User's Guide

AcuSQL®

Version 8.1

Micro Focus
9920 Pacific Heights Blvd.
San Diego, CA 92121
858.790.1900

E-01-UG-080901-AcuSQL-8.1

Contents

Chapter 1: Getting Started

1.1 Introduction to AcuSQL 1-2
1.2 AcuSQL Pre-compiler 1-2
1.3 AcuSQL Runtime Library 1-3
1.4 Requirements 1-4
1.5 Installation 1-7
  1.5.1 Installation Under Windows 1-7
  1.5.2 Installation Under UNIX 1-8
1.6 Organization 1-11
1.7 Demonstration Programs 1-12
1.8 Technical Services 1-12

Chapter 2: Program Preparation

2.1 Preparing Your Programs 2-2
2.2 Coding Considerations 2-4
  2.2.1 User-Supplied Object Names 2-4
  2.2.2 Coding Area 2-4
  2.2.3 Commas 2-5
  2.2.4 String Delimiters 2-5
2.3 Data Division 2-5
  2.3.1 Cursors 2-6
  2.3.2 SQL INCLUDE files 2-6
  2.3.3 Host Variables 2-8
2.4 Data Types 2-9
  2.4.1 DB2 Data Type Compatibility 2-9
  2.4.2 Microsoft SQL Server Data Type Compatibility 2-10
  2.4.3 Access Data Type Compatibility 2-11
  2.4.4 MySQL Data Type Compatibility 2-12
2.5 Procedure Division 2-13
  2.5.1 Cursors 2-13
  2.5.2 Rowset Functions 2-14
  2.5.3 Stored Procedures 2-15
    2.5.3.1 Restrictions 2-17

2.6 SQL Verbs....2-17
2.6.1 CONNECT Statement....2-18
2.6.2 DISCONNECT Statement....2-19
2.7 Checking Syntax....2-19
2.8 Error Handling....2-20
2.9 Error Messages....2-21
2.10 Limits and Restrictions....2-23
2.11 Detecting and Handling NULL Values....2-23

Chapter 3: Pre-compiler Function and Use

3.1 The ESQL Translation Process....3-2
3.2 Using the Pre-compiler....3-3
3.2.1 Using AcuSQL From Within AcuBench....3-5
3.2.2 Using AcuSQL as a Standalone Program....3-5
3.2.3 Using AcuSQL from the Compiler....3-9
3.3 Environment Variables....3-12
ACUSQL_DATASET....3-13
ACUSQL_INCLUDE....3-13
ACUSQL_PASSWORD....3-13
ACUSQL_USER....3-14
3.4 Pre-compilation Errors....3-14

Chapter 4: Program Execution

4.1 Running Your Application....4-2
4.2 Running Your Application with Microsoft SQL Server....4-2
4.2.1 Runtime Configuration Variables for SQL Server....4-4
ASQL_BUFFER_SIZE....4-4
ASQL_CONNECT_DATABASE....4-4
ASQL_NULL_ALPHA_SPACES....4-5
ACUSQL_SQLSTATE_2000_ON_EOD....4-5
4.2.2 Runtime Configuration Variables for esqllib....4-5
ACUSQL_NO_AUTOCOMMIT....4-6
ACUSQL_ODBC_CURSORS....4-6
ACUSQL_USE_CONCURRENT....4-6
ODBC_CURSOR_TYPE....4-7
4.3 Debugging and File Tracing....4-7
4.4 Error Messages....4-8

Chapter 5: Sample Programs

5.1 Sample Programs 5-2
5.2 Static SQL 5-3
5.3 Creating Tables With ESQL - create.sqb 5-3
5.3.1 WHENEVER Directive 5-4
5.3.2 Using DDL 5-5
5.3.3 INSERT Statement 5-6
5.3.4 Date Format 5-7
5.3.5 Putting It All Together 5-7
5.4 Using Working-Storage Items – select1.sqb 5-8
5.4.1 SELECT Statement 5-9
5.4.2 Host Variables 5-10
5.4.3 INTO Clause 5-10
5.4.4 WHERE Clause 5-12
5.4.5 MIN and MAX Group Functions 5-12
5.4.6 Putting It All Together 5-13
5.5 Group Items in the INTO Clause – select2.sqb 5-14
5.6 Working With More Than One Row – select3.sqb 5-15
5.6.1 SET ROWCOUNT Statement 5-16
5.6.2 Cursors 5-16
5.6.3 Declaring a Cursor 5-16
5.6.4 Opening a Cursor 5-17
5.6.5 FETCH Statement 5-17
5.6.6 Closing a Cursor 5-18
5.6.7 Putting It All Together 5-19
5.7 Updating Data – update.sqb 5-20
5.7.1 FOR UPDATE Clause 5-21
5.7.2 The SET Clause 5-21
5.7.3 Positioning the Cursor for the UPDATE 5-22
5.7.4 Putting It All Together 5-23
5.8 Full SQL Program 5-25

Glossary of Terms

Index

1 Getting Started

Key Topics

Introduction to AcuSQL 1-2
AcuSQL Pre-compiler 1-2
AcuSQL Runtime Library 1-3
Requirements 1-4
Installation 1-7
Organization 1-11
Demonstration Programs 1-12
Technical Services 1-12

1.1 Introduction to AcuSQL

Welcome to the AcuSQL® pre-compiler and runtime library. This add-on tool to the ACUCOBOL-GT® development system is a simple, cost-effective solution for those who want or need to use *embedded* SQL (ESQL) statements in COBOL programs to access SQL-conversant data sources. AcuSQL is part of the ***extend***® family of technologies and is designed to give COBOL applications access to IBM® DB2®, Microsoft® SQL Server, and ISO/ANSI SQL92 compliant data sources. AcuSQL also supports the open source database, MySQL®.

AcuSQL is available for 32-bit Windows® operating systems (Windows Vista, Windows XP, Windows NT® 4.0, Windows 2000) 64-bit versions of the Windows operating system (Windows Server 2003, 2008 x64, Vista x64) and for most UNIX® operating system platforms, including 64-bit environments.

The AcuSQL pre-compiler is a powerful programming tool that allows you to embed SQL statements in your COBOL application. ESQL enables you to access the power and flexibility of Structured Query Language (SQL) from within your COBOL application. With AcuSQL, you can execute complex data selection and processing operations using the database engine to perform most of the work, greatly simplifying the COBOL code needed to perform the operation. AcuSQL supports both static and dynamic SQL.

AcuSQL support of native dialects of SQL (DB2, MySQL, SQL Server), ISO/ANSI SQL92, and a “relaxed” version of SQL enables you to write applications custom-tailored to a wide variety of individual databases or a more generic version able to run across multiple databases and data sources.

AcuSQL support for ESQL is provided by a special *pre-compiler* and runtime library.

1.2 AcuSQL Pre-compiler

The pre-compiler gives you the option of specifying various levels of SQL syntax checking, including modes that are specific to IBM DB2 SQL, MySQL, Microsoft SQL Server, and a mode that directs the pre-compiler to send SQL statements directly to the database engine for syntax verification. These syntax checking options give you a great deal of flexibility in developing your ESQL applications. In addition, you can stipulate relaxed or strict checking for certain individual commands in your program.

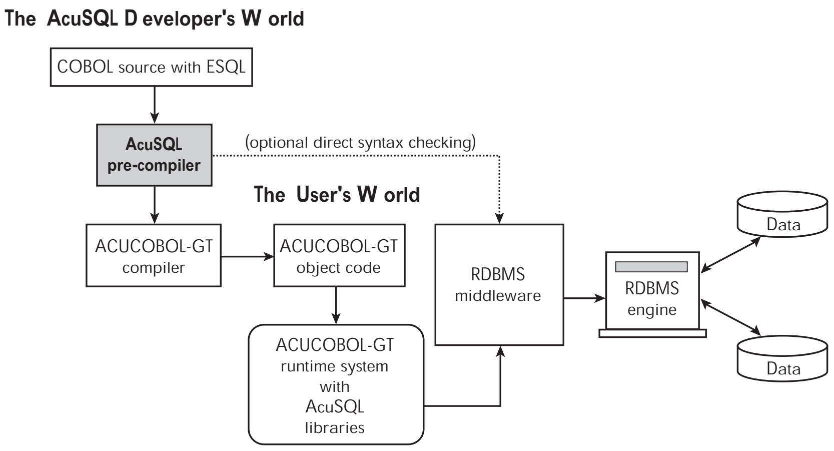
The AcuSQL pre-compiler works by scanning both the COBOL source file and all copy files for ESQL statements. It preserves the original statements in a comment block, performs syntax checking (either directly or indirectly), and then translates each statement into a standard COBOL CALL statement to the AcuSQL runtime library. All other code is untouched.

Although pre-compiling adds an additional step to the development process, a great deal of time and effort are saved by not requiring you to write database-specific API calls in your application. A single SQL statement in your COBOL program may translate to many API calls in the generated AcuSQL code. Embedded SQL is also easier to write and maintain than the corresponding API calls.

Pre-compilation is initiated from within the AcuBench® development environment or from the ACUCOBOL-GT command line. In AcuBench, when you set the “pre-compile” option on an ESQL file, the workbench automatically pre-compiles the file every time the project is *built* or the file is compiled. All pre-compilation errors are displayed in the AcuBench *Output Window*. On the ACUCOBOL-GT command line a set of compiler flags initiates pre-compilation. Error messages are directed to *standard output*.

1.3 AcuSQL Runtime Library

Successful pre-compilation followed by successful compilation results in the creation of an *object* file. The object file is ready for execution by the runtime. When the object file is executed, the embedded SQL statements (now standard COBOL statements) are passed to the AcuSQL library, which in turn communicates with the database connectivity software (an ODBC level 2 API), establishing connections, retrieving and writing data, and exchanging messages.



The AcuSQL development and deployment model

1.4 Requirements

To use AcuSQL, your environment must satisfy the following requirements:

- Your COBOL application must run on a Windows system or a UNIX system supported by Micro Focus. Unless otherwise indicated, the references to “Windows” in this manual denote the following 32-bit versions of the Windows operating systems: Windows Vista, Windows XP, Windows NT 4.0 or later, Windows 2000, Windows 2003; and the following 64-bit versions of the Windows operating system: Windows Server 2003 and 2008 x64, Vista x64. In those instances where it is necessary to make a distinction among the individual versions of those operating systems, we refer to them by their specific version numbers (“Windows 2000,” “Windows NT 4.0,” etc.).

- AcuSQL must be installed, along with the ACUCOBOL-GT development system, on your Windows or UNIX system.

- *If you are working with a database other than Microsoft SQL Server,* you must have a working ODBC level 2 API connection to your database, including any required networking software support.

It is up to you to procure and configure this software and any related network software.

Your data source may be located on any host for which you have a working ODBC level 2 API connection.

SQL Server Requirements

If you are running the AcuSQL interface to Microsoft SQL Server, you must have the SQL Server client software from Microsoft.

To see if the SQL Server client software from Microsoft is on your system, Open the Query Analyzer. (For information on opening the Query Analyzer see SQL Server client documentation.)

If the Query Analyzer opens and you are able to connect to the database, the client libraries are most likely all present.Your SQL Server data source may be hosted on one (or more) of the following environments:

- Windows NT Server
- Windows 2000, 2003, 2008 Server

DB2 Requirements

If your application is going to access DB2 data, we recommend IBM's DB2 Connect™ software. At Micro Focus, access to DB2 databases has been tested with DB2 Connect. However, any vendor's properly configured ODBC level 2 API connectivity software should work.

Your DB2 data source may be hosted on one (or more) of the following environments:

- OS/390®
- AS/400®

- Windows NT
- Windows 2000, 2003, 2008
- HP-UX
- Solaris™
- AIX®

MySQL Requirements

If you are running the AcuSQL interface to MySQL, you must have the following software:

- MySQL 5.0 Database Server Version 5.0.18 or later (Generally Available release). Testing was done with MySQL 5.0.18 Standard.
- MySQL Connector/ODBC Version 3.51.11 or later (Generally Available release). Testing was done with the “libmyodbc3-3.51.12.so” library. This file is available from http://dev.mysql.com/.

You can check the version of your server by connecting using “mysql”. The version of the server will be printed upon connection. For example :

```
[testing ]: mysql
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 29 to server version:
5.0.18-standard
Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
```

Once in MySQL, you can also use the following:

```
mysql> select version();
+-----------------+
| version()       |
+-----------------+
| 5.0.18-standard |
+-----------------+
1 row in set (0.09 sec)
```

1.5 Installation

Before installing AcuSQL, you should install, configure, and test your application-host to data-host connection software. For accessing DB2 databases, AcuSQL has been tested with IBM DB2 Connect software, and we recommend its use. However, any vendor's properly installed and configured ODBC level 2 API connection is acceptable. This *middleware* connection software is essential to AcuSQL's successful use. *You must ensure that your connection middleware is functioning properly before attempting to run an AcuSQL-supported application.*

1.5.1 Installation Under Windows

AcuSQL is installed using Windows Installer. To install AcuSQL on your Windows host:

1. Log onto your Windows system.

   Under Windows NT or Windows 2000, to ensure that all files are installed with the correct permissions, we recommend that you log on using the Administrator account or an account that belongs to the Administrators group. We also recommend that you close all applications running on the system.

2. Insert the installation CD-ROM into your disk drive. If the installation program does not start automatically, click **Start**, select **Run**, and then enter:

   ```
   D:\autorun.exe
   ```

   replacing "D" with the device designation of your CD-ROM drive. Follow the instructions on the screen, entering your product code and product key when prompted.

3. When the Products Selection screen appears, select **Custom** from the list and click **Next**.

4. Select **AcuSQL** from the product list, and if desired, **Online Documentation**. Before clicking **Next**, take special notice of the default installation directory appearing near the bottom of the window. Click **Browse** to designate a different directory. If you specify a

different directory, a subdirectory named with the release version (e.g. Acucbl800) is not created. AcuSQL is installed in the directory you specify.

5. Click **Next** to proceed with the installation and follow the instructions on the screen. The program will inform you when installation is complete.

There is nothing on the AcuSQL distribution media to install on your data source host machines.

On the Start menu under "Programs/Acucorp 8.*x.x*/AcuSQL" (or the name of the program group you specified), you will find entries for the AcuSQL Release Notes file. To view the online help for AcuSQL, select **extend8 Documentation Set** from the Start/Programs/Acucorp 8.*x.x* menu. This opens a document containing a listing of available documentation. Then select the *AcuSQL User's Guide* from this listing.

1.5.2 Installation Under UNIX

Note: In the following directions, the term "runtime system" refers to the runtime shared object on systems where the ACUCOBOL-GT runtime is a shared object and to **runcbl** on other systems, where the runtime is static. The runtime is a shared object on the following systems: AIX 5.1 and later, HP/UX 11 and later, and Solaris 7 and later. To check, look at the contents of the "lib" subdirectory of your ACUCOBOL-GT installation. If the files "libruncbl.so" or "libruncbl.sl" reside in that directory, the runtime is a shared object on your system.

To install AcuSQL on your UNIX host:

1. We recommend that you install AcuSQL into your ACUCOBOL-GT home directory. When you install in this way, the AcuSQL libraries and executables are added to the existing ACUCOBOL-GT "bin" and "lib" subdirectories. If you choose to install into a different directory, separate "bin" and "lib" subdirectories are created. You must include the location of the new "bin" directory in the definition of your "path" environment

variable. No matter which directory you select for your installation, you must make sure that AcuSQL and the ACUCOBOL-GT compiler executable (**ccbl**) are in the same subdirectory.

2. Insert the installation CD-ROM, floppy disk, or tape into its corresponding device. The instructions for installing from CD-ROM are provided on a *Getting Started* card supplied with the CD-ROM. If you are installing from floppy or tape, change ("cd") into the target directory and extract the AcuSQL software.

   Because the software is in TAR format, you should use a command similar to the following:

   ```
   tar xfv device_name
   ```

   where *device_name* is the appropriate hardware device on your system (for example, "/dev/rfd0" or "/dev/rmt0"). Follow the instructions on the screen, entering your product code and product key when prompted.

3. Relink the runtime system to include your ODBC API library.

   Note: Relinking the runtime requires access to the C compiler supplied by the vendor of your UNIX software; in some cases the C compiler is offered as an add-on option. If the C compiler used to build the ODBC libraries creates files that are incompatible with those created by the C compiler used to build the runtime, the link may fail.

   a. In the "lib" subdirectory (of the directory into which you installed AcuSQL), edit "config85.c" and set the value of "NO_ACUSQL" to "0".

   b. Edit "Makefile" and locate the line that reads:

      ```
      ACUSQL_ODBC_LIB =
      ```

      Add the name of your ODBC API library to the end of the line. For example:

      ```
      ACUSQL_ODBC_LIB = ODBC_lib
      ```

      where *ODBC_lib* is the name of your ODBC API library. If you are working with DB2, the name of the library is "libdb.so". If you are working with MySQL, the name of the library is "libmyodbc3".

If this library is not statically linked, your operating system must be able to locate this library. On some systems, this is accomplished by listing the directory in the environment variable LD_LIBRARY_PATH. (See your operating system documentation for more information on shared libraries.)

c. Also in “Makefile”, locate the lines:

```
ACUSQL_FLAGS = -DNO_ACUSQL=1
ACUSQL_LIBS = # no acusql runtime libraries are   necessary
```

d. Make sure that the above lines are commented out (use the comment character “#”).

e. Next, locate the lines:

```
#ACUSQL_FLAGS =
#ACUSQL_LIBS = $(ACU_LIBDIR)/libesql.a   $(ACUSQL_ODBC_LIB)
```

Make sure that the above lines are un-commented (remove the comment character “#”).

f. Save and close “Makefile”.

g. Relink the runtime system by entering the following commands:

```
make clean
make
```

The default target of the Makefile is the runtime system, so this relinks **runcbl** on machines that do not have shared objects, and **libruncbl.so** on those that do.

You can also relink the runtime system by executing the following command:

```
make relinkrun
```

This works on all systems.

If you get unresolved reference errors that refer to your ODBC library, you will need to further modify the “Makefile” to include the needed libraries (see your ODBC API documentation). Such libraries can be added to the end of the LDFLAGS lines, or at the end of the list for “runcbl” and “acusql”.

h. If the runtime system is a statically linked **runcbl**, move the new executable file into the “bin” directory. This step is not necessary when the runtime system is a shared library. In these cases, the shared object runtime, **lib/libruncbl.so**, is already in the correct location.

4. If you plan to use the pre-compiler mode that connects directly to your database engine to verify the SQL syntax, you must relink **acusql** to include your ODBC API libraries.

   a. If you haven’t already done so, perform steps “a” and “b” of item 3 above.

   b. Relink the pre-compiler by entering the following command:

   ```
   make acusql
   ```

   c. Move the new **acusql** executable file into the “bin” directory.

1.6 Organization

This book comprises five chapters:

Chapter 1, “Getting Started,” provides an overview of the AcuSQL product. It also provides installation information.

Chapter 2, “Program Preparation,” describes how to prepare your embedded SQL program, including an introduction to SQL syntax.

Chapter 3, “Pre-compiler Function and Use,” discusses the pre-compiler, including usage, environment variables, and errors.

Chapter 4, “Program Execution,” discusses running your application.

Chapter 5, “Sample Programs” describes five of the sample programs included with AcuSQL. In addition, several SQL commands and concepts are introduced.

1.7 Demonstration Programs

Included with your AcuSQL software are several ESQL demonstration programs. These programs provide working examples of ESQL COBOL program code that connects to DB2, Access, MySQL, and Oracle databases. The source files are located in the "sample/acusql" subdirectory where ACUCOBOL-GT is installed. A different directory is provided for each database. Program source files have the three-character suffix ".sqb". For those using AcuBench, an AcuSQL demonstration program is included in the sample project "Samprj".

Chapter 5, "Sample Programs," provides a description of five of these programs and uses them to highlight examples of SQL syntax. We recommend that you acquaint yourself with these programs and, if you have the underlying middleware support, that you pre-compile, compile, and run them (the DB2 program requires a Windows-to-DB2 ODBC data connection; the Access and Oracle programs require an ODBC data source). The program code includes examples of how to establish connections and how to access and manipulate data.

1.8 Technical Services

You can reach Technical Services in the United States Monday through Friday from 6:00 a.m. to 5:00 p.m. Pacific time, excluding holidays. You can also raise and manage product issues online and follow the progress of the issue or post additional information directly through the website. Following is our contact information:

Phone:	+1 858.795.1902
Phone:	800.399.7220 (in the USA and Canada)
Fax:	+1 858.795.1965
E-mail:	support@microfocus.com
Online:	http://supportline.microfocus.com

For worldwide technical support information, please visit http://supportline.microfocus.com.

2 Program Preparation

Key Topics

Preparing Your Programs 2-2
Coding Considerations 2-4
Data Division.. 2-5
Data Types... 2-9
Procedure Division 2-13
SQL Verbs 2-17
Checking Syntax 2-19
Error Handling... 2-20
Error Messages.. 2-21
Limits and Restrictions.. 2-22
Detecting and Handling NULL Values 2-23

2.1 Preparing Your Programs

Preparing your ESQL source code for use with the pre-compiler is straightforward. All ESQL statement blocks must start with the keywords “EXEC SQL” and end with the keyword “END-EXEC”. The structure and syntax of the SQL statements within each code block should conform to the variant of SQL required by the target database.

Note that the pre-compiler mode used to perform syntax checking can create variability with respect to errors encountered at runtime and, therefore, should be taken into consideration when you prepare your source code. For example, if you direct the pre-compiler to pass ESQL statements to the database engine for syntax checking (provided that your ODBC driver supports the return of syntax errors), you will know exactly what is acceptable and unacceptable to the database engine. This method ensures that your SQL statements are syntactically correct when the source code successfully pre-compiles. If, instead, you use the pre-compiler’s *relaxed* syntax checking mode (by specifying the “-Pr” option at compile time), some of your ESQL statements will pass through the pre-compiler without any test against the syntax accepted by the target database, increasing the likelihood of undetected SQL syntax errors causing problems at run time. For a description of the “-Pr” option, see section 3.2.2, “Using AcuSQL as a Standalone Program.”

Note: The AcuSQL® precompiler does not currently support any *large object* data types (LOBs). If your existing applications include LOBs, contact our Technical Service personnel for assistance in evaluating the steps you may need to take to make the best use of AcuSQL.

Also note: This manual provides only a brief treatment of the rules and application of SQL and ESQL. It is up to the programmers to know SQL and ESQL and to have a comprehensive SQL/ESQL reference manual at their disposal**.** Programmers will benefit from studying the special provisions of ESQL because they offer some supports for the programmatic application of SQL that are not included in interactive SQL.

By default, the pre-compiler performs syntax checking for compliance with the ISO/ANSI SQL 92 standard. For complete information about pre-compiler options, see section 3.2, “Using the Pre-compiler.”

Working with DB2 databases

If your program will access a DB2 database, your ESQL statements should conform to the IBM SQL standard described in IBM publication S10J-8158-00, "Embedded SQL Programming Guide, Version 5." A reference level definition of IBM SQL is included in IBM publication S10J-8165, "SQL Reference." When you pre-compile your program, specify the "-Pk DB2" option to direct the pre-compiler to perform DB2-specific syntax checking, or use the "-Pc" option to have the DB2 engine perform direct validation.

Working with Microsoft SQL Server databases

If your program will access a Microsoft SQL Server database, specify the "-Pk mssql" option to direct the pre-compiler to perform SQL Server-specific syntax checking.

In order to accommodate SQL Server function syntax (db_name(), for example), AcuSQL also supports SELECT statements that do not include "FROM … tablelist". Note that if this is the only SQL Server-specific syntax your program contains, you do not need to use the "-Pk mssql" option on your command line.

AcuSQL supports the following SQL Server syntax for embedded transactions:

- BEGIN TRANSACTION *name*
- COMMIT TRANSACTION *name*
- ROLLBACK TRANSACTION *name*

The *name* will be passed to SQL Server as is, and must match the rules for embedded transaction names. For information on configuration variables supported by AcuSQL in a SQL Server environment, see section 4.2.1, "Runtime Configuration Variables for SQL Server." For additional information on running applications with AcuSQL in a SQL Server environment, see section 4.2, "Running Your Application with Microsoft SQL Server."

In addition, it is possible to nest transactions, as supported by SQL Server. When you use nested transactions, note how many transactions you BEGIN, because you must COMMIT the same number of transactions for the transaction to be considered complete. For example, if you execute three BEGIN TRANSACTION statements, you must later execute three COMMIT TRANSACTION or COMMIT WORK statements to commit the work you have done.

Note, however, that a single ROLLBACK rolls back all nested transactions. This behavior is a feature of SQL Server, and is not controlled by the AcuSQL interface.

2.2 Coding Considerations

This section describes several points to keep in mind as your write your ESQL program:

2.2.1 User-Supplied Object Names

AcuSQL is case sensitive when reading the names of objects in SQL supplied by the user, such as connection names or cursor names. For example, AcuSQL reads “mycursor”, “MyCursor”, and “MYCURSOR” as the names of three different cursors. Therefore, be consistent in your use of uppercase and lowercase letters when referring to these objects.

2.2.2 Coding Area

Code EXEC SQL statements in columns 12 – 72 unless you are using Terminal mode input. You can indicate the format of a source file with the “-Sa” and “-St” compile options. For more information on these options, see section 3.2.2, “Using AcuSQL as a Standalone Program.”

2.2.3 Commas

Use commas to separate list items in embedded SQL statements. For example:

```
EXEC SQL
    SELECT FIRST_NAME, LAST_NAME FROM EMPLOYEE
END-EXEC.
```

2.2.4 String Delimiters

To ensure the greatest portability of your SQL code, use single quotation marks around string constants in SQL.

```
EXEC SQL
     SELECT LAST_NAME INTO :LAST-NAME FROM EMPLOYEE
     WHERE LAST_NAME = 'SMITH'
END-EXEC.
```

2.3 Data Division

The following SQL statements are supported in the Data Division:

```
SQL DECLARE CURSOR
SQL INCLUDE
SQL declarations
```

For example:

```
EXEC SQL
    DECLARE cursorname CURSOR FOR SELECT statement
END_EXEC.
EXEC SQL INCLUDE SQLCA END-EXEC.
EXEC SQL INCLUDE SQLDA END-EXEC.
EXEC SQL INCLUDE filename END-EXEC.
EXEC SQL BEGIN DECLARE SECTION END-EXEC.
... SQL declarations
EXEC SQL END DECLARE SECTION END-EXEC.
```

2.3.1 Cursors

You can define cursors in both the Data Division and the Procedure Division. Note that when cursors are defined in the Data Division, the period (.) after the END-EXEC keyword is optional. However, if you do include the period, it must appear on the same line as the END-EXEC keyword. Therefore, both of the following are acceptable syntax when a cursor is defined in the Data Division section.

```
EXEC SQL
    DECLARE COBCUR1 CURSOR FOR
        SELECT
            C_FIRST_NAME,
            C_LAST_NAME,
        FROM CUSTOMER
            WHERE C_NUMBER >= 'SMITH'
END-EXEC.
EXEC SQL
    DECLARE COBCUR1 CURSOR FOR
        SELECT
            C_FIRST_NAME,
            C_LAST_NAME,
        FROM CUSTOMER
            WHERE C_NUMBER >= 'SMITH'
END-EXEC
```

See section 5.6.2 for a brief introduction to cursors. Consult any of the commercially available books on SQL for more complete details.

2.3.2 SQL INCLUDE files

You can include files of terminal or ANSI format in your ESQL source by using the SQL INCLUDE statement.

The standard SQLCA and SQLDA INCLUDE files are built into the pre-compiler and are included whenever they are named in the source, as in:

```
*  Include the SQL Communications Area (SQLCA)
   EXEC SQL INCLUDE SQLCA END-EXEC.
```

These files include specific adaptations for AcuSQL. *Do not attempt to substitute other versions of these files.*

User-defined INCLUDE files must begin with an SQL BEGIN DECLARE statement and end with an SQL END DECLARE statement if the code is to be used as ESQL code. Variables not within the BEGIN/END DECLARE section cannot be used as host variables. They can still be used as "local" variables. See section 2.3.3, "Host Variables," for additional information.

In statements of the form:

```
EXEC SQL INCLUDE filename END-EXEC.
```

the compiler adds to *filename* the standard file extension for COBOL source files (the default is ".cbl"). To specify another extension, or to exclude an extension, *filename* must be enclosed in either single or double quotation marks. For example:

```
EXEC SQL INCLUDE 'SQLCOPYBOOK' END-EXEC.
```

If the user-defined INCLUDE files do not reside in the current directory, you must indicate where they're located using the "-Pi" compiler option or ACUSQL_INCLUDE environment variable.

SQL Communications Area (SQLCA)

The SQLCA is a COBOL group item that is used to provide information to your ESQL/COBOL program. The SQLCA contains data items for error checking, warnings, and status information that is updated whenever an ESQL statement is executed. Your COBOL program can check this structure after each ESQL operation or in a centralized error-checking routine if you are using the WHENEVER directive in your application. See section 2.7 and section 5.3.1 for information on the WHENEVER directive.

The field most commonly used in the SQLCA group item is SQLCODE, which contains the status of the last statement executed. SQLCODE can contain the following values:

0	The executed statement succeeded without an error or warning.
>0	The executed statement succeeded, but there was a warning. Check the other fields in the SQLCA structure to determine the nature of the warning.
<0	The executed statement encountered an error. Check the other fields in the SQLCA structure to determine the nature of the error.

If SQLCODE contains a non-zero value, your application can check the value of SQLERRMC for the text associated with the error or warning, and, when appropriate, display that text as an error message or warning.

SQL Descriptor Area (SQLDA)

The SQLDA is a COBOL group item used in performing dynamic SQL operations. SQLDA is used to store information about the input and output variables of dynamically prepared SQL statements.

2.3.3 Host Variables

Host variables are the key to communication in an ESQL application. These are data items in Working-Storage that are made available to ESQL for communicating information to and from the database. Put another way, host variables can provide input to your ESQL program and the program can return information to host variables.

Host variables are standard COBOL Working-Storage items that are enclosed by BEGIN/END DECLARE statements. For example:

```
EXEC SQL BEGIN DECLARE SECTION END-EXEC.
01  EMPLOYEE-NAME.
    03  FIRST-NAME              PIC X(20).
    03  LAST-NAME               PIC X(20).
EXEC SQL END DECLARE SECTION END-EXEC.
```

This makes the variables “FIRST-NAME” and “LAST-NAME” available to the AcuSQL pre-compiler. When these variables are used in the Procedure Division in an SQL statement, they must be preceded by a colon (:). This differentiates host variables from local data objects with the same name. If you are using the variables in COBOL operations (rather than SQL operations), the colon should not be used. The pre-compiler recognizes COBOL variables only if they have been declared in an SQL BEGIN DECLARE section. See section 5.4.2 for a further discussion of host variables.

2.4 Data Types

To ensure proper and accurate data matches between the data source and your COBOL program, you must match the SQL data type in your source code with the appropriate COBOL data type. Note that only fixed-length and variable-length character strings are supported (LOB-type data is not supported). This section describes the data type compatibility between AcuSQL and DB2, Microsoft SQL Server, Microsoft Access, and MySQL.

2.4.1 DB2 Data Type Compatibility

The following table describes data type compatibility for DB2.

SQL Type	**COBOL Type**	**Description**
CHAR	01 *name* PIC X(n)	Fixed-length character string
DATE	01 *name* PIC X(10).	10-byte character string
DECIMAL	01 *name* PIC S9(M) V9(N) COMP-3.	Packed decimal (where m = p - n)
DOUBLE	01 *name* USAGE IS DOUBLE.	Double-precision floating point
INTEGER	01 *name* PIC S9(9) COMP-5.	32-bit signed integer
LONG VARCHAR(n)	01 *name*. 49 *name*-length PIC S9(4) COMP-5. 49 *name*-name PIC X(n).	Variable-length character string
REAL	01 *name* PIC S9(4) USAGE IS COMP-1.	Single-precision floating point
TIME	01 *name* PIC X(8).	8-byte character string
TIMESTAMP	01 *name* PIC X(26).	26-byte character string
VARCHAR(n)*	01 *name*. 49 *name*-length PIC S9(4) COMP-5. 49 *name*-name PIC X(n).	Variable-length character string (n <= 4000)

* To be treated as a VARCHAR, the elementary items in a group item consisting of only a numeric item and a character item need to be level 49 as in the following:

```
01  my-record.
    49  my-number          pic s9(3) comp-5.
    49  my-name            pic x(20).
```

2.4.2 Microsoft SQL Server Data Type Compatibility

The following table describes data type compatibility for SQL Server.

SQL Type	**COBOL Type**	**Description**
CHAR(n)*	`01` *name* `PIC X(n).`	Fixed-length character string
DATETIME	`01` *name* `PIC X(25).`	25-byte character string
DECIMAL(p,n)	`01` *name* `PIC S9(m)V9(n) COMP-3.`	Packed decimal (where m = p - n)
DOUBLE PRECISION	`01` *name* `USAGE IS DOUBLE.`	Double-precision floating point
INTEGER	`01` *name* `PIC S9(9) COMP-5.`	32-bit signed integer
REAL	`01` *name* `PIC S9(4) USAGE IS COMP-1.`	Single-precision floating point
SMALLINT	`01` *name* `PIC S9(4) COMP-5.`	16-bit signed integer
VARCHAR(n)**	`01` *name*`.` `49` *name*`-length PIC S9(4) COMP-5.` `49` *name*`-name PIC X(n).`	Variable-length character string (n <= 4000)

* To optimize performance, make your data variables one byte larger than the columns you are selecting. Data variables that are exactly the same size of the columns you are selecting may negatively impact performance. This is due to the API used to access the data, which requires room for a NUL-termination character for string data.

**To be treated as a VARCHAR, the elementary items in a group item consisting of only a numeric item and a character item need to be level 49 as in the following:

```
01  my-record.
    49  my-number          pic s9(3) comp-5.
    49  my-name            pic x(20).
```

2.4.3 Access Data Type Compatibility

The following table describes data type compatibility for Microsoft Access and AcuSQL.

SQL Type	**COBOL Type**	**Description**
CHAR (n)	`01` *name* `PIC X(n).`	Fixed-length character string
DOUBLE PRECISION	`01` *name* `USAGE IS DOUBLE.`	Double-precision floating point
INTEGER	`01` *name* `PIC s9(8) COMP-5.`	32-bit signed integer
SMALLINT	`01` *name* `PIC s9(3) COMP-5.`	16-bit signed integer
VARCHAR(n)*	`01` *name*`.` `49` *name*`-length PIC s9(4) COMP-5.` `49` *name*`-name PIC X(n).`	Variable-length character string (n <= 4000)
DATETIME	`01` *name* `PIC X(24).`	8-character string

* To be treated as a VARCHAR, the elementary items in a group item consisting of only a numeric item and a character item need to be level 49 as in the following:

```
01  my-record.
    49  my-number          pic s9(3) comp-5.
    49  my-name            pic x(20).
```

2.4.4 MySQL Data Type Compatibility

The following table describes data type compatibility for MySQL and AcuSQL.

SQL Type	COBOL Type	Description
CHAR (n)	`01 name PIC X(n).`	Fixed-length character string n <= 255
VARCHAR(n)	01 *name*. 49 *name*-length PIC S9(4) COMP-5. 49 *name*-name PIC X(n).	Variable-length character string (n <= 255). Sub-items should be level 49.
TINYINT	PIC 9(04) / PIC 9(02). *	1 byte -128 to +127
SMALLINT	PIC 9(09) / PIC 9(04). *	2 bytes -32768 to +32767
MEDIUMINT	PIC 9(09) / PIC 9(06). *	3 bytes -8388608 to +8388607
INTEGER (INT)	PIC 9(10) / PIC 9(09). *	4 bytes -2147483648 to +2147483647
FLOAT	01 *name* USAGE IS FLOAT.	4-byte field
DOUBLE	01 *name* USAGE IS DOUBLE.	8-byte field
DECIMAL(p,n)	01 *name* PIC S9(m)V9(n).	Numeric where m = p - n and p denotes the precision (in digits).
DATE	01 *name* PIC X(10).	Format is YYYY-MM-DD
TIME	01 *name* PIC X(8) or X(14)	Format HH:MM:SS or HHHH:MM:SS range -839:59:59 to 838:59:59

SQL Type	COBOL Type	Description
DATETIME	01 *name* PIC X(19)	Format is YYYY-MM-DD HH:MM:SS (space between day and hour) Range 1000-01-01 00:00:00 to 9999-12-31 23:59:59
TIMESTAMP	01 name PIC X(19)	Format is YYYY-MM-DD HH:MM:SS (space between day and hour) Range 1970-01-01 00:00:00 to 2037

* For numeric fields, unless the COBOL program is compiled with the “-Dz” option, the number will be truncated based on the number of 9’s in the picture clause. For maximum compatibility, we recommend using the larger picture size.

Note: Database and table names in MySQL are case-sensitive. Column names are not. There are multiple possible engines with MySQL .

2.5 Procedure Division

AcuSQL supports many SQL programming concepts and techniques. Several of the these areas are described in this section.

2.5.1 Cursors

Cursors are defined in the Procedure Division or in the Data Division. For example:

```
EXEC SQL DECLARE MYCURSOR CURSOR FOR
   SELECT FIRST_NAME, LAST_NAME
   FROM TEST_TABLE
END-EXEC.
```

AcuSQL also supports the declaration of scroll cursors, and the use of the following FETCH extensions:

Extension	**Use**
FETCH FIRST	Returns the first row of query results.
FETCH LAST	Returns the last row of query results.
FETCH PRIOR	Returns the row immediately preceding the current curser row.
FETCH NEXT	Returns the row immediately following the current curser row.
FETCH ABSOLUTE	Returns a specific row identified by a constant or host variable*.
FETCH RELATIVE	Moves the cursor forward or backward a specified number of times from its current position.

* If specifying a row position through a host variable, the host variable does not need to be declared in the DECLARE section of working-storage, and can be any numeric working-storage variable.

See section 5.6.2 for a brief introduction to cursors. Consult any of the commercially available books on SQL for more complete details.

2.5.2 Rowset Functions

The AcuSQL pre-processor allows for the use of Rowset functions. Microsoft SQL Server includes several built-in rowset functions, and also allows users to define their own rowset functions.

Please refer to the Microsoft SQL Server documentation for information about using and creating rowset functions.

2.5.3 Stored Procedures

If using Microsoft SQL Server, stored procedures can be called via embedded SQL. Some restrictions apply. (See section 2.5.3.1 for details.)

Generally, you can execute two types of stored procedures: those that return no result sets, and those that return a single result set. If you execute stored procedures that return more than one result, the results are not available to the COBOL program. AcuSQL does support output parameters and return code values, however.

Consider the following stored procedure that has one output parameter and also returns a value:

```
create procedure sp_listcustomer
    @lastname varchar(100) = NULL,
    @numrows int output
    as
select @numrows = count(*) from customer where c_last_name = @lastname
select c_last_name, c_first_name, c_birthday from customer
    where c_last_name = @lastname
return 23
```

This stored procedure returns a single result set (c_last_name, c_first_name, and c_birthday). Depending on the rows in the customer table, this can consist of many rows.

There are two ways to execute this procedure from a COBOL program:

1. Ignore the result sets, and just use the output parameters and return code. This can be done with the following code:

   ```
   display "Enter the name to search for: ", no.
   accept c-last-name.
   EXEC SQL exec :ret-code = exec sp_listcustomer
   (:c-last-name, :num-rows out)
   END-EXEC.
   ```

   Note that c-last-name, ret-code, and num-rows need to have been declared as valid variables in a DECLARE section. When executed this way, all the rows returned by the stored procedure are thrown away, and only the num-rows variable and the ret-code variable are changed. ret-code will be the value 23 (based on the "return 23" in the stored procedure)and num-rows will be the number of rows that match the WHERE.

2. Have the result set returned to the COBOL program. This requires a cursor to be declared, and is done with the following code:

```
display "Enter the name to search for: ", no.
accept c-last-name.
EXEC SQL declare spcursor cursor for
        :ret-code = exec sp_listcustomer (:c-last-name,
         :num-rows out)
END-EXEC.

EXEC SQL
    open spcursor
END-EXEC.

move 0 to num-rows-read.
perform until SQLCODE not = 0
    EXEC SQL
        FETCH spcursor into
         :c-last-name, :c-first-name,
         :c-birthday
    END-EXEC
    if SQLCODE = 0
        add 1 to num-rows-read
        display c-last-name, ", ", c-first-name, ", ",
               c-birthday
    end-if
end-perform.
if num-rows not = num-rows-read
    display "stored procedure error, " num-rows,
            " not = ", num-rows-read
end-if.
```

Note that c-last-name, ret-code, num-rows, c-first-name, and c-birthday need to have been declared as valid variables in a DECLARE section. When executed this way, the num-rows and ret-code variables are set to the values given by the stored procedure at the time the cursor is opened. You must then execute FETCH commands in order to get the result set columns. This particular example tests the number of rows actually fetched against the num-rows value returned by the stored procedure.

There are two types of syntax allowed by the precompiler when MSSQL syntax is in effect (that is, when you specify the “-Pk mssql” option).

```
EXEC SQL  [:status-var = ] EXEC procedure-name
[[:param-var [out[put]]],...]
```

and

```
EXEC SQL DECLARE cursor-name CURSOR FOR
[:status-var = ] EXEC procedure-name [[:param-var
[out[put]]], ...]
```

The second form requires all the usual steps necessary for cursors: you must open the cursor and then fetch from it until all the rows have been fetched.

Any output parameters, including the return value, are not returned to the COBOL program until an UNPREPARE or CLOSE (for cursors) is executed.

2.5.3.1 Restrictions

The stored procedures you execute must return no result sets or one result set. Stored procedures that return multiple result sets will execute, but all result sets except the first are thrown away.

The connection used when getting result sets from a stored procedure is not available for any other processing while the cursor is open. In other words, if you need to execute other SQL statements based on information from a stored procedure, you must open multiple connections to the database.

2.6 SQL Verbs

In embedded SQL applications, two of the most vital SQL verbs are CONNECT and DISCONNECT. These are the commands that open and close communication between your application and your data source. Your data source vendor most likely recommends the CONNECT and DISCONNECT syntax to access the data source. When accessing your data source through AcuSQL, however, you should use the CONNECT and DISCONNECT syntax described below instead of the syntax recommended for your data source.

With AcuSQL, SQL data can be read (using the SELECT or FETCH statement) or written to (using the INSERT statement). Most other SQL verbs are supported as well, including, but not limited to, DELETE and UPDATE. While it is not within the scope of this document to explain these standard SQL verbs, brief introductions are provided in Chapter 5 with the sample programs. In order to understand which SQL features your database supports, be sure to work with a knowledgeable database administrator.

2.6.1 CONNECT Statement

When connecting to an SQL data source through AcuSQL, use the following syntax for the CONNECT statement:

```
CONNECT TO server-name [AS connection-name] [USER userid USING passwd]
```

server-name, connection-name, and *userid* are alphanumeric literals or host variables defined in an ESQL DECLARE section of Working-Storage. Note that you must place a colon (":") before the host variable in the SQL statement.

passwd is a variable-length character string (VARCHAR, in DB2). Note that leading and trailing spaces are not stripped.

The CONNECTION statement attempts to establish a connection with *server-name* as user *userid*. The "AS *connection-name*" clause assigns the specified name to the connection. This is especially desirable when you want to establish multiple connections.

Example:

```
CONNECT TO big-data AS con1 USER :uid USING :pwd
```

Note that "big-data" and "con1" are alphanumeric literals. "uid" and "pwd" are host variables. A colon must precede host variables (":"). See the sample programs for a working example

To use Windows authentication when connecting to SQL Server, omit the USER and USING phrases..

2.6.2 DISCONNECT Statement

When disconnecting from an SQL data source through AcuSQL, use the following syntax for the SQL DISCONNECT statement.

```
DISCONNECT { CURRENT | ALL | connection-name }
CONNECT RESET
```

connection-name is an alphanumeric literal or host variable defined in an ESQL DECLARE section of Working-Storage. Note that you must place a colon (":") before the host variable in the statement.

DISCONNECT closes the *current* connection, *connection-name*, or all connections. DISCONNECT CURRENT and CONNECT RESET cause the *current* connection to be closed. When the current connection is closed the value of current connection remains empty. You must use SET CONNECTION to set a new current connection.

2.7 Checking Syntax

The pre-compiler supports both relaxed and strict syntax checking. You indicate relaxed checking with the "-Pr" command line option, strict checking by not including this option. See section 3.2.2, "Using AcuSQL as a Standalone Program," for information on the "-Pr" option. These choices operate on the entire source code file.

However, you may prefer to exercise strict checking in some parts of your program and relaxed checking in others. You can indicate the mode of syntax checking by indicating "Relaxed" or "Strict" with your EXEC ESQL command.

For example,

```
EXEC SQL RELAXED
    DECLARE COBCUR1 FOR
    SELECT
        C_NUMBER, C_NAME, TO_CHAR(C_BIRTHDAY, 'DD-MM-YY'), C_INFO
        FROM CUSTOMER
        WHERE C_NUMBER >= :C_NUMBER
END-EXEC.
```

or

```
EXEC SQL STRICT
    DECLARE COBCUR1 FOR
        SELECT
            C_NUMBER, C_NAME, TO_CHAR(C_BIRTHDAY, 'DD-MM-YY'), C_INFO
        FROM CUSTOMER
        WHERE C_NUMBER >= :C_NUMBER
END-EXEC.
```

(For a description of this code, see section 5.6, "Working With More Than One Row – select3.sqb.")

Note that you cannot indicate “Relaxed” or “Strict” in the following instances:

- EXEC SQL BEGIN DECLARE SECTION
- EXEC SQL END DECLARE SECTION
- EXEC SQL INCLUDE

2.8 Error Handling

To strengthen error handling, you should consider embedding the SQL WHENEVER directive in your source code. The WHENEVER directive is used to direct program logic when the application encounters a warning or an error. You can choose to ignore errors or redirect them with WHENEVER.

The default behavior of AcuSQL is to attempt to ignore warnings and error conditions, and continue processing, if possible. Your application can either explicitly check error conditions or, for more automatic error processing, use the WHENEVER directive. Your application can contain more than one WHENEVER directive. A WHENEVER directive applies to all embedded ESQL statements that follow it, until it is superseded by another WHENEVER directive.

Use the WHENEVER directive to specify what actions the program takes when it encounters one of the following conditions:

- SQLERROR
- SQLWARNING
- NOT FOUND. This condition applies only to SELECT and FETCH statements.

Specify one of the following actions when an error condition occurs:

- CONTINUE – control passes to the next statement
- PERFORM *paragraph_name* – the program performs *paragraph_name* and then returns to read the next statement.

- GOTO *paragraph_name* – the program branches to *paragraph_name*, a specified procedure in the program. Flow of the program continues down this branch.

- STOP

Syntax

```
EXEC SQL WHENEVER  {SQLERROR}   | {CONTINUE}
                   {SQLWARNING} | {PERFORM paragraph}
                   {NOT FOUND}  | {GO TO paragraph}
                                | {STOP}
```

where *paragraph* is a specified procedure in the program.

See section 5.3.1 for an example of the WHENEVER directive.

2.9 Error Messages

The SQLCA can accommodate only 70 characters of error information in the error message field. Generally, this is not enough for a useful error message. See “SQL Communications Area (SQLCA)” for additional information on this required file.

SQL Server Environments

In Microsoft SQL Server environments, the external variable F-ERRMSG is set to the full text of the error message returned to accommodate error messages longer than 70 characters.

F-ERRMSG is an external variable of USAGE POINTER that must be manipulated in order to see its contents. The following code demonstrates how to see the contents of the error message string:

```
01  f-errmsg          usage pointer external.
01  my-errmsg         pic x(512).

....
0000-START.
    EXEC SQL WHENEVER SQLERROR GO TO ERROR-EXIT END-EXEC.
....
```

```
ERROR-EXIT.
   EXEC SQL WHENEVER SQLERROR CONTINUE END-EXEC.
     DISPLAY "SQL ERROR: SQLCODE  " SQLCODE
     DISPLAY "        SQLSTATE " SQLSTATE
     DISPLAY SQLERRMC
     call "c$memcpy" using my-errmsg, by value f-errmsg, 512.
        display my-errmsg
     ACCEPT OMITTED
     STOP RUN.
```

where

f-errmsg is an external variable that contains the full text of the error message.
my-errmsg is the full text of the error message, which in this case can contain up to 512 bytes.
See Book 3, *Reference Manual*, of the ACUCOBOL-GT documentation set for information on C$MEMCPY and USAGE POINTER.

2.10 Limits and Restrictions

AcuSQL does not currently support any *large object* data types (LOBs). If your existing applications include LOBs, contact our Technical Services department for help evaluating the steps you need to take in order to use AcuSQL.

In addition, AcuSQL does not currently support *compound SQL* statements. For this reason, statements of the form:

```
EXEC SQL
     DROP TABLE table-name1
     DROP TABLE table-name2
END-EXEC
```

should be broken into single statements of the form:

```
EXEC SQL
     DROP TABLE table-name1
END-EXEC
EXEC SQL
     DROP TABLE table-name2
END-EXEC
```

Likewise, *DCLGEN generated code* must be “split” into data and procedure section files, and FILLER cannot be used as a group item.

Aside from these restrictions, you need to be aware that different databases react differently to SQL commands. For instance, during transaction management, Access performs an SQL COMMIT after a DROP TABLE, but DB2 does not. In order to discover the limits and restrictions of your database, talk with a knowledgeable database administrator.

2.11 Detecting and Handling NULL Values

Many SQL-conversant data sources allow a NULL (empty) value in a data field. COBOL and ACUCOBOL-GT do not recognize or support the NULL value. When the value of a field returned from a query is NULL, the contents of the bound variable are undefined. As a result, the ESQL programmer must make provisions for detecting and handling NULL when it is retrieved from or must be stored in the database.

To identify a NULL value, ESQL offers the indicator variable. The indicator variable is a host variable that is used for the special purpose of indicating whether the value of a field is NULL or some other value. In the COBOL program, indicator variables are declared in an SQL DECLARE section of Working-Storage (the same as other host variables). Indicator variables take the form:

```
01  indicator_name  pic s9(5) usage comp-5.
```

where *indicator_name* is a user-defined name such as “myind” (used in the example below).

In an ESQL statement, the indicator variable is paired with the bound variable and together they indicate a value. In the ESQL statement, the indicator variable immediately follows the bound variable and must be preceded with a colon (“:”). The ESQL reserved word “indicator” is optional. For example:

```
exec sql fetch next from mytable into :myfield indicator :myind ...
```

The value of the indicator valuable is checked to determine if the value of the field is NULL or some other value. Note that there are no commas between the variable and the indicator.

For input variables, such as those found in an INSERT or UPDATE statement:

If the value of the indicator variable is ...	**Then ...**
-1	AcuSQL attempts to set the value of the column to NULL.
>=0	AcuSQL attempts to set the value of the column to that of the host variable, or to the number that appears. This behavior is database dependent, and we strongly recommend that you test your data source to understand the values that are returned.

For output variables, such as those found in SELECT statements:

If the value of the indicator variable is ...	**Then ...**
-1	The selected column’s value is NULL.
0	The column’s value is assigned to the host variable.
>0	The column’s value is assigned to the host variable, but that value is truncated. The indicator variable is set to the number of characters read by the program.

Consult the ESQL chapter of your SQL reference manual for more information about the use of the indicator variable and the handling of NULL values.

3 Pre-compiler Function and Use

Key Topics

The ESQL Translation Process 3-2
Using the Pre-compiler........ 3-3
Environment Variables 3-12
Pre-compilation Errors 3-14

3.1 The ESQL Translation Process

The AcuSQL® pre-compiler identifies and translates ESQL statements into COBOL statements. It begins its work in the program's Data Division. The pre-compiler identifies ESQL statements by searching for the keywords "EXEC SQL" and "END-EXEC" in the source code. When it finds these markers, it encapsulates them with comments and line numbers and then parses and generates COBOL code, including CALLs to the AcuSQL library.

For example, if your code specifies an include file such as:

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

This line is translated into:

```
*(( PREPROC ACUSQL LINE BEGIN 12 ))
   01  SQLCA IS EXTERNAL.
       05  SQLCAID                PIC X(8).
       05  SQLCABC         COMP-5 PIC S9(9).
       05  SQLCODE         COMP-5 PIC S9(9).
       05  SQLERRM.
           10  SQLERRML    COMP-5 PIC S9(4).
           10  SQLERRMC           PIC X(70).
       05  SQLERRP                PIC X(8).
       05  SQLERRD OCCURS 6 TIMES COMP-5 PIC S9(9).
       05  SQLWARN.
           10  SQLWARN0           PIC X.
           10  SQLWARN1           PIC X.
           10  SQLWARN2           PIC X.
           10  SQLWARN3           PIC X.
           10  SQLWARN4           PIC X.
           10  SQLWARN5           PIC X.
           10  SQLWARN6           PIC X.
           10  SQLWARN7           PIC X.
           10  SQLWARN8           PIC X.
           10  SQLWARN9           PIC X.
           10  SQLWARNA           PIC X.
       05  SQLSTATE               PIC X(5).
       05  SQLERRM-PREFIX.
           10  SQLERRPL    COMP-5 PIC S9(4).
           10  SQLERRPC           PIC X(70)
*EXEC SQL INCLUDE SQLCA END-EXEC.
*(( PREPROC ACUSQL LINE END 12 ))
           .
```

The following ESQL statement:

```
EXEC SQL
    SELECT MIN(C_NUMBER)
        INTO :MIN-C-NUMBER
        FROM CUSTOMER
END-EXEC.
```

is translated into:

```
*(( PREPROC ACUSQL LINE BEGIN 41 ))
        PERFORM CALL "SQL$START" END-CALL CALL "SQL$PREPARE" USING 'S
     -  'QLISTM' "SELECT MIN(C_NUMBER) FROM CUSTOMER " END-CALL IF SQ
     -  LCODE OF SQLCA < 0 THEN GO TO Error-Exit END-IF CALL "SQL$BIN
     -  "DCOLUMN" USING 'SQLISTM' 1 MIN-C-NUMBER   END-CALL IF SQLCOD
     -  E OF SQLCA < 0 THEN GO TO Error-Exit END-IF CALL "SQL$CURSOR"
         USING 'SQLICUR' 'SQLISTM' IF SQLCODE OF SQLCA < 0 THEN GO TO
         Error-Exit END-IF CALL "SQL$OPEN" USING 'SQLICUR' IF SQLCODE
         OF SQLCA < 0 THEN GO TO Error-Exit END-IF CALL "SQL$FETCH" U
     -  SING 1 0 'SQLICUR' IF SQLCODE OF SQLCA < 0 THEN GO TO Error-E
     -  xit END-IF CALL "SQL$CLOSE" USING 'SQLICUR' IF SQLCODE OF SQL
     -  CA < 0 THEN GO TO Error-Exit END-IF CALL "SQL$UNPREPARE" USIN
      -  G 'SQLISTM'  END-CALL END-PERFORM
      *  EXEC SQL
      *      SELECT MIN(C_NUMBER)
      *          INTO :MIN-C-NUMBER
      *          FROM CUSTOMER
      *  END-EXEC.
      *(( PREPROC ACUSQL LINE END 45 ))
          .
```

The pre-compiler generates a standard 72-character line without regard for the end of statements or other formatting considerations.

The line numbers that appear in the comment lines specify the original line numbers in the input file.

Note: The precise results of these translations may vary depending on your version of the pre-compiler.

3.2 Using the Pre-compiler

The ACUSQL pre-compiler offers three modes of SQL syntax checking.

- The first mode, specified with the “-Pk” switch, uses internal reserved word lists and syntax rules to check IBM DB2 SQL, Transact-SQL, or ISO/ANSI SQL 92.

- The second mode, called *relaxed* mode, is enabled with the “-Pr” switch and provides only minimal syntax checking. It is intended for use with unspecified SQL variants.

- The third mode establishes a direct connection to the target database engine and uses the engine to directly check the syntax. This option is specified with the “-Pc” switch. Note that for this mode to be effective, your ODBC API middleware must support the return of syntax error messages (not all ODBC drivers do). Also note that during syntax verification it is possible that some *error* messages will be converted to *warning* messages. This is done, when possible, to prevent the compiler from aborting so that syntax checking can continue. The text of the warning message preserves the text of the error message.

These modes operate on the file as a whole. See section 2.7, “Checking Syntax,” for information on indicating relaxed or strict checking for individual EXEC SQL commands.

When the pre-compiler starts, it examines the source file to determine whether it is in ANSI or terminal format and then processes the file accordingly. If the file is in ANSI format, the pre-compiler removes characters from columns 1-6 and 72-80, replacing them with spaces. Information contained in these columns may not appear in the ACUCOBOL-GT compilation listing file.

You can start and use the AcuSQL pre-compiler in several ways:

- From AcuBench

- As a standalone program from the command line

- From the compiler command line

3.2.1 Using AcuSQL From Within AcuBench

Using AcuSQL within AcuBench is easy. Simply set the “-Ps” pre-compiler option (and any other desired options) in the File Compiler Options dialog box whenever you add an ESQL source file to the project. Thereafter, AcuBench automatically pre-compiles the file every time the project is *built* or the file is compiled.

Pre-compiler options, such as “-Pe”, are set in the File Compiler Options dialog box. To open the File Compiler Options dialog box, right-click on the ESQL file in the Source tab and select **Program Compile Options** from the menu. In the Catalog drop-down list of the File Compiler Options dialog box, select **Pre-compiler** from the list.

Pre-compiler options are identified on the command line by the “-P*x*” flag. Pre-compiler options are described in Section 3.2.3

If your ESQL files have a unique file suffix, such as “.sqb” (a suffix used by many IBM ESQL developers), you will want to add that suffix to the Source tab properties so that your ESQL files will automatically be included in the Source tab. To modify the Source tab properties, right click on the Source tab and select **Properties** from the pop-up menu. Add the suffix to the list in the “Extension names” entry field. Note that a semicolon separates each suffix except the last one.

ESQL files, like all COBOL source files, can be opened into the Code Editor by double-clicking on the file name in the Source tab or by right-clicking the file icon and selecting **View code** from the pop-up menu.

3.2.2 Using AcuSQL as a Standalone Program

You can run the AcuSQL pre-compiler from the command line as a standalone program or you can run it through the ACUCOBOL-GT compiler. Running the pre-compiler as a standalone gives you access to more pre-processor options, however, certain situations may require you to run AcuSQL from the main compiler. In particular, if you want to pre-process ESQL inside of copy files, you need to run AcuSQL from the compiler since the copy files must first be located by the compiler.

To start the pre-compiler from the command line, use the following syntax:

```
acusql [options] input_filename
```

This causes ACUCOBOL-GT to pre-compile and compile the specified source file. The following additional options affect pre-compiler behavior:

-Pc	Directs the pre-compiler to connect to and use the database engine to perform SQL syntax checking. This switch should be used in conjunction with the "-Pd", "-Pu", and "-Pp" switches; with the environment variables ACUSQL_DATASET, ACUSQL_USER, and ACUSQL_PASSWORD (see section 3.3, "Environment Variables"); or with some combination of switches and environment variables. Values given on the command line take precedence over values held by environment variables. Note that during syntax verification, in some cases an error message may be converted to a warning message in order to prevent the compiler from aborting. The text of the warning message preserves the text of the error message.
-Pd *dataset*	Is used in conjunction with the "-Pc" option and must be followed by the name of the ODBC data source to use.
-Pe* *(see special note at the end of this list)	Includes ESQL lines in a listing or in the debugger. If you want to see the preprocessed output, use "-Pe" and include a source listing by using "-Lf".

-Pi *dir*	Specifies the path that the pre-compiler will search to locate files named in SQL INCLUDE statements. For example “-Pi c:\SQL\include\” causes the pre-compiler to look in the “C:\SQL\include” directory for files named in SQL INCLUDE statements. You can also specify a search path via the ACUSQL_INCLUDE environment variable (see section 3.3, “Environment Variables”). Note that “-Pi” takes precedence over ACUSQL_INCLUDE when both are used.
-Pk *rlist*	Directs the pre-compiler to use the designated reserved word set for syntax checking. *rlist* can be “DB2”, to use the DB2 word set, “MSSQL” to use the Microsoft SQL Server word set, or “EXT”, to use the extended reserved word set. “EXT” is the default.
	Note: The syntax of the ALTER, CREATE, and GRANT statements is not checked by either the “-Pr” or “-Pk *rlist*” modes. These statements are checked only when you use the “-Pc” mode to connect directly to the database engine.
-Po	use output-file for prepocessed output
-Pp *password*	Is used in conjunction with the “-Pc” option and must be followed by the password that matches the username specified by the “-Pu” switch or by the ACUSQL_USER environment variable.

-Pr	Directs the pre-compiler to perform *relaxed* syntax checking. "-Pr" is intended for use in situations where the embedded SQL syntax does *not* conform to IBM SQL, Transact-SQL, or SQL92, but is correct for the database that you want to access.
	
	"-Pr" does not perform syntax checking on the following statements:
	
	`SELECT INTO FROM`
	`VALUES`
	`DELETE`
	`INSERT`
	`DECLARE CURSOR FOR SELECT`
	`UPDATE`
	
	Note: The syntax of the ALTER, CREATE, and GRANT statements is not checked by either the "-Pr" or "-Pk *rlist*" modes. These statements are checked only when you use the "-Pc" mode to connect directly to the database engine.

-Pu *username*	Is used in conjunction with the "-Pc" option and must be followed by the username of an authorized database user.
-Pv	Causes the compiler to treat group items as they were treated in releases earlier than Version 6.2; that is, group items containing both a numeric item and a character item only do not need to be level 49 to be treated as a VARCHAR. See Section 2.4, "Data Types," for additional information.

-Pw	Causes the pre-compiler to dump the current reserved word list to standard output (use the “-Pk” option to set the reserved word list). No pre-compilation or compilation is performed. Note that “-Pw” should not be used with any other pre-compiler switches. In AcuBench, “-Pw” must be specified in the “Additional options” entry field at the top-right of the File Compiler Options dialog box.
-St	Source is in terminal format
-Sa	Source is in ansi format

Note: Previous versions of the compiler accepted “-Po” *filename*, which enables you to view the pre-processed output. The compiler no longer accepts “-Po”, however, it is allowed if running AcuSQL as a standalone program. If you are running AcuSQL from the compiler and want to view the pre-processed output, use “-Pe” as described above.

3.2.3 Using AcuSQL from the Compiler

Use the “-Ps” compiler switch, and specify the source file you wish to pre-process. For example:

```
ccbl -Ps [options] input_filename
```

The following options are available to modify how AcuSQL executes from the compiler.

-Pc	Directs the pre-compiler to connect to and use the database engine to perform SQL syntax checking. This switch should be used in conjunction with the "-Pd", "-Pu", and "-Pp" switches; with the environment variables ACUSQL_DATASET, ACUSQL_USER, and ACUSQL_PASSWORD (see section 3.3, "Environment Variables"); or with some combination of switches and environment variables. Values given on the command line take precedence over values held by environment variables. Note that during syntax verification, in some cases an error message may be converted to a warning message in order to prevent the compiler from aborting. The text of the warning message preserves the text of the error message.
-Pd *dataset*	Is used in conjunction with the "-Pc" option and must be followed by the name of the ODBC data source to use.
-Pe* *(see special note at the end of this list)	Includes ESQL lines in a listing or in the debugger. If you want to see the preprocessed output, use "-Pe" and include a source listing by using "-Lf".
-Pi *dir*	Specifies the path that the pre-compiler will search to locate files named in SQL INCLUDE statements. For example "-Pi c:\SQL\include\" causes the pre-compiler to look in the "C:\SQL\include" directory for files named in SQL INCLUDE statements. You can also specify a search path via the ACUSQL_INCLUDE environment variable (see section 3.3, "Environment Variables"). Note that "-Pi" takes precedence over ACUSQL_INCLUDE when both are used.

-Pk *rlist*	Directs the pre-compiler to use the designated reserved word set for syntax checking. *rlist* can be “DB2”, to use the DB2 word set, “MSSQL” to use the Microsoft SQL Server word set, or “EXT”, to use the extended reserved word set. “EXT” is the default.

Note: The syntax of the ALTER, CREATE, and GRANT statements is not checked by either the “-Pr” or “-Pk *rlist*” modes. These statements are checked only when you use the “-Pc” mode to connect directly to the database engine.

-Pp *password*	Is used in conjunction with the “-Pc” option and must be followed by the password that matches the username specified by the “-Pu” switch or by the ACUSQL_USER environment variable.
-Pr	Directs the pre-compiler to perform *relaxed* syntax checking. “-Pr” is intended for use in situations where the embedded SQL syntax does *not* conform to IBM SQL, Transact-SQL, or SQL92, but is correct for the database that you want to access.

“-Pr” does not perform syntax checking on the following statements:

```
SELECT INTO FROM
VALUES
DELETE
INSERT
DECLARE CURSOR FOR SELECT
UPDATE
```

Note: The syntax of the ALTER, CREATE, and GRANT statements is not checked by either the “-Pr” or “-Pk *rlist*” modes. These statements are checked only when you use the “-Pc” mode to connect directly to the database engine.

-Pu *username*	Is used in conjunction with the "-Pc" option and must be followed by the username of an authorized database user.
-Pv	Causes the compiler to treat group items as they were treated in releases earlier than Version 6.2; that is, group items containing both a numeric item and a character item only do not need to be level 49 to be treated as a VARCHAR. See Section 2.4, "Data Types," for additional information.
-Pw	Causes the pre-compiler to dump the current reserved word list to standard output (use the "-Pk" option to set the reserved word list). No pre-compilation or compilation is performed. Note that "-Pw" should not be used with any other pre-compiler switches. In AcuBench, "-Pw" must be specified in the "Additional options" entry field at the top-right of the File Compiler Options dialog box.

Note: Previous versions of the compiler accepted "-Po" *filename*, which enables you to view the pre-processed output. The compiler no longer accepts "-Po", however, it is allowed if running AcuSQL as a standalone program. If you are running AcuSQL from the compiler and want to view the pre-processed output, use "-Pe" as described above.

3.3 Environment Variables

The following variables can be used to control the way AcuSQL accesses your data source. They can be set at the system/environment level or they can be specified in a runtime configuration file.

ACUSQL_DATASET
ACUSQL_INCLUDE
ACUSQL_PASSWORD
ACUSQL_USER

ACUSQL_DATASET

The ACUSQL_DATASET environment variable is used in conjunction with the “-Pc” pre-compiler switch (“-Pc” directs the pre-compiler to connect to the target database engine to perform syntax checking). ACUSQL_DATASET should be assigned the name of the ODBC data source to use. (Windows users refer to this value as the User DSN.)

ACUSQL_INCLUDE

The ACUSQL_INCLUDE environment variable can be used in place of the “-Pi” option to specify multiple search directories for SQL INCLUDE files. ACUSQL_INCLUDE is defined in the environment. Directories are searched in the order specified, first to last. A semicolon (“;”) is used after each path except the last one. For example, to set ACUSQL_INCLUDE to search the current directory, the directory “C:\SQL\LIBRARY”, and the directory “C:\COBOL\ESQL\LIBRARY”, you would execute the following statement in the DOS shell or in “autoexec.bat”:

```
SET ACUSQL_INCLUDE = ; C:\sql\library; C:\cobol\esql\library
```

The initial semicolon indicates an empty directory path (i.e., the current directory). Note that when both “-Pi” and ACUSQL_INCLUDE are used, the “-Pi” option takes precedence.

ACUSQL_PASSWORD

The ACUSQL_PASSWORD environment variable is used in conjunction with the “-Pc” pre-compiler switch (“-Pc” directs the pre-compiler to connect to the target database engine to perform syntax checking). ACUSQL_PASSWORD should be assigned the value of the password that goes with the username specified with the “-Pu” switch or in the ACUSQL_USER environment variable.

ACUSQL_USER

The ACUSQL_USER environment variable is used in conjunction with the “-Pc” pre-compiler switch (“-Pc” directs the pre-compiler to connect to the target database engine to perform syntax checking). ACUSQL_USER should be assigned the username of an authorized database user.

3.4 Pre-compilation Errors

If the pre-compiler encounters illegal ESQL, it attempts to report the error and continue pre-compilation. If the pre-compiler cannot continue, pre-compilation is terminated.

Refer to your SQL reference manual for the meanings of pre-compiler error messages.

4 Program Execution

Key Topics

Running Your Application 4-2
Running Your Application with Microsoft SQL Server 4-2
Debugging and File Tracing.. 4-7
Error Messages.. 4-8

4.1 Running Your Application

Before using the AcuSQL® runtime for the first time, you should test your Windows-to-data or UNIX-to-data host connection software. If you're using IBM's DB2 Connect software, you can test your connection under Windows by launching the Client Configuration Assistant and clicking **Test**. For full testing instructions, consult your DB2 Connect documentation. To test other vendors' connectivity software, consult your product documentation. It is essential that you prove the proper functioning of your database connection software before attempting to run your program.

The program object file created by successful pre-compilation and compilation is ready for immediate execution. Generally, there are no special options or configuration settings needed to run an ESQL program. However, if you are using transactions in a SQL Server environment, you will need to set some runtime configuration variables. Refer to section 4.2.1 for information on these variables.

To start the program within AcuBench® integrated development environment, simply double-click on the executable file in the Object tab. To start the program from the DOS or UNIX command line, enter the name of your ACUCOBOL-GT® runtime executable ("wrun32.exe" on DOS, "runcbl" on UNIX) followed by the name of the executable file (the runtime executable must be in a directory that is included in the current PATH definition). For a detailed discussion of the runtime environment and runtime options, see section 2.2 of the *ACUCOBOL-GT User's Guide*.

4.2 Running Your Application with Microsoft SQL Server

AcuSQL offers a special runtime DLL for running the product with Microsoft SQL Server. Note that the pre-compiler is unchanged. To pre-compile COBOL programs with embedded SQL, use the procedures found in Chapter 3.

The new runtime DLL is named "asqlsrvr.dll". You can use this DLL in two ways:

- Rename the DLL to “esqllib.dll” for the runtime to use this DLL when attempting to execute any AcuSQL commands.
- Give the DLL a name of your choice, and set the configuration variable ACUSQL_RUNTIME_DLL to the name you have chosen. We recommend keeping the name as “asqlsrvr.dll” and setting ACUSQL_RUNTIME_DLL to “asqlsrvr.dll”.

When getting runtime version information, in order to list the versions of the AcuSQL runtime DLLs, the runtime attempts to find and load all AcuSQL DLLs that it knows about. Currently these are “esqllib.dll” and “asqlsrvr.dll”. This is why we recommend not changing the name of the DLL.

ACUSQL_RUNTIME_DLL must be set in the configuration file; this variable cannot be set from the COBOL program. In addition, note that the value of ACUSQL_RUNTIME_DLL cannot be changed from the COBOL program once an AcuSQL command has been executed. In other words, it is not possible to use both “esqllib.dll” and “asqlsrvr.dll” in the same run unit.

When connecting to a server, the runtime DLL for the AcuSQL interface to SQL Server environments takes the name of the server, not the name of a datasource. Username and password information is used as described in section 3.3, “Environment Variables.”

The following statements are not supported by AcuSQL interface to Microsoft SQL Server:

DESCRIBE *ident* INTO *parameter*

FETCH … USING DESCRIPTOR *parameter*

EXECUTE *statement* USING DESCRIPTOR *parameter*

OPEN *cursor* USING DESCRIPTOR *parameter*

CALL *procname* USING DESCRIPTOR *parameter*

All other statements are supported.

4.2.1 Runtime Configuration Variables for SQL Server

In addition to the runtime configuration variable ACUSQL_RUNTIME_DLL, described in section 4.2, AcuSQL support for SQL Server includes additional runtime configuration variables:

ASQL_BUFFER_SIZE
ASQL_CONNECT_DATABASE
ASQL_NULL_ALPHA_SPACES
ACUSQL_SQLSTATE_2000_ON_EOD

These variables are of particular use when working with transactions. The variables can be set in the environment or in a runtime configuration file.

ASQL_BUFFER_SIZE

ASQL_BUFFER_SIZE can be set to any positive number, and causes AcuSQL to cache that many rows from a cursor when the cursor is read. This can result in much less traffic across the network, and, therefore, better performance. Values as small as “5” (to cache 5 rows) can result in a significant performance boost. Test your application for the optimal value to which to set this variable.

ASQL_BUFFER_SIZE is tested when a cursor is opened. Setting the variable programmatically after a cursor is open has no effect.

ASQL_CONNECT_DATABASE

The configuration variable ASQL_CONNECT_DATABASE enables you to specify a database other than the default for accessing data. If ASQL_CONNECT_DATABASE is set to a non-blank value, the driver executes “use [database_name]” as soon as the connection is established. From that point, all data access via that connection will be relative to the specified database.

Note that the database can be specified only before the connection; it is not possible to change databases on the fly.

If the system cannot execute the USE statement, the connection attempt fails with SQLCODE 08004. This can occur, for example, if the specified database does not exist, or if the user doesn not have permissions to access that database.

ASQL_NULL_ALPHA_SPACES

ASQL_NULL_ALPHA_SPACES allows you to determine how NULL data being returned to an ALPHA field is handled. In some cases, you may want the NULL data returned as ALL SPACES, or you may want the NULL data returned unchanged. Set this variable to the non-default value of "TRUE" if you want NULL data returned to an ALPHA field as ALL SPACES. The default value is "FALSE", and will return NULL data to an ALPHA field unchanged.

ACUSQL_SQLSTATE_2000_ON_EOD

Some embedded SQL products set SQLSTATE to the value of “02000” when end of data is reached on FETCH (SQLCODE = 100). The "ASQL_SQLSTATE_2000_ON_EOD" enables you to duplicate this behavior. Set this variable to the non-default value of “TRUE” to cause SQLSTATE to have the value "02000" whenever end of data is reached. When set to its default value of “FALSE”, SQLSTATE will have the value "00000."

4.2.2 Runtime Configuration Variables for esqllib

When using the esqllib.dll as described in Section 4.2, AcuSQL support for esqllib.dll includes additional runtime configuration variables:

ACUSQL_NO_AUTOCOMMIT
ACUSQL_ODBC_CURSORS
ACUSQL_USE_CONCURRENT
ODBC_CURSOR_TYPE

ACUSQL_NO_AUTOCOMMIT

ACUSQL_NO_AUTOCOMMIT explicitly turns off any auto-commit behavior that your ODBC driver may have. AUTOCOMMIT affects items such as record locking and the ability to roll back work. It is up to the driver vendor to determine whether a driver auto-commits work. If ACUSQL_NO_AUTOCOMMIT is set to “1”, the runtime explicitely turns auto-commit off whether or not it is the default. If ACUSQL_NO_AUTOCOMMIT is set to “0”, the runtime leaves the driver in its default state. Please refer to the documentation provided by your driver vendor for default COMMIT behavior.

ACUSQL_ODBC_CURSORS

This variable prevents the Microsoft cursor library from loading. When loaded, this library removes the "FOR UPDATE" clause from SELECT statements resulting in records not being locked. To prevent this, set ACUSQL_ODBC_CURSORS to “0”, which will block the library from loading. The default value is “1”, and will allow the Microsoft cursor library to load.

Note: The effect on your application is application and ODBC driver dependent. Please refer to the Microsoft documentation and the documentation from your driver vendor.

ACUSQL_USE_CONCURRENT

On some machines, calls to set concurrency mode can cause a memory access violation (MAV). The ACUSQL_USE_ CONCURRENT configuration variable can be used to prevent such a violation. Set ACUSQL_USE_ CONCURRENT to “0” (off, false, no) to prevent concurrency mode calls from being made. The default is “1” (on, true, yes).

ODBC_CURSOR_TYPE

Set ODBC_CURSOR_TYPE to one of the following to specify a cursor type to use:

For cursor type ...	**Set ODBC_CURSOR_TYPE to ...**
SQL_CURSOR_FORWARD_ONLY	0
SQL_CURSOR_KEYSET_DRIVEN	1
SQL_CURSOR_DYNAMIC	2
SQL_CURSOR_STATIC	3

For example:

```
    ODBC_CURSOR_TYPE 3
```

sets the curor type to a static cursor.

On Windows, the default is “3” (static). On other systems, the default is the driver default.

Note that not all cursor types are supported by all drivers or in all circumstances. Therefore, you are encouraged to check your database and operating system documentation before using this variable.

4.3 Debugging and File Tracing

Should you encounter an error in the execution of your program, you can use the debugging facilities in AcuBench or ACUCOBOL-GT to investigate the problem. See section 1.7.2 and Chapter 3 of the *ACUCOBOL-GT User’s Guide* for a detailed description of debugger options and use. Once the debugger is running, it may be helpful to use the “tf” option to turn on file tracing. Included in the file trace are all calls to “esqllib.dll” and “asqlsrvr.dll”. To generate the most detailed level of file tracing, follow “tf” with the number “9”.

4.4 Error Messages

SQL error messages reported during program execution are directed to *standard error* output. Refer to your SQL reference manual for the meanings of SQL error messages. Programmers can check for error codes in their programs by examining the values of SQLSTATE and SQLCODE.

MySQL Users

Note that some MySQL drivers do not correctly clear their error buffers after reporting an error. This can cause the driver to produce an infinite loop reporting the same error message. If you keep receiving the same error message over and over, this could be the source of the problem. Please check with your driver manufacturer for detail.

DB2 Users

For error messages generated by DB2 Connect or other connectivity software, refer to your product documentation.

5 Sample Programs

Key Topics

Sample Programs... 5-2
Static SQL 5-3
Creating Tables With ESQL — create.sqb.. 5-3
Using Working-Storage Items – select1.sqb 5-8
Group Items in the INTO Clause – select2.sqb....................................... 5-14
Working With More Than One Row – select3.sqb 5-15
Updating Data – update.sqb 5-20
Full SQL Program 5-25

5.1 Sample Programs

Your AcuSQL® pre-compiler comes with several sample programs that demonstrate embedded SQL in a COBOL program. This chapter describes five of these sample programs, and uses these programs to introduce SQL concepts and syntax. In several places, a syntax statement appears. Given the variety of databases, this syntax statement is quite simplified. Full syntax statements can run pages in length. We encourage you to check your database documentation or any of the commercially available SQL texts for a broader treatment of the concepts and syntax described in this chapter.

The sample progams on your distribution have been placed in different sub-directories for different data sources, for example Access, DB2, etc. Not all of the data sources will support all of the sample applications. The samples also differ in areas such as table creation syntax and date formats. We recommend that you take the time to review the differences in these sample programs if you intend to write ESQL applications that run on multiple databases. If you data source is not explicitely listed in the examples, you may examine the various samples to find the one whose syntax most closely matches your target data source.

Note: All examples of syntax given in this chapter pertain to embedded SQL in ACUCOBOL-GT® programs with the specified sample database. If necessary, please consult any of the commercially available books on SQL for a more generic treatment of embedded SQL.

The following sample files are provided with AcuSQL and documented in this chapter:

File Name	**Description**
create.sqb	Loads sample data into database
select1.sqb	Simple SELECT using Working-Storage items
select2.sqb	Simple SELECT using a group item
select3.sqb	Working with CURSORs
update.sqb	Modifying data through a CURSOR

The programs were compiled on a Windows using the following command in a Windows2000 environment:

```
ccbl32 -ps filename.sqb
```

The program was then run by executing the following command:

```
wrun32 filename
```

5.2 Static SQL

Embedded SQL programs are based on either static SQL or dynamic SQL. A program can be said to be static SQL when access to the database has been predetermined by the programmer. While the user may input values in response to a query, the query syntax itself is already in place. Put another way, all SQL statements are already part of the program when it is executed.

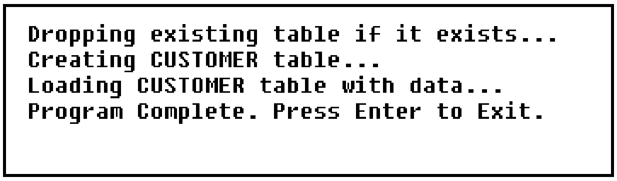
5.3 Creating Tables With ESQL — create.sqb

The “create” program creates a table and loads data into it from the text or line sequential “customer” file. The table that is created provides the source data for the other sample programs, so be sure to compile “create.sqb” and run “create.acu” before working with the other programs.

This program demonstrates the following ESQL and SQL concepts:

- The WHENEVER statement
- Use of DDL (Data Definition Language) to create tables
- The INSERT statement

When you run the “create” program, you see the following on your screen:



```
Dropping existing table if it exists...
Creating CUSTOMER table...
Loading CUSTOMER table with data...
Program Complete. Press Enter to Exit.
```

5.3.1 WHENEVER Directive

The WHENEVER directive controls the flow of the program in the event it encounters an error, warning, or NOT FOUND condition. Note that NOT FOUND conditions are the result of a SELECT or FETCH operation. See section 5.4.1 and section 5.6.5, respectively, for more information on SELECT and FETCH.

Syntax

```
EXEC SQL WHENEVER  {SQLERROR}   | {CONTINUE}
                   {SQLWARNING} | {PERFORM paragraph}
                   {NOT FOUND}  | {GO TO paragraph}
                                | {STOP}
```

where *paragraph* is a specified procedure in the program.

Example

The following statement appears in the “create.sqb” program:

```
EXEC SQL WHENEVER SQLERROR GO TO Error-Exit END-EXEC.
```

This indicates that when the program encounters an error condition, it proceeds to the Error-Exit routine. See section 2.8 for additional information on the WHENEVER directive.

5.3.2 Using DDL

DDL is an acronym for Data Definition Language. You use DDL statements to create or modify the structure of the database. Creating or dropping tables, or changing the number of columns in a table are all done by means of the DDL.

The “create.sqb” program contains two examples of DDL statements: DROP TABLE and CREATE TABLE. The DROP TABLE statement causes the program to delete the specified table from the database. Conversely, the CREATE TABLE statement causes the program to create a table with the specified name in the database.

Syntax

```
DROP TABLE tablename
CREATE TABLE tablename (column1 [, column 2, … ])
```

Note: These statements are intended to demonstrate basic, generic syntax. However, syntax for creating tables is database dependent. When creating tables, please refer to the documentation that accompanied your database.

Examples

The following statements appear in the “create.sqb” program.

```
EXEC SQL
     DROP TABLE customer
END-EXEC.
EXEC SQL
    CREATE TABLE customer
        C_NUMBER           INTERGER NOT NULL,
        C_FIRST_NAME       CHAR(20).
        .
        .
        .
        PRIMARY KEY        (C_NUMBER)
END-EXEC.
```

With these statements, the program deletes from the database any tables called “customer” that are owned by the user issuing the command. (The user’s area of the database is called a *schema*). A new table by the same name is then created and defined. This precludes overwriting the original “customer” table.

Note: It is possible to delete tables from other areas of the database by using a qualified name. See any of the commercially available SQL texts for more information.

5.3.3 INSERT Statement

The INSERT statement is an example of DML (Data Manipulation Language). With INSERT, you are adding data to a table in the database, but you are not changing the structure of the data in any way. The database still contains the same number of tables, and the tables still contain the same number of columns. (Compare this with the CREATE statement, which creates the table, but does not insert data into the table.)

Syntax

The following is a general syntax statement for INSERT

```
INSERT INTO <tablename> [(field-list)]
VALUES (<values-list>)
```

where …

tablename is the name of the table where you are inserting data

field-list is an optional list of fields to which you want to assign values. This list must be written according to the following syntax:

```
<field-name> [,<field-name] …
```

Omitting this list specifies all the fields of the table, using the same order as they are defined.

values-list is a list of literals. The values of these literals are assigned to the fields specified in <field-list>, according to the order in which they are listed. For this reason, the number of elements of both lists must be equal. At the same time, the value of the literal must be compliant with the type of field (alphanumeric, numeric, etc.) used in the table definition.

value-list has the following format:

```
<literal> [,<literal>] …
```

Example

```
EXEC SQL
    INSERT INTO CUSTOMER VALUES
        (:C-NUMBER, :C-FIRST-NAME, :C-LAST-NAME, :C=BIRTHDAY: C:-INFO)
END-EXEC.
```

With this statement, the program inserts values into the columns of the “customer” table. See section 2.3.3 for a general introduction to host variables and section 5.4.2 for an example using host variables.

5.3.4 Date Format

As a developer, you must keep in mind that date formats can vary by database and by country. Another thing to consider is whether you want to indicate the date alone or a timestamp. Be sure to check the documentation that comes with your database for information on date formats. In the coming sections, you’ll see sample code refering to the “c_birthday” variable. This is simply a placeholder; if you are working with the sample files, you will want to substitute database-specific code to specify the date.

5.3.5 Putting It All Together

You have now created a database table with the following format:

Name	**Nullable**	**Type**
C_NUMBER	Not null	NUMBER(38)
C_FIRST_NAME	Yes	CHAR(20)

Name	**Nullable**	**Type**
C_LAST_NAME	Yes	CHAR(20)
C_BIRTHDAY	Yes	DATE
C_INFO	Yes	CHAR(10)

The contents of the “customer” table looks similar to the following:

C_NUMBER	**C_FIRST_NAME**	**C_LAST_NAME**	**C_BIRTHDAY**	**C_INFO**
1	Sam	Snead	13-APR-52	New
2	John	Future	01-JAN-01	New
3	Betty	Falcon	01-FEB-90	New
4	Sally	Snead	31-DEC-54	New
5	John	Jones	29-FEB-60	New

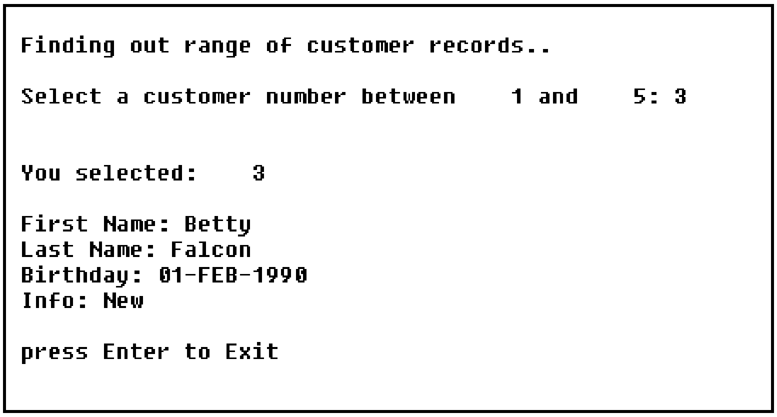
5.4 Using Working-Storage Items – select1.sqb

Compile and run the “select1” program to find the range of customer numbers in the “customer” table, prompt the user for a customer number, and then retrieve that record. Remember to compile and run the “create” program before running “select1.”

This program demonstrates the following SQL concepts:

- SELECT statements for returning a single row of data.
- host variables.

- MIN and MAX functions. These are group functions that operate on a field, or column, of data.



```
Finding out range of customer records..

Select a customer number between    1 and    5: 3


You selected:    3

First Name: Betty
Last Name: Falcon
Birthday: 01-FEB-1990
Info: New

press Enter to Exit
```

5.4.1 SELECT Statement

Use SQL's SELECT statement to view information in the database. You can see the values of all fields for an entry or you can see a specified subset.

Note: You can also use SELECT statements to view data from more than one table. See your SQL documentation for information about JOIN statements.

Syntax

At its most basic, the syntax for the SELECT statement is as follows:

```
SELECT <field-name-1>[, <field-name-2>] …
FROM tablename
```

where

field-name represents a column or columns in *tablename*.

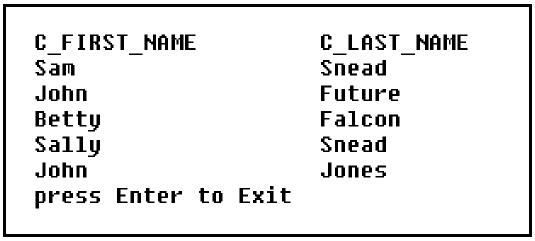
tablename is the name of the database table .

Example

For example, the statement

```
SELECT C_FIRST_NAME, C_LAST_NAME FROM CUSTOMER
```

returns the first and last names of customers listed in the “customer” table:



```
C_FIRST_NAME          C_LAST_NAME
Sam                   Snead
John                  Future
Betty                 Falcon
Sally                 Snead
John                  Jones
press Enter to Exit
```

5.4.2 Host Variables

Host variables provide a way for embedded SQL to use variables that are declared in COBOL. They are declared in the SQL DECLARE section of Working-Storage. They are distinguished from database objects (such as columns or tables) by a leading colon (“:”). Host variables can be used as input to a SELECT statement or they can be used as output to store the results of an SQL query. The pre-compiler recognizes COBOL variables only if they have been declared in an SQL BEGIN DECLARE section.

For example, in the “select1.sqb” program, the variables MIN-C-NUMBER and MAX-C-NUMBER are declared in Working-Storage. When embedded SQL statements refer to these variables, they appear as :MIN-C-NUMBER and :MAX-C-NUMBER. See section 2.3.3 for more information on host variables.

5.4.3 INTO Clause

As indicated above, host variables are used to store the result of an SQL query. The resulting value is put INTO the variable.

Syntax

```
SELECT database-field(s)
INTO :host-variable
```

where

database-field(s) is one or more variables defined in Working-Storage.

host-variable is the name of a COBOL variable.

Example

The “select1.sqb” program contains the following code:

```
EXEC SQL
   SELECT
       C_FIRST_NAME, C_LAST_NAME,
       C_BIRTHDAY, C_INFO
   INTO
       :C-FIRST-NAME, :C-LAST-NAME, :C-BIRTHDAY, :C-INFO
   FROM CUSTOMER
       ...
END-EXEC.
```

This series of statements instructs the program to place the values for these fields INTO the corresponding COBOL variables, which are C-FIRST-NAME, C-LAST-NAME, etc., so that COBOL statements can refer to them. The semicolon in the code indicates that these are COBOL (host) variables.

Note: In this example, there is a one-to-one match between variables declared in Working-Storage and the columns in the table. The program “select2.sqb” provides an example where the values to INTO a group item rather than individual fields.

5.4.4 WHERE Clause

The WHERE clause imposes conditions on the SELECT statement. Put another way, it restricts the number of rows returned, based on the indicated criteria.

Syntax

```
SELECT <field-name-1>[, <field-name-2>] …
FROM tablename
WHERE condition
```

Example

```
EXEC SQL
    SELECT
        C_FIRST_NAME, C_LAST_NAME,
        TO_CHAR, C_INFO
    INTO
        :C-FIRST-NAME, :C-LAST-NAME, :C-BIRTHDAY, :C-INFO
    FROM CUSTOMER
    WHERE C_NUMBER = :C-NUMBER
END-EXEC.
```

Using the example from the previous section, the program to place values for these fields INTO the corresponding COBOL variables, which are C-FIRST-NAME, C-LAST-NAME, etc. The semicolon in the code indicates that these are COBOL (host) variables. See an SQL text for more information on conditions and operators supported by SQL.

5.4.5 MIN and MAX Group Functions

A SELECT statement can include any of a number of SQL aggregate functions. The MIN and MAX functions are two functions supported by AcuSQL. The MIN function returns the minimum value of the column from all the rows in a table; the MAX function returns the maximum value.

Syntax

The field name to which the function applies must follow the name of the function and must be enclosed in parentheses.

```
SELECT MIN(fieldname)
SELECT MAX(fieldname)
```

Example

The “select1.sqb” program specifies records containing a customer number. This field is called “c-number”.

Given that this field contains a range of numbers, if you want to find the smallest value in the range, use the following SELECT statement:

```
SELECT MIN(C_NUMBER)
```

Similarly, to find the highest value in the range, use the following SELECT statement:

```
SELECT MAX(C_NUMBER)
```

5.4.6 Putting It All Together

Putting this all together, the following statement that appears in the “select1.sqb” program

```
EXEC SQL
    SELECT MIN(C_NUMBER)
    INTO :MIN-C-NUMBER
    FROM CUSTOMER
END-EXEC.
```

instructs the program to do the following:

1. Start executing an embedded SQL statement.
2. Find the minimum value in the “c_number” field of the customer table.
3. Assign that value to the COBOL host variable “:min-c-number”.
4. Stop executing the embedded SQL statement.

If you haven't done so already, compile and run the "select1" program. These steps provide one input for the statement "Select a customer number between 1 and 5."

5.5 Group Items in the INTO Clause – select2.sqb

The program "select2.sqb" is very similar to "select1.sqb." Users will see no difference when they run it. From a developer standpoint, however, the use of the INTO clause is different: the data is placed INTO a group item. This provides a "shortcut" for you, the programmer. Please read section 5.4.2 and section 5.4.3 about host variables and the INTO clause, respectively, for background information.

The "select2.sqb" program contains the following code:

```
EXEC SQL
   SELECT
        C_NUMBER, C_FIRST_NAME, C_LAST_NAME,
        C_BIRTHDAY, C_INFO
   INTO :C-RECORD
   FROM CUSTOMER
        WHERE C_NUMBER = :C-NUMBER
END-EXEC.
```

This instructs the program to take the values of fields in the database table and put these respective values INTO a COBOL group item host variable called :C-RECORD. The host variable is a group item; generated code will have individual fields.

Note: There is a special case when working with group items where the generated code is not broken up into the individual fields. This is the case of a VARCHAR datatype. The structure of the COBOL group item must meet a strict format. Please refer to the data type compatibility section of the documentation for your data source for more information.

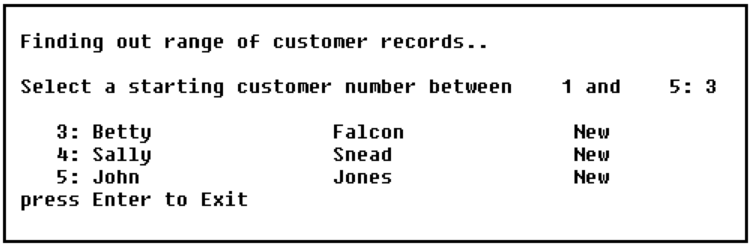
5.6 Working With More Than One Row – select3.sqb

The programs “select1.sqb” and “select2.sqb” each returned one row of data when you entered a customer number. Compile and run the program “select3” to receive data on more than one row. (Remember to compile and run the “create” program if you haven’t already. This generates the data for the sample programs.)

This program demonstrates

- working with cursors
- the FETCH statement

When you run “select3.acu”, you are asked for a starting customer number, and the program returns information on all customers with the number you provide, or higher, as shown below:



```
Finding out range of customer records..

Select a starting customer number between    1 and    5: 3

   3: Betty              Falcon             New
   4: Sally              Snead              New
   5: John               Jones              New
press Enter to Exit
```

Note: The behavior and duration of a cursor can be affected by the state of transaction processing for your data source. For example, many databases do not have cursors that maintain their state across a commit or rollback. Please refer to you database documentation for more details.

5.6.1 SET ROWCOUNT Statement

AcuSQL supports the SQL Server-specific SET ROWCOUNT statement. SET ROWCOUNT affects all SELECT statemets made on the connection, and remains in effect until the connection ends or a new *rowcount* to “0” to turn off this option.

5.6.2 Cursors

A cursor is an embedded SQL query that returns multiple rows. The cursor identifies the current row from a SELECT operation that returns multiple rows.

When you work with a cursor, you:

1. DECLARE the cursor, which identifies it to the program.
2. OPEN the cursor, which starts the SELECT process.
3. FETCH until not found, which positions the cursor at each row that meets the conditions set in the SELECT statement and INSERTs the value as designated in the SELECT statement.
4. CLOSE the cursor, which ends the SELECT operation and releases any resources that had been assigned to the cursor.

5.6.3 Declaring a Cursor

Essentially, a cursor is an SQL statement, introduced with a DECLARE statement in the Procedure Division or the Data Division. The cursor does not start the query; the cursor simply identifies the query in the program.

Syntax

```
DECLARE cursorname CURSOR FOR SELECT_statement
```

where *cursorname* is the name you assign to the cursor and *SELECT_statement* is a SELECT statement that returns multiple rows.

Example

The program “select3.sqb” contains the following code:

```
EXEC SQL
   DECLARE COBCUR1 CURSOR FOR
       SELECT
            C_NUMBER, C_FIRST_NAME, C_LAST_NAME,
            TO_CHAR, C_INFO
       FROM CUSTOMER
       WHERE C_NUMBER >= :C_NUMBER
END-EXEC.
```

Here COBCUR1 is the name of the cursor. The operator in the WHERE clause indicates the possibility that more than one row will be returned.

5.6.4 Opening a Cursor

To start the query, you OPEN the cursor.

Syntax

```
EXEC SQL
    OPEN cursorname
END-EXEC.
```

Example

```
EXEC SQL
    OPEN COBCUR1
END-EXEC.
```

5.6.5 FETCH Statement

The FETCH statement positions the cursor at the start of the first row, takes the values returned by the query, and assigns these values to the appropriate host variables. Each time a FETCH is performed, the cursor moves to the next row of results and again assigns the values to host variables.

Syntax

```
EXEC SQL
    FETCH cursorname INTO :host-variable
END-EXEC.
```

Example

The program “select3.sqb” contains the following code:

```
EXEC SQL
    FETCH COBCUR1 INTO :C_RECORD
END-EXEC.
```

This positions the cursor, COBCUR1, at the beginning of the table of results returned by the query. The values returned by the query are FETCHed and placed INTO the group item host variable :C_RECORD. FETCH, then, is similar to a SELECT or INSERT operation, except that SELECT applies to results that return a single row and FETCH applies to results that return multiple rows.

5.6.6 Closing a Cursor

When the cursor has completed its work, usually by encountering no more data, you must CLOSE it.

Syntax

```
EXEC SQL
    CLOSE cursorname
END-EXEC.
```

Example

```
EXEC SQL
    CLOSE COBCUR1
END-EXEC.
```

5.6.7 Putting It All Together

In this program, which builds upon the previous programs, the following lines of code:

```
EXEC SQL
   DECLARE COBCUR1 CURSOR FOR
       SELECT
            C_NUMBER, C_FIRST_NAME, C_LAST_NAME,
            TO_CHAR, C_INFO
       FROM CUSTOMER
       WHERE C_NUMBER >= :C-NUMBER
END-EXEC.
```

```
EXEC SQL
   OPEN COBCUR1
END-EXEC.

perform until SQLCODE not equal 0

    EXEC SQL
         FETCH COBCUR1 INTO :C-RECORD
    END-EXEC
    IF sqlcode EQUAL 0
         display c-number, ": ",c-first-name,
                 c-last-name, c-info
    end-if
end-perform.

EXEC SQL
    CLOSE COBCUR1
END-EXEC.
```

cause the program to do the following:

1. Identify the cursor: for all customers numbers greater than or equal to that supplied by the user, retrieve the values of C_NUMBER, C_FIRST_NAME, C_LAST_NAME, C_BIRTHDAY, and C_INFO and place them in a table of results.
2. Open the cursor.
3. Fetch the values of the indicated variables and assign them to the group item host variable :C_RECORD.
4. If there are no errors or warnings, display the values of the variables.
5. Do this until the program internally indicates that there are no more rows left in the table of results.
6. Close the cursor.

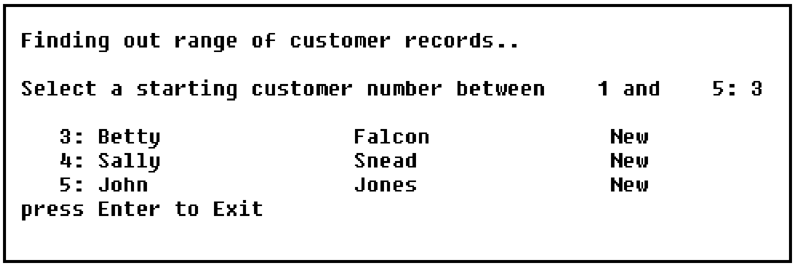
5.7 Updating Data – update.sqb

The “update.sqb” program uses a cursor to select rows from a table and update information in the database. The program demonstrates

- the FOR UPDATE clause when declaring a cursor
- positioned updates
- the SET clause

Note: This sample is not supported by MySQL, because it does not support updatable cursors or implement the “UPDATE WHERE CURRENT OF” and “DELETE WHERE CURRENT OF” phrases. Please refer to the MySQL documentation for any changes in database support of these features.

Run the “update” program to change information for Sam and Sally Snead in the sample data source you created with the “create” program.



```
Finding out range of customer records..

Select a starting customer number between    1 and    5: 3

   3: Betty               Falcon              New
   4: Sally               Snead               New
   5: John                Jones               New
press Enter to Exit
```

5.7.1 FOR UPDATE Clause

You must indicate in the program that the cursor you are declaring may be used to update data in the database table. Programs operate more efficiently if the purpose of the cursor is identified: to only read (SELECT) or to write (INSERT or UPDATE) as well. Remember that you can declare the cursor in either the Procedure or Data Divisions.

Syntax

```
EXEC SQL
    DECLARE cursorname FOR SELECT_statement
    FOR UPDATE
END-EXEC.
```

Example

```
EXEC SQL
    DECLARE COBCUR1 CURSOR FOR
    SELECT C_FIRST_NAME, C_LAST_NAME
        FROM CUSTOMER
        WHERE C_LAST_NAME = 'Snead'
    FOR UPDATE
END-EXEC.
```

This code opens a cursor that contains a query to SELECT the C_FIRST_NAME and C_LAST_NAME fields for customer "Snead." It also indicates, by means of the FOR UPDATE clause, that this information will change and the cursor will write to the database. If you are working in a transaction environment, declaring a cursor FOR UPDATE may start transaction logging or may impose locks on the database table.

5.7.2 The SET Clause

Use the SET clause to modify current values at the location pointed to by the cursor.

Syntax

```
EXEC SQL
    UPDATE tablename SET field = 'new_value'
...
END-EXEC
```

where

tablename is the name of the database table containing the data you want to update.

new_value is the new value you are assigning to that field.

Example

```
EXEC SQL
    UPDATE CUSTOMER SET C_INFO = 'Revised'
...
END-EXEC
```

This tells the “update” program to change the value of C_INFO in the “customer” table to “Revised.”

5.7.3 Positioning the Cursor for the UPDATE

How does the program know which row in the table to update?

In section 5.7.1, you saw that when the cursor was declared, it contained a WHERE condition to indicate which rows were affected by the SQL query. The WHERE CURRENT OF clause uses this information to position the cursor. This is called a *positioned update*.

Syntax

```
EXEC SQL
    UPDATE tablename SET field = 'new_value'
    WHERE CURRENT OF cursorname
END-EXEC
```

where *cursorname* is the name of the cursor containing the SQL query.

Example

```
EXEC SQL
    UPDATE CUSTOMER SET C_INFO = 'Revised'
    WHERE CURRENT OF COBCUR1
END-EXEC
```

In section 5.7.1, a cursor was declared to select the C_FIRST_NAME and C_LAST_NAME fields from the “customer” table in those instances where C_LAST_NAME was equal to “Snead.”

Now, when the cursor gets to a row that meets this condition, the value of the C_INFO field will be updated to “Revised”.

Note: You can also use cursors to perform positioned deletes in AcuSQL. See any commercially available SQL text for information on positioned deletes.

5.7.4 Putting It All Together

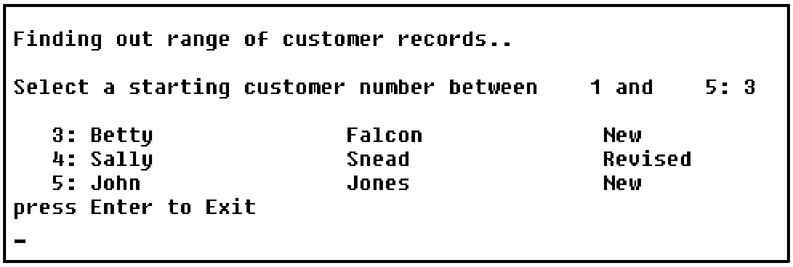
This section showed how to use a cursor to update a table in the database. The following code:

```
    EXEC SQL
        DECLARE COBCUR1 CURSOR FOR
        SELECT C_FIRST_NAME, C_LAST_NAME
            FROM CUSTOMER
            WHERE C_LAST_NAME = 'Snead'
        FOR UPDATE
    END-EXEC.

    EXEC SQL
        OPEN COBCUR1
    END-EXEC.

perform until SQLCODE not equal 0

    EXEC SQL
        FETCH COBCUR1
        INTO :C-FIRST-NAME, :C-LAST-NAME
    END-EXEC
    IF sqlcode EQUAL 0

        DISPLAY "Updating " C-FIRST-NAME , C-LAST-NAME
        EXEC SQL
            UPDATE CUSTOMER SET C_INFO = 'Revised'
            WHERE CURRENT OF COBCUR1
        END-EXEC
    end-if
end-perform.

EXEC SQL
    CLOSE COBCUR1
END-EXEC.
```

instructs the program to perform these steps:

1. Declare a cursor to SELECT the values of C_FIRST_NAME and C_LAST_NAME from the "customer" table in those rows where C_LAST-NAME equals "Snead". These rows will be updated.
2. Open the cursor.
3. Fetch the values and insert them into the host variables :C_FIRST_NAME and :C_LAST-NAME.
4. If the program doesn't encounter an error condition, display a message indicating that the fields are being updated.
5. Change the value of the C_INFO field to "Revised" for those rows that meet the condition set in the cursor (C_LAST_NAME equals "Snead").
6. Close the cursor.

The figure in section 5.7 showed the output of the "update" program. If you want, run the "select3" program to see the contents of the updated fields.



```
Finding out range of customer records..

Select a starting customer number between    1 and    5: 3

   3: Betty                 Falcon              New
   4: Sally                 Snead               Revised
   5: John                  Jones               New
press Enter to Exit
_
```

5.8 Full SQL Program

The following program performs a simple ESQL CONNECT, DROP, CREATE, INSERT, SELECT, UPDATE, DELETE and DISCONNECT statement.

```
IDENTIFICATION DIVISION.
       PROGRAM-ID.                          t006146.
```

```
*****************************************************************
       ENVIRONMENT DIVISION.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
           SELECT CUSTFILE
               ASSIGN TO "customer"
               ORGANIZATION IS LINE SEQUENTIAL
               FILE STATUS IS CUSTFILE-STATUS.

       DATA DIVISION.
       FILE SECTION.
       FD  CUSTFILE.
       01  CUST-RECORD.
           05  CUST-NUMBER                     PIC 9(3).
           05  CUST-FIRST-NAME                 PIC X(20).
           05  CUST-LAST-NAME                  PIC X(20).
           05  CUST-BIRTHDAY.
              07 SYEAR                         PIC X(4).
              07 SMONTH                        PIC X(2).
              07 SDAY                          PIC X(2).

       WORKING-STORAGE SECTION.
       01  CUSTFILE-STATUS                     PIC XX.

       EXEC SQL INCLUDE SQLCA END-EXEC.

       EXEC SQL BEGIN DECLARE SECTION END-EXEC.
       01  C-RECORD.
           05  C-NUMBER                        PIC S9(3) COMP-5.
           05  C-FIRST-NAME                    PIC X(20).
           05  C-LAST-NAME                     PIC X(20).
           05  C-BIRTHDAY                      PIC X(10).
           05  C-INFO                          PIC X(10).

       01  servername                          pic x(30).
       01  userid                              pic x(8).
       01  passwd.
          49 passwd-length                   PIC s9(4) comp-5 value 0.
           49 passwd-name                      PIC x(18).
       EXEC SQL END DECLARE SECTION END-EXEC.


       01  pyyyymmdd.
           03 syear                             pic 9(4).
           03 smonth                            pic 9(2).
           03 sday                              pic 9(2).

       01  e-yyyymmdd.
           03 syear                             pic 9(4).
           03 filler                            pic x value "-".
           03 smonth                            pic 9(2).
           03 filler                            pic x value "-".
           03 sday                              pic 9(2).


 ******************************************************************
        PROCEDURE DIVISION.
        Main Section.
            EXEC SQL WHENEVER SQLERROR GO TO Error-Exit END-EXEC.
            PERFORM initialization-routine.
            PERFORM make-connection.
            PERFORM drop-table.
            PERFORM create-table.
            PERFORM load-table
            DISPLAY "Table created with 15 rows."
            DISPLAY "Press Enter to retrieve and display these rows."
            ACCEPT OMITTED.
            PERFORM list-rows.
            DISPLAY "Press Enter to perform a UPDATE on table row."
            ACCEPT OMITTED.
            PERFORM update-row.
            DISPLAY "Press Enter to perform to DISPLAY updated list."
            ACCEPT OMITTED.
            PERFORM list-rows.
            DISPLAY "Press Enter to DELETE row 15."
            ACCEPT OMITTED.
            PERFORM delete-row.
            DISPLAY "Press Enter to display all remaining rows."
            ACCEPT OMITTED.
            PERFORM list-rows.
            DISPLAY "Press Enter to DISCONNECT and Exit program."
            PERFORM disconnect-connection.
            ACCEPT OMITTED.
            STOP RUN.

        drop-table.
            DISPLAY "Dropping existing table if it exists...".
            EXEC SQL WHENEVER SQLERROR CONTINUE END-EXEC.
            EXEC SQL
                DROP TABLE CUSTOMER
            END-EXEC.
      * Ignore SQLCODE as we do not expect table to exist at this point
            EXEC SQL WHENEVER SQLERROR GO TO Error-Exit END-EXEC.

        create-table.
            DISPLAY "Creating CUSTOMER table...".
            EXEC SQL
               CREATE TABLE CUSTOMER (
                C_NUMBER        INTEGER NOT NULL,
                C_FIRST_NAME    CHAR(20),
                C_LAST_NAME     CHAR(20),
                C_BIRTHDAY      DATETIME,
                C_INFO          CHAR(10),
                PRIMARY KEY     (C_NUMBER))
        END-EXEC.
        IF SQLCODE < 0 PERFORM error-exit.

    load-table.
        DISPLAY "Loading CUSTOMER table with data...".
        OPEN INPUT CUSTFILE.
        PERFORM UNTIL CUSTFILE-STATUS = "10"
            READ CUSTFILE NEXT RECORD
                AT END
                    CONTINUE
                NOT AT END
                    PERFORM insert-record
            END-READ
        END-PERFORM.
        CLOSE CUSTFILE.

    insert-record.
        MOVE CUST-NUMBER TO  C-NUMBER.
        MOVE CUST-FIRST-NAME TO C-FIRST-NAME.
        MOVE CUST-LAST-NAME TO C-LAST-NAME.
        MOVE "New" TO C-INFO.

        MOVE CORRESPONDING CUST-BIRTHDAY TO E-YYYYMMDD.
        MOVE E-YYYYMMDD TO C-BIRTHDAY.

        EXEC SQL
            INSERT INTO CUSTOMER VALUES
                (:C-NUMBER, :C-FIRST-NAME,
                 :C-LAST-NAME, :C-BIRTHDAY, :C-INFO)
        END-EXEC.
        MOVE LOW-VALUES TO CUST-RECORD.
        MOVE LOW-VALUES TO C-RECORD.


    initialization-routine.
        DISPLAY "Enter MS SQL Servername or ODBC DSN:", no.
        ACCEPT servername.
        DISPLAY "Enter your user id (default none): ", no.
        ACCEPT userid.
        DISPLAY "Enter your password : ", no.
        ACCEPT passwd-name.
        IF userid = spaces then
            DISPLAY "Using NT Authentication...".
    INSPECT passwd-name TALLYING passwd-length FOR CHARACTERS
       BEFORE INITIAL " ".

make-connection.
    EXEC SQL CONNECT TO :servername as C1
            USER :userid USING :passwd
    END-EXEC.
    IF SQLCODE < 0 PERFORM error-exit.
    IF SQLCODE = 0 DISPLAY "Connection successful".

disconnect-connection.
    EXEC SQL COMMIT END-EXEC.
    EXEC SQL DISCONNECT ALL END-EXEC.
    IF SQLCODE < 0 PERFORM error-exit.

Error-Exit.
    EXEC SQL WHENEVER SQLERROR CONTINUE END-EXEC.
    DISPLAY "SQL Error: SQLCODE  " SQLCODE of SQLCA.
    DISPLAY "           SQLSTATE " SQLSTATE of SQLCA.
    DISPLAY SQLERRMC OF SQLCA.
    ACCEPT OMITTED.
    EXEC SQL DISCONNECT ALL END-EXEC.
    STOP RUN.

list-rows.
    EXEC SQL
       DECLARE COBCUR1 CURSOR FOR
           SELECT
                C_NUMBER,
                C_FIRST_NAME,
                C_LAST_NAME,
                C_BIRTHDAY,
                C_INFO
           FROM CUSTOMER
                WHERE C_NUMBER > 0
    END-EXEC.

    EXEC SQL
        OPEN COBCUR1
    END-EXEC.

    PERFORM UNTIL SQLCODE NOT EQUAL 0

       EXEC SQL
            FETCH COBCUR1 INTO :C-RECORD
       END-EXEC
       IF SQLCODE EQUAL 0
            DISPLAY c-number, ": ",c-first-name,
                    c-last-name, c-info
       END-IF
    END-PERFORM.

    EXEC SQL
        CLOSE COBCUR1
    END-EXEC.

update-row.
    EXEC SQL
       DECLARE COBCUR2 CURSOR FOR
       SELECT C_FIRST_NAME,
                 C_LAST_NAME
            FROM CUSTOMER
                WHERE C_NUMBER = 002
            FOR UPDATE
    END-EXEC.

    EXEC SQL
        OPEN COBCUR2
    END-EXEC.

    PERFORM UNTIL SQLCODE NOT EQUAL 0

       EXEC SQL
            FETCH COBCUR2
            INTO :C-FIRST-NAME,
                 :C-LAST-NAME
       END-EXEC
       IF SQLCODE EQUAL 0

            DISPLAY "Updating " C-FIRST-NAME , C-LAST-NAME
            EXEC SQL
               UPDATE CUSTOMER SET C_INFO = 'Revised'
                  WHERE CURRENT OF COBCUR2
            END-EXEC
       END-IF
    END-PERFORM.

    EXEC SQL
        CLOSE COBCUR2
    END-EXEC.

delete-row.
    DISPLAY "Deleting row 15..."
    EXEC SQL
         DELETE FROM CUSTOMER
                WHERE C_FIRST_NAME = 'OOOOO'
    END-EXEC
```

Glossary of Terms

cursor

An embedded SQL query that returns multiple rows. Cursors can be declared in either the Procedure Division or the Data Division.

Data Definition Language (DDL)

The set of SQL statements related to the structure of the database, such as creating tables. This book contains examples of the CREATE statement, an element of SQL's Data Definition Language. Data Definition Language is also referred to as DDL. Compare this with *Data Manipulation Language*.

Data Manipulation Language (DML)

The set of SQL statements that pertain to manipulating the data. The structure of the database, such as the number of tables or the number of columns in a table, remains the same. Data Manipulation Language is also called DML. Compare this with *Data Definition Language*.

dynamic SQL

A program in which user input determines how the program interacts with the database. Put another way, the SQL statements are not part of the program when it is launched; instead they are built as the program runs. Compare this with static SQL.

ESQL

Embedded Structured Query Language. Commands that enable non-SQL programs such as COBOL programs to communicate with SQL databases. These commands reside in the program along with commands in the host, in this case COBOL, language.

host variable

Any data item defined in an SQL DECLARE section of Working-Storage. Host variables are often used in SQL statements and must be preceded with a colon (“:”).

intermediate file

An AcuSQL system file created between pre-compilation and compilation. Useful for debugging purposes. Intermediate files can be opened into the Code Editor; however, you should not modify the intermediate files in any way. Intermediate files created by AcuSQL have the extension “.asq”.

ODBC

Open Database Connectivity. A standard protocol used to facilitate communications between Windows-based applications and data sources.

positioned update

An update using a cursor. Use the WHERE CURRENT OF clause to position the cursor at the proper row(s).

pre-compiler

A compiler that is run prior to a standard program compiler to perform preparatory translation tasks. AcuSQL is an ESQL pre-compiler. It is run before the standard ACUCOBOL-GT compiler to translate ESQL commands into ODBC API calls.

schema

A user’s area of the database. Generally speaking, users can add tables to or delete tables from only their area of the database unless they use a qualified name for the table.

static SQL

A program where access to the database has been predetermined by the programmer and any input from the user will not change this access pattern. Put another way, all SQL statements are already part of the program when it is executed. Compare this with *dynamic SQL*.

Index

A

Access data type mapping 2-11, 2-12
ACUSQL_DATASET configuration variable 3-13
ACUSQL_INCLUDE configuration variable 3-13
ACUSQL_PASSWORD configuration variable 3-13
ACUSQL_RUNTIME_DLL configuration variable 4-3
ACUSQL_USE_CONCURRENT configuration variable 4-6
ACUSQL_USER configuration variable 3-14
adding database tables 5-5
application execution 4-2
ASQL_BUFFER_SIZE configuration variable 4-4
asqlsrvr.dll 4-3
autocommit, turning off 4-6

C

caching rows 4-4
case sensitivity for SQL objects 2-4
CHAR data type
    DB2 2-9
    SQL Server 2-10
checking syntax 2-19
CLOSE statement 5-18
closing cursors 5-18
coding area for EXEC SQL statements 2-4
colons in host variables 2-8
commas in list items 2-5
compiler options
    -Pk DB2 2-3
    -Pk mssql 2-3

configuration variables
- ACUSQL_DATASET 3-13
- ACUSQL_INCLUDE 3-13
- ACUSQL_NO_AUTOCOMMIT 4-6
- ACUSQL_PASSWORD 3-13
- ACUSQL_RUNTIME_DLL 4-3
- ACUSQL_USE_CONCURRENT 4-6
- ACUSQL_USER 3-14
- ASQL_BUFFER_SIZE 4-4
- defining 5-10

CONNECT RESET statement 2-19
CONNECT statement 2-18
create.sqb sample program 5-3
creating database tables 5-3, 5-5
current connection 2-19
cursors
- caching rows 4-4
- closing 5-18
- declared in Data Division section 2-6
- declared in Procedure Division section 2-13
- declaring 5-16
- defined 5-16
- opening 5-17

D

Data Definition Language 5-5
Data Division statements 2-5
Data Manipulation Language 5-6
data types, matching SQL to COBOL 2-9
database tables
- creating 5-3
- updating data in 5-21

DATE data type, DB2 2-9
DATETIME data type, SQL Server 2-10
DB2

data type mapping 2-9
demonstration program 1-12
host environments 1-5
middleware 1-5, 1-7
reserved word list 3-7, 3-11
DB2 Connect 1-5, 1-7
testing 4-2
DDL 5-5
debugging programs 4-7
DECIMAL data type
DB2 2-9
SQL Server 2-10
declaring cursors 5-16
FOR UPDATE clause 5-21
in Data Division section 2-6
in Procedure Division section 2-13
defining variables 5-10
deleting database tables 5-5
demonstration programs 1-12
DISCONNECT statement 2-19
DLLs, naming 4-2
DOUBLE data type, DB2 2-9
DOUBLE PRECISION data type, SQL Server 2-10

E

error handling 2-7
WHENEVER statement 5-4
error messages, mapping to SQLSTATE values 4-8
errors, pre-compiler 3-14
ESQL 2-2
translation 3-2
esqllib.dll 4-3
EXEC SQL statement 3-2

F

FETCH statement 5-17
File Compiler Option dialog 3-5
file tracing 4-7
FOR UPDATE clause 5-21

G

group items, storing values in 5-14

H

host variables 2-8, 5-10
  in the CONNECT statement 2-18

I

IBM SQL, working with 2-3
INCLUDE files 2-6
  SQLCA 2-7
  SQLDA 2-8
indicator variables 2-24
  input 2-24
  output 2-25
INSERT statement 5-6
installation
  UNIX 1-8
  Windows 1-7
INTEGER data type
  DB2 2-9
  SQL Server 2-10
INTO clause 5-10
  group items in 5-14

L

large object data types (LOBs) 2-2, 2-23
list items 2-5
LONG VARCHAR data type, DB2 2-9

M

mapping SQL Server errors to SQLSTATE values 4-8
MAX function 5-12
Microsoft SQL Server data type mapping 2-10
Microsoft SQL Server databases, working with 2-3
MIN function 5-12
MySQL Connector/ODBC 1-6
MySQL Data Type 2-12
MySQL Database Server 1-6
MySQL requirements 1-6

N

nested transactions 2-4
NULL values 2-23

O

ODBC API 1-5
OPEN statement 5-17
opening cursors 5-17

P

-Pk DB2 compiler option 2-3
-Pk mssql compiler option 2-3
positioned updates 5-22

pre-compiler
- errors 3-14
- invoking 3-4
- invoking from ACUCOBOL-GT 3-5
- options 3-5
- use in AcuBench 3-5

pre-compiling
- DB2 syntax 2-3
- SQL Server syntax 2-3

Procedure Division 2-13

Q

query results, multiple rows 5-15

quotation marks in string delimiters 2-5

R

REAL data type
- DB2 2-9
- SQL Server 2-10

relaxed syntax checking 2-19, 3-8, 3-11

relinking
- pre-compiler 1-11
- runtime 1-9

reserved word list 3-7, 3-11
- to standard output 3-9, 3-12

S

sample programs 5-2
- create.sqb 5-3
- select1.sqb 5-8
- select2.sqb 5-14
- select3.sqb 5-15

- update.sqb 5-20

SELECT statement 5-8, 5-9
select1.sqb sample program 5-8
select2.sqb sample program 5-14
select3.sqb sample program 5-15
SET clause 5-21
SMALLINT data type, SQL Server 2-10
source code preparation 2-2
SQL
- error messages 4-8
- relaxed syntax checking 3-8, 3-11
- static 5-3

SQL Communications Area 2-7
SQL Descriptor Area 2-8
SQL Server data type mapping 2-10
SQL syntax
- CLOSE statement 5-18
- cursors 5-16
- declaring cursors 5-16
- FETCH statement 5-17
- FOR UPDATE clause 5-21
- INSERT statement 5-6
- INTO clause 5-10, 5-14
- MAX function 5-12
- MIN function 5-12
- OPEN statement 5-17
- SELECT statement 5-8, 5-9
- SET clause 5-21
- WHENEVER directive 5-4
- WHERE clause 5-12
- WHERE CURRENT OF clause 5-22

SQLCA 2-7
SQLCODE 4-8
SQLDA 2-8
SQLSTATE values, mapping error messages 4-8
statements not supported 4-3
static SQL 5-3

strict syntax checking 2-19
string delimiters 2-5
syntax
    case sensitivity 2-4
    coding area 2-4
    colons 2-8
    commas 2-5
    host variables 2-8
    quotation marks in string deliminters 2-5
syntax checking 2-19
    designated reserved word list 3-7, 3-11
    direct 3-3, 3-6, 3-10
    modes 3-3
    relaxed 3-3, 3-8, 3-11
    strict 2-19
system requirements 1-4

T

tables
    creating in databases 5-5
    deleting from databases 5-5
TIME data type, DB2 2-9
TIMESTAMP data type, DB2 2-9
transactions
    in Microsoft SQL Server 2-3
    nested 2-4

U

UNIX, installation 1-8
unsupported statements 4-3
update.sqb sample program 5-20
updating data 5-21
    positioning the cursor 5-22
    SET clause 5-21

V

VARCHAR data type
    DB2 2-9
    SQL Server 2-10
variables. *See* configuration variables

W

WHENEVER directive 5-4
WHERE clause 5-12
WHERE CURRENT OF clause 5-22
Working-Storage section 5-10