix precompiler, the INFORMIXCOB, INFORMIXCOBTYPE and INFORMIXCOBDIR environment variables all need to be setup before using COBSQL. For more information about these environment variables, refer to the ***Informix COBOL/ESQL Programmers Guide***.
- Informix only produces error messages in its list files. Normally the error message will contain the line the error occured on. If COBSQL cannot locate the line number from the error message, it will not report the error.
- To run COBSQL correctly with Informix, the INFORMIXDIR environment variable should be set before using COBSQL.

Index

Database Access