

AD/ADVANTAGE

MANTIS for Windows SQL Support for
SUPRA Programming Guide

P19-2307-01

AD/Advantage[®] MANTIS for Windows SQL Support for SUPRA Programming Guide

Publication Number P19-2307-01

© 1989, 1998, 1999 Cincom Systems, Inc.
All rights reserved

This document contains unpublished, confidential, and proprietary information of Cincom. No disclosure or use of any portion of the contents of these materials may be made without the express written consent of Cincom.

The following are trademarks, registered trademarks, or service marks of Cincom Systems, Inc.:

AD/Advantage[®]
AuroraDS[®]
CINCOM[®]
CINCOM SYSTEMS[®]



CONTROL[™]
CONTROL:Financial[™]
CONTROL:Manufacturing[™]
CPCS[™]

DOCVIEW[™]
EAGLE ADVANTAGESM
Enterprise Analyst Series[™]
MANTEXT[®]
MANTIS[®]
META*STAR[®]
M/Archive[™]
M/Exchange[™]
M/Graph[™]
M/Post[™]

M/Spell[™]
M/Text[™]
NORMAL[®]
PC CONTACT[®]
SPECTRA[™]
SUPRA[®]
SUPRA[®] Server
The Smart Choice[®]
TIS/XA[™]
TOTAL[®]
TOTAL FrameWork[®]

All other trademarks are trademarks or registered trademarks of:

Acucobol, Inc.
AT&T
Data General Corporation
Digital Equipment Corporation
Gupta Technologies, Inc.
International Business Machines Corporation

JSB Computer Systems Ltd.
Micro Focus, Inc.
Microsoft Corporation
Systems Center, Inc.
TechGnosis International, Inc.
UNIX System Laboratories, Inc.

or of their respective companies.

Cincom Systems, Inc.
55 Merchant Street
Cincinnati, OH 45246-3732
U.S.A.

PHONE: (513) 612-2300
FAX: (513) 612-2000
WORLD WIDE WEB: <http://www.cincom.com>

Attention:

Some Cincom products, programs, or services referred to in this publication may not be available in all countries in which Cincom does business. Additionally, some Cincom products, programs, or services may not be available for all operating systems or all product releases. Contact your Cincom representative to be certain the items are available to you.

Release information for this manual

AD/Advantage MANTIS for Windows SQL Support for SUPRA Programming Guide, P19-2307-01, is dated November 30, 1998. This document supports Release 2.2.01 and higher of MANTIS for Windows.

We welcome your comments

We encourage critiques concerning the technical content and organization of this manual. A [Reader Comment Sheet](#) is included at the end of the manual for your convenience.

Cincom Technical Support for AD/Advantage

FAX: (513) 612-2000
Attn: MANTIS Support

E-mail: helpna@cincom.com

Phone: 1-800-727-3525

Mail: Cincom Systems, Inc.
Attn: MANTIS Support
55 Merchant Street
Cincinnati, OH 45246-3732
U.S.A.

Contents

About this book	vii
Using this document	vii
Document organization	viii
Conventions	ix
MANTIS documentation series	xii
Overview	13
Embedding SQL statements in MANTIS SQL Support	14
SQL in MANTIS SQL Support versus SQL in C	15
Embedding SQL statements in MANTIS programs	17
Embedding rules	17
Host variables	21
Referencing values in a MANTIS array	22
MANTIS versus SQL data types	22
Indicator variables	23
Data conversion between MANTIS SQL Support and SUPRA	24
Programming considerations	25
EXEC_SQL-END	27
Cursors, statements, and SQLDAs	28
Connection to SUPRA and multiple session support	29
Disconnection from SUPRA	30
MANTIS EXEC_SQL statement	31
SQL WHENEVER statement	33
Declarative versus interpretive WHENEVER statements	38
WHENEVER statement	38
SQL COMMIT/ROLLBACK statements	39
SQL SET DBNAME statement	39

SQLCA in MANTIS SQL Support.....	40
SQLCA syntax.....	40
SQLCA elements	42
COMMIT and ROLLBACK in SUPRA and COMMIT and RESET in MANTIS SQL Support.....	45
Error messages.....	46
Dynamic SQL in MANTIS SQL Support	47
Execute an SQL statement dynamically	48
SQLDA structure	50
Allocate an SQLDA	52
Deallocate an SQLDA	53
Move data from your program into an SQLDA header element.....	54
Move data from your program into an SQLDA repeating element.....	57
Move data from an SQLDA header element into your program.....	60
Move data from an SQLDA repeating element into your program.....	61
Sample MANTIS SQL programs	65
Static insert routine	66
Dynamic insert routine	68
Static update routine	70
Dynamic update routine	71
Static select routine.....	72
Dynamic select routine.....	73
Static delete routine	75
Dynamic delete routine	75
SQL query function	76
Dynamic column select	79
Features not supported	81
MANTIS SQL Support error messages	83
Differences: MANTIS SQL Support versus SQL in C; MANTIS versus SQL	99
SQL in MANTIS SQL Support versus SQL in C	99
MANTIS versus SQL.....	100
Index	101

About this book

Using this document

MANTIS[®] is an application development system that consists of design facilities (e.g., screens and files) and a programming language. MANTIS SQL Support is used to create MANTIS applications that access SUPRA. This manual explains how to code MANTIS SQL programs. Beginning with an overview, the manual discusses the rules for combining MANTIS and SQL, programming considerations, and dynamic MANTIS SQL. Sample programs are provided.

This manual supplements *MANTIS for Windows Facilities Reference Manual*, P19-2301, *MANTIS for Windows Language Reference Manual*, P19-2302, and documentation on the SUPRA database system. System administration considerations for MANTIS SQL Support are in *MANTIS for Windows SQL Support for SUPRA Administration Guide*, P19-2308.

Document organization

The information in this manual is organized as follows:

Chapter 1—Overview

Provides an overview of MANTIS and MANTIS SQL Support.

Chapter 2—Embedding SQL statements in MANTIS programs

Describes the rules you must follow when embedding SQL statements in a MANTIS program. It also explains how to reference host variables and MANTIS entities, and how SUPRA converts data values between SQL and MANTIS.

Chapter 3—Programming considerations

Discusses implications related to the interpretive nature of MANTIS SQL Support.

Chapter 4—Dynamic SQL in MANTIS SQL Support

Discusses how dynamic SQL works in MANTIS SQL Support.

Appendix A—Sample MANTIS SQL programs

Provides listings of sample MANTIS SQL programs.

Appendix B—Features not supported

Lists features that are not supported by MANTIS for Windows.

Appendix C—MANTIS SQL Support error messages

Lists MANTIS SQL Support messages, and provides a suggested corrective action.

Appendix D—Differences: MANTIS SQL Support versus SQL in C; MANTIS versus SQL

Summarizes the differences between SQL in MANTIS SQL Support and SQL in other languages.

Index

Conventions

The following table describes the conventions used in this document series:

Convention	Description	Example
Constant width type	Represents screen images and segments of code.	Screen Design Facility GET NAME LAST INSERT ADDRESS
Slashed b (<i>b</i>)	Indicates a space (blank). The example indicates that a password can have a trailing blank.	WRITEPASS b
Brackets []	Indicate optional selection of parameters. (Do not attempt to enter brackets or to stack parameters.) Brackets indicate one of the following situations: A single item enclosed by brackets indicates that the item is optional and can be omitted. The example indicates that you can optionally enter a program name. Stacked items enclosed by brackets represent optional alternatives, one of which can be selected. The example indicates that you can optionally enter NEXT, PRIOR, FIRST, or LAST. (NEXT is underlined to indicate that it is the default.)	COMPOSE [<i>program-name</i>] <u>NEXT</u> PRIOR FIRST LAST

Convention	Description	Example
Braces { }	<p>Indicate selection of parameters. (Do not attempt to enter braces or to stack parameters.) Braces surrounding stacked items represent alternatives, one of which you must select.</p> <p>The example indicates that you must enter FIRST, LAST, or a value for <i>begin</i>.</p>	<pre>{FIRST begin LAST}</pre>
<u>Underlining</u>	<p>Indicates the default value supplied when you omit a parameter.</p> <p>The example indicates that if you do not specify ON, OFF, or a row and column destination, the system defaults to ON.</p> <p>Underlining also indicates an allowable abbreviation or the shortest truncation allowed.</p> <p>The example indicates that you can enter either PRO or PROTECTED.</p>	<pre>SCROLL [ON OFF [row] [,col]]</pre> <p><u>PROTECTED</u></p>
Ellipsis points...	<p>Indicate that the preceding item can be repeated.</p> <p>The example indicates that you can enter (A), (A,B), (A,B,C), or some other argument in the same pattern.</p>	<pre>(argument, ...)</pre>
SMALL CAPS	<p>Represent a keystroke. Multiple keystrokes are hyphenated.</p>	<pre>ALT-TAB</pre>

Convention	Description	Example
UPPERCASE	Indicates MANTIS reserved words. You must enter them exactly as they appear. The example indicates that you must enter CONVERSE exactly as it appears.	CONVERSE <i>name</i>
lowercase	Indicates generic names of parameters for which you supply specific values as needed.	COMPOSE [<i>program-name</i>]
<i>Italics</i>	Indicate variables you replace with a value, a column name, a file name, and so on. The example indicates that you can supply a name for the program.	COMPOSE [<i>program-name</i>]
Punctuation marks	Indicate required syntax that you must code exactly as presented. () parentheses . period , comma : colon ' ' single quotation marks	[LET] _v $\begin{bmatrix} (i) \\ (i, j) \end{bmatrix}$ [ROUNDED(<i>n</i>)] = <i>e1</i> [, <i>e2</i> , <i>e3</i> ...]

MANTIS documentation series

MANTIS is fourth-generation programming language used for application development. MANTIS is part of AD/Advantage[®], which offers additional tools for application development. The following list shows the manuals offered with MANTIS for Windows, organized by task. You may not have all the manuals that are listed here.

Getting started

- ◆ *MANTIS for Windows Administration Guide*, P19-2304*

General use

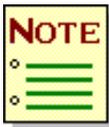
- ◆ *MANTIS for Windows Language Reference Manual*, P19-2302
- ◆ *MANTIS for Windows Facilities Reference Manual*, P19-2301
- ◆ *MANTIS for Windows Quick Reference*, P19-2303

SQL support

- ◆ *MANTIS for Windows SQL Support for SUPRA Programming Guide*, P19-2307
- ◆ *MANTIS for Windows SQL Support for SUPRA Administration Guide*, P19-2308*

Master user tasks

- ◆ *MANTIS for Windows Administration Guide*, P19-2304*
- ◆ *MANTIS for Windows SQL Support for SUPRA Administration Guide*, P19-2308*



Manuals marked with an asterisk (*) are listed twice because you use them for different tasks.



MANTIS educational material is available from your regional Cincom education department.

1

Overview

MANTIS is an application development system for developing, testing, executing, and documenting applications interactively. MANTIS SQL Support is an extended version of MANTIS for Windows that enables you to create MANTIS applications that access SUPRA using SQL. System administration information is provided in *MANTIS for Windows SQL Support for SUPRA Administration Guide*, P19-2308.

The presence of MANTIS SQL Support does not affect non-SQL MANTIS applications. MANTIS SQL Support programs can run side-by-side or in conjunction with non-SQL MANTIS programs, with neither affecting the other.

As each SQL statement is encountered, MANTIS transparently prepares and executes the statement. Data values are implicitly passed between SQL variables (see SALARY in the code sample under “[Embedding SQL statements in MANTIS SQL Support](#)” on page 14) and MANTIS variables (see EMP SAL in the code sample under “[Embedding SQL statements in MANTIS SQL Support](#)” on page 14).

Embedding SQL statements in MANTIS SQL Support

Embedding SQL in MANTIS is similar to embedding SQL statements in other host languages. You embed SQL statements in a MANTIS application program as standard MANTIS comments. The broken vertical bar (|) is the MANTIS comment character. Precede each SQL statement with an EXEC_SQL statement and follow it with an END statement, as shown in the following example (MANTIS automatically sets the indentation level (number of preceding periods (.)) for the EXEC_SQL structure):

```
4580 TEXT EMP NAME(30)
4590 BIG EMP SAL
4600 WHILE SQLCODE<>100
4610 EXEC_SQL
4620 .| SELECT SALARY
4630 .| INTO :EMP SAL
4640 .| FROM EMPLOYEE TABLE
4650 .| WHERE NAME=:EMP NAME
4660 END
4670 DO BONUS ROUTINE
```

MANTIS variables in SQL statements are called host variables. Syntactically, a colon (:) always precedes a host variable in an SQL statement (as in line 4630 above). An input host variable is a MANTIS variable passed to SQL and is used to select, insert, delete, or update data. A MANTIS variable that receives data from SUPRA is called an output host variable. Host variables are also used as parameters of the SQL statements. Optionally, you can specify an indicator variable along with a host variable. SUPRA sets the indicator variable to indicate null values or to signal that a value was truncated. [“Embedding SQL statements in MANTIS programs”](#) on page 17 describes embedding SQL statements in MANTIS programs.

SQL in MANTIS SQL Support versus SQL in C

SQL in MANTIS SQL Support is essentially the same as its implementation in SQL in C. The differences exist because MANTIS is an interpretive rather than a compiled language. These differences are noted in the appropriate chapters of this manual and are summarized in “Differences: MANTIS SQL Support versus SQL in C; MANTIS versus SQL” on page 99.

2

Embedding SQL statements in MANTIS programs

This chapter describes the rules you must follow when embedding SQL statements in a MANTIS program. It also explains how to reference host variables and MANTIS entities, and how SUPRA converts data values between SQL and MANTIS. A general working knowledge of MANTIS, SUPRA, and SQL is assumed. A summary of MANTIS language conventions is in *MANTIS for Windows Language Reference Manual*, P19-2302.

Embedding rules

You embed SQL statements in a MANTIS application program as standard MANTIS comments. The broken vertical bar (!) is the MANTIS comment character. Standard rules apply for using MANTIS comments. Precede each SQL statement with an EXEC_SQL statement and follow it with an END statement. Standard SQL syntax rules apply for all text between the EXEC_SQL and END statements. As the examples show, EXEC_SQL causes the statements that follow it to be indented.

The following rules apply when embedding SQL statements in a MANTIS application program:

- ◆ Only one SQL statement can be within an EXEC_SQL-END block. The following example shows an invalid statement:

```
EXEC_SQL
.| OPEN C1
.| FETCH C1 INTO ...
.| CLOSE C1
END
```

Invalid: Three SQL statements in the EXEC_SQL-END block.

- ◆ Any text between EXEC_SQL and END must be part of an SQL statement and must be preceded by a broken vertical bar (!). Once MANTIS SQL Support encounters a broken vertical bar, it treats the rest of the program line as a single SQL statement. Other MANTIS statements or comments are not permitted. The following examples show invalid statements:

```
EXEC_SQL
.| OPEN C1
.OPENED=TRUE
END
```

Invalid: A statement other than a comment is between EXEC_SQL and END.

```
EXEC_SQL
.| OPEN C1:OPENED=TRUE
END
```

Invalid: A MANTIS statement is appended to a valid SQL statement.

```
EXEC_SQL
.| OPEN C1:|EMPLOYEE CURSOR
END
```

Invalid: A comment is appended to a valid SQL statement.

- ◆ A colon within an EXEC_SQL-END block identifies a MANTIS host variable, not a new statement, as the following example demonstrates:

```
EXEC_SQL
.| FETCH C1 INTO :A
END
```

C1 is an SQL entity; A is a MANTIS host variable.

- ◆ An SQL statement in an EXEC_SQL-END block can be broken into multiple lines, as shown in the following example:

EXEC_SQL	<i>is equivalent to</i>	EXEC_SQL
. OPEN		. OPEN C1
. C1		END
END		

MANTIS reads the text on two consecutive comment lines in an EXEC_SQL-END block as if it were separated by a single blank (one statement). SQL text literals (characters between apostrophes) can not span lines.

- ◆ In an SQL statement, multiple blanks at the beginning or end of an SQL statement, or even spaces between words on the same line, are treated as a single blank. The following example demonstrates how multiple blanks at the beginning of a statement are treated as a single blank:

EXEC_SQL	<i>is equivalent to</i>	EXEC_SQL
. OPEN C1		. OPEN C1
END		END

Multiple spaces between words in statements are compressed, as shown in the following example

EXEC_SQL	<i>is equivalent to</i>	EXEC_SQL
.		. OPEN C1
. OPEN		END
. C1		
.		
.END		

- ◆ An SQL statement on the same line as an EXEC_SQL statement is part of the SQL statement; it is considered to be within the EXEC_SQL-END block. The following example shows a valid statement:

```
EXEC_SQL:| SELECT ...           Valid
.| FROM ...
.| WHERE ...
END
```

- ◆ A MANTIS statement on the same line as the END in an EXEC_SQL-END block is not part of the EXEC_SQL-END block and is not executed, as shown in the following example.

```
EXEC_SQL                               "OPENED=
.| OPEN C1                               TRUE" is
END:OPENED=TRUE                          disregarded.
EXEC_SQL                               A valid comment.
.| OPEN C1
END:| C1 IDENTIFIES TAG FILE ENTRIES
```

This rule is consistent with the rules for using END with MANTIS IF, WHILE, FOR, WHEN, and UNTIL statements. MANTIS comments are permitted.

Host variables

A MANTIS variable that provides input to or receives output from SUPRA is called a host variable. A host variable is identified within an SQL statement by a colon prefix. In the following example, EMPL is a host variable (MANTIS variable):

```
..SMALL EMPL
..EXEC_SQL
...| FETCH CURSOR1 INTO :EMPL
..END
```

Like other MANTIS variables, host variables are implicitly declared when they are first used if they are not explicitly declared before appearing in the EXEC_SQL-END block. Any previously undefined MANTIS variable referred to in an SQL statement is automatically declared as a MANTIS BIG variable. (A MANTIS BIG variable is a numeric floating point variable that holds up to 15 digits.)

..BIG A	<i>is equivalent</i>	EXEC_SQL
..EXEC_SQL	<i>to</i>	FETCH C1 INTO :A
... FETCH C1 INTO :A		END
..END		

If necessary, you can explicitly declare a host variable as a type other than BIG.

Referencing values in a MANTIS array

A host variable can be an element in a MANTIS array. You can use arithmetic expressions and MANTIS functions to specify subscripts of host variables. MANTIS rules apply to subscripting, even though the subscript is in an SQL statement. In the following example, all text following the colon must conform to MANTIS syntax:

```
..SMALL EMPL(20,40)
..EXEC_SQL
...| FETCH ENTRY1 INTO :EMPL(1+N,INT(T))
..END
```



Only the host variable, not other MANTIS variables referred to in subscript expressions, can be prefixed with a colon. In the previous example, the variables N and T are not prefixed with a colon, but are assumed to be MANTIS variables.

MANTIS versus SQL data types

Data sent to and from SUPRA is BIG, or TEXT. Any conversion of data from one data type to another is performed by SUPRA. Consult the table in [“Data conversion between MANTIS SQL Support and SUPRA”](#) on page 24 for a summary of how types are converted. Be sure to note that truncation, overflow, and rounding may occur.

Indicator variables

You can include an indicator variable along with a host variable in SQL statements. As its name implies, the indicator variable indicates whether the host variable contains a real value or is NULL (does not exist). Indicator variables are interpreted as follows:

Value	Meaning
= 0	Host variable contains data.
< 0	Host variable data is NULL (does not exist).
> 0	Host variable contains truncated data. The indicator variable value is the original length of the host variable data.

An indicator variable is prefixed with a colon and immediately follows the corresponding host variable (or subscript expression). In the following example, EMPLIV and NAMEIV are indicator variables:

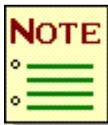
```

..EXEC_SQL: | SELECT EMPLNO, EMPLNA
... | INTO:EMPL(15,3):EMPLIV,:NAME:NAMEIV
... | FROM EMPLOYEES WHERE DEPT=17
..END

```

Like host variables, indicator variables can be explicitly or implicitly defined. Only SMALL and BIG variables can be used as indicator variables. The default in implicit declaration is a MANTIS BIG variable. Variable types are described in [“Data conversion between MANTIS SQL Support and SUPRA”](#) on page 24. Note that indicator values are interpreted by SUPRA SQL as integers whereas they are specified as floating point values in MANTIS. Therefore a value of -0.9 will not specify a NULL value because it is converted to 0 before being interpreted.

When reading data from SUPRA (SELECT/FETCH), you must supply indicator variables for any columns that allow NULL values.



If the column is NULL, the value of the host variable is not defined. Check the value of the indicator before examining the host variable data.

Data conversion between MANTIS SQL Support and SUPRA

Data sent to and from SUPRA is BIG, or TEXT. Any conversion of data from one data type to another is performed by SUPRA. The following table lists permissible data conversions. Check the notes for information about overflow, truncation, and rounding. Any combination of MANTIS and SQL data types not listed below results in a run-time error. Truncation is indicated through the SQLWARN elements in the SQLCA and indicator variables, if used.

SQL data type	MANTIS data type	Notes
0 (Fixed)	BIG	When converting from MANTIS to SQL, overflow may occur.
	SMALL	When converting from MANTIS to SQL, overflow may occur.
1 (Float)	BIG	When converting from MANTIS to SQL, overflow may occur.
	SMALL	When converting from MANTIS to SQL, overflow may occur.
2 (Character)	TEXT	When converting in either direction, truncation may occur.
4 (Date)	TEXT	When converting in either direction, truncation may occur.
5 (Time)	TEXT	When converting in either direction, truncation may occur.
6 (String)	TEXT	When converting in either direction, truncation may occur.

3

Programming considerations

To use MANTIS SQL Support, you simply embed the appropriate SQL statements in your MANTIS application program as standard MANTIS comments, enclosed in EXEC_SQL-END blocks. As each SQL statement is encountered, MANTIS prepares and executes it, in effect performing the same steps (preprocess, compile, link, and run) that are executed with a C program that contains embedded SQL statements. However, unlike C programs, the MANTIS program can be modified, including the SQL statements, and then immediately reexecuted by issuing the RUN command. The fact that MANTIS SQL Support is interpretive has several implications for program design, as discussed in this chapter.

Before you begin writing MANTIS SQL Support programs, you should be aware of the following implications and other programming considerations:

- ◆ MANTIS stores an EXEC_SQL-END block as a single line of text internally and associates the line number of the last program line in the block with this single line. There are minor distinctions between bound and unbound versions of a program. (See “EXEC_SQL-END” on page 27.)
- ◆ The scope of a cursor or SQL statement name is local to a MANTIS program context. Because both are SQL entities and not MANTIS entities, you cannot pass them as parameters or use them in non-SQL MANTIS statements. (See “Cursors, statements, and SQLDAs” on page 28.)

- ◆ The WHENEVER statement in MANTIS SQL Support differs slightly from the WHENEVER statement used in other languages. It has different syntax, defaults, and possible effects (see “[SQL WHENEVER statement](#)” on page 33).
- ◆ Elements in the SQLCA (SQL Communications Area) are accessed through a MANTIS function called SQLCA, rather than as elements of an SQLCA data structure (see “[SQLCA in MANTIS SQL Support](#)” on page 40).
- ◆ The effects of COMMIT and ROLLBACK in SUPRA differ slightly from COMMIT and RESET in MANTIS. In MANTIS SQL Support, an SQL COMMIT WORK commits only SQL updates, and only for the specified SQL session. In MANTIS, COMMIT and RESET commit everything, including SQL (see “[COMMIT and ROLLBACK in SUPRA and COMMIT and RESET in MANTIS SQL Support](#)” on page 45).
- ◆ Error messages can come from different sources: MANTIS SQL Support, the MANTIS nucleus, and SUPRA (see “[Error messages](#)” on page 46).
- ◆ Elements in the SQLDA (SQL Descriptor Area) are accessed through a MANTIS function called SQLDA, rather than as elements of an SQLDA data structure (see “[SQLDA structure](#)” on page 50).

EXEC_SQL-END

The EXEC_SQL-END block may continue over several program lines, but when executed, it internally becomes a single line of text to MANTIS SQL Support. Enter the following block:

```
10 EXEC_SQL
20 | SELECT * FROM table-name
30 | WHERE col-name>:MIN VALUE
40 END
```

You can execute it using the RUN command by entering RUN 10. MANTIS stores the block as a single line of text and associates it with the last program line in the SQL block, in this example, line 40. Therefore, if MANTIS encounters an error in the program block, it returns the error message and displays line 40.

If you bind the program, however, MANTIS stores the block as a single line and associates it with the line number of the last SQL statement, in this case line 30. If you want to run the bound SQL block using the RUN command, enter RUN 30. If MANTIS encounters an error in the SQL block, it returns the error message and displays line 30.

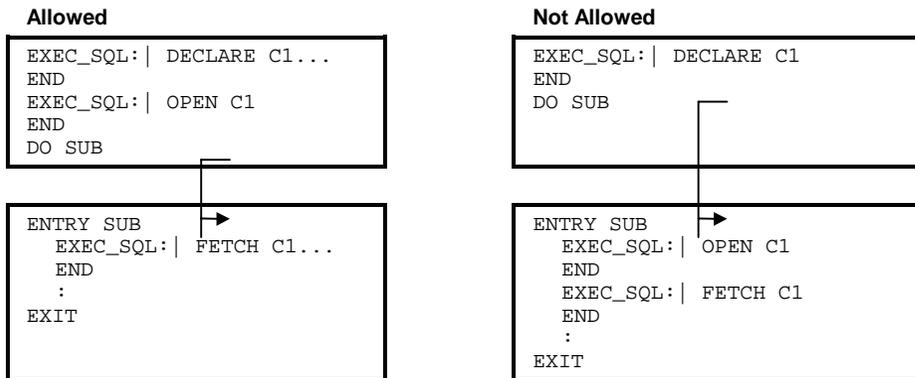
Cursors, statements, and SQLDAs

SQL statement names are MANTIS entities and not SQL entities. Hence, their scope is limited strictly to the program or external subprogram where they are prepared. SQL cursors are SQL entities and not MANTIS entities. Cursors are global to SUPRA but local to a MANTIS program context, which has the following implications:

- ◆ You cannot pass SQL statements and cursors as parameters or use them in non-SQL MANTIS statements because SQL statements and cursors are SQL entities and not MANTIS entities.
- ◆ You should not declare or use the same cursor in two different programs because it may not work.

A cursor is synonymous with the name of a result table and can be used in FETCH statements even though they have not been explicitly declared or opened. Because MANTIS interprets cursor names differently in different statements, a cursor name can be declared and opened in one program and used in a FETCH statement in an external subprogram. However, you cannot open a cursor that is declared in a calling program because MANTIS uses the cursor name to link the OPEN statement to the DECLARE statement and cursor names are local in scope from MANTIS's point of view.

The following examples illustrate this point:



Connection to SUPRA and multiple session support

MANTIS supports explicit and implicit database connection, as described in *MANTIS for Windows SQL Support for SUPRA Administration Guide*, P19-2308.

Multiple session support allows you to connect up to eight SUPRA databases concurrently. You can specify the name of the SUPRA database in your MANTIS program in one of two ways:

- ◆ `SQLCA("DBNAME")="database-name"` (See “SQLCA in MANTIS SQL Support” on page 40.)
- ◆ `EXEC_SQL: | SET DBNAME dbname-spec` (See “MANTIS EXEC_SQL statement” on page 31 and “SQL SET DBNAME statement” on page 39.)

The specified DBNAME will be used on the next implicit or explicit database connection.

Having specified the database name, you can then specify the SUPRA session number which is associated with a database connection. The session number can only be specified on an EXEC_SQL statement.

Disconnection from SUPRA

MANTIS disconnects from SUPRA by executing an SQL COMMIT WORK RELEASE statement. This can be performed explicitly by a MANTIS program or implicitly by the MANTIS nucleus.

Possible causes of disconnection are as follows:

- ◆ EXEC_SQL:!
COMMIT [WORK] RELEASE
- ◆ MANTIS main program context clean-up
- ◆ A MANTIS CHAIN statement, if the MANTIS option for database sign-off on a CHAIN statement is enabled

MANTIS main program context clean-up occurs in the following circumstances:

- ◆ The current test program context is released in Program Design as the result of a NEW, LOAD, EDIT, or RUN command, or when the Program Design Facility is exited.
- ◆ When a main program terminates in RUN mode; that is, when not under the control of Program Design.

MANTIS EXEC_SQL statement

The common usage of this statement has already been shown in the examples in this manual. This section discusses using EXEC_SQL for multiple session support.

```
EXEC_SQL[(exp1[,exp2])] [: ! SQL statement]
```

```
    ! SQL statement continued . . .
```

```
END
```

exp1

- | | |
|-----------------------|---|
| Description | <i>Optional.</i> Specifies the database system type (DBTYPE), or the SUPRA SQL CONNECT session number when multiple SUPRA connections are required. |
| Default | The current DBTYPE remains in effect until one of the following occurs: <ul style="list-style-type: none"> ◆ It is explicitly changed by an EXEC_SQL or SQLCA statement. ◆ It is implicitly changed by signing on to another MANTIS user. |
| Format | Text expression equal to SUPRA or a numeric expression evaluating to a session number in the range of 1–8. |
| Considerations | <ul style="list-style-type: none"> ◆ If it is a numeric session number, the current DBTYPE must be SUPRA. ◆ The default DBTYPE is specified in the MANTIS User Profile by using the Master User Facilities. |

exp2

Description *Optional.* Specifies the CONNECT session number when the first expression is used to specify SUPRA as the DBTYPE.

Format Numeric expression equal to a session number in the range of 1–8.

General consideration

MANTIS support for SUPRA SQL offers multiple session support within SUPRA.

Example

```
..EXEC_SQL("SUPRA"):| SELECT EMPLNO, EMPLNA  
...| INTO :EMPL,:NAME  
...| FROM EMPLOYEES WHERE DEPT=17  
..END
```

SQL WHENEVER statement

The WHENEVER statement in MANTIS SQL Support differs from that of SQL in C in the following ways:

- ◆ It is interpretive, not compiled.
- ◆ GOTO is replaced by DO, and STOP is replaced by FAULT.
- ◆ The default for SQLERROR is FAULT, not CONTINUE.

The syntax of the MANTIS SQL Support WHENEVER statement is shown below. Note that any action (DO, FAULT, or CONTINUE) can be selected for any condition (SQLERROR, SQLWARNING, NOT FOUND, or SQLEXCEPTION).

WHENEVER *condition action*

condition

Description *Required.* Indicates the condition you want to check for.

Options Valid conditions are SQLERROR, SQLWARNING, NOT FOUND, and SQLEXCEPTION. Each is explained in more detail below:

SQLERROR

Description *Optional.* Indicates that SUPRA returned an error code as the result of an SQL statement; SQLCODE<0.

Default action FAULT

SQL WARNING

Description *Optional.* Indicates that SQLCA(SQLWARN0)=W and that SQLCODE=0.

Default action CONTINUE

NOT FOUND

Description *Optional.* Indicates that SUPRA cannot find a row to satisfy your SQL statement, or there are no more rows to fetch (SQLCODE=100).

Default action CONTINUE

SQL EXCEPTION

Description *Optional.* Indicates SQL timeout error conditions; SQLCODE>100. Timeouts can be configured for locks and access to SUPRA.

Default action CONTINUE

action

- Description** *Required.* Specifies the action to be taken when the named condition is met.
- Options** Valid actions are DO *entry-name*[(parms)], FAULT, and CONTINUE.

DO *entry-name*[(parms)]

- Description** *Optional.* Indicates a standard MANTIS DO (internal or external) that corresponds to the WHENEVER-GOTO SQL statement in SQL in C. WHENEVER-DO transfers control to the specified internal routine or external program whenever the named condition is encountered.

Considerations

WHENEVER-DO can transfer control to an internal routine or external program, which in turn can contain any MANTIS logic, including CHAIN, EXIT, or STOP statements. The current values of any DO arguments at the time of the EXEC_SQL that caused the DO to occur are passed to the named subroutine. The subroutine EXIT returns control to the next statement following the EXEC_SQL that caused the DO to occur.

The WHENEVER-DO action resembles the existing functionality of the SET TRAP statement in MANTIS for Windows. If the DO portion of a WHENEVER-DO contains an error, MANTIS returns a MANTIS error message associated with the DO statement, not an SQL WHENEVER-type error. MANTIS displays the line in error in the subroutine. The WHENEVER statement may be outside of the current execution path. Remember that DO is executed as a result of an SQL statement raising the condition with which the DO action is associated.

FAULT

Description *Optional.* Terminates execution of the program and displays the error message returned by SUPRA in the form of a MANTIS fault message.



Only if the WHENEVER condition FAULT is in effect will MANTIS SQL Support intercept the specified condition and fault the MANTIS program. Remember that FAULT is the default action for SQLERROR.

CONTINUE

Description *Optional.* Permits program execution to continue without interruption when the named condition occurs. Your program should then check the SQLCA SQLCODE for the results of each EXEC_SQL.

The following table provides a quick reference for the WHENEVER conditions and default actions:

Condition	Default action
SQLERROR	FAULT
SQLWARNING	CONTINUE
NOT FOUND	CONTINUE
SQLEXCEPTION	CONTINUE

Example

```
00200 |  
00210 | SET 'WHENEVER' SETTINGS TO DESIRED VALUES  
00220 |  
00230 EXEC_SQL: | WHENEVER SQLEERROR DO DO  
          ROUTINE(PARM1, PARM2, PARM3)  
00240 END  
00250 EXEC_SQL: | WHENEVER SQLWARNING FAULT  
00260 END  
00270 EXEC_SQL: | WHENEVER NOT FOUND CONTINUE  
00280 END  
00290 EXEC_SQL: | WHENEVER SQLEXCEPTION CONTINUE  
00300 END
```

Declarative versus interpretive WHENEVER statements

In SQL in C, WHENEVER is a declarative statement that is processed when the program is precompiled, not when it is executed. Thus, in a C program the current WHENEVER setting is determined by its location in the source program.

In contrast, in MANTIS SQL Support the last-executed WHENEVER statement is in effect regardless of its location in the program. This difference is important when a WHENEVER statement is used with conditional statements. The following illustrates the different effects of a declared versus interpreted WHENEVER statement. In the following figure, the letter C denotes a condition, and 1 and 2 denote actions. The same considerations apply to FOR, UNTIL, WHEN, and IF structures in MANTIS.

SQL in C pseudocode	Setting in effect	MANTIS SQL Support pseudocode	Setting in effect
20 WHENEVER C1 .	C1	20 WHENEVER C1 .	C1
40 WHILE condition	C1	40 WHILE condition	C1 first, then C2*
50 WHENEVER C2 .	C2	50 WHENEVER C2 .	C1 or C2*
70 ENDWHILE	C2	70 ENDWHILE	C1 or C2*
80 EXEC_SQL	C2	80 EXEC_SQL	C1 or C2*

Since the setting is established before run time, it remains unchanged regardless of whether lines 50–70 are executed.

The first time statement 40 is executed, the setting is C1; thereafter it is C2.

WHENEVER statement

The scope of the WHENEVER statement is the current MANTIS DOLEVEL and every EXEC_SQL until a new WHENEVER is executed. If the default WHENEVER settings are not desired, WHENEVER must be issued in each externally done program.

SQL COMMIT/ROLLBACK statements

The SQL COMMIT/ROLLBACK statements perform the following functions:

- ◆ **COMMIT.** Terminates the Logical Unit of Work (LUW) and applies any SUPRA updates or changes made during the LUW.
- ◆ **ROLLBACK.** Terminates the LUW and backs out any SUPRA updates or changes.

COMMIT [WORK] [RELEASE]

ROLLBACK

The WORK parameter is supported for compatibility with MANTIS SQL Support for the IBM mainframe, but if you specify it, it will be ignored. WORK applies to both COMMIT and ROLLBACK.

The RELEASE parameter is a request to disconnect from SUPRA upon successful completion of the COMMIT or ROLLBACK (see “[Disconnection from SUPRA](#)” on page 30).

SQL SET DBNAME statement

The SQL SET DBNAME statement provides an alternate way to change the setting of SQLCA(DBNAME). It allows you to specify the SUPRA name to use in all subsequent connects. See “[SQLCA elements](#)” on page 42 for more information.

SET DBNAME 'database-name'
:parameter

SQLCA in MANTIS SQL Support

In SQL in C, the SQLCA (SQL Communications Area) is a data structure. An application written in SQL in C accesses elements in the SQLCA as items of data. In MANTIS SQL Support, the SQLCA function and statement perform the complementary operations of reading and writing elements of the SQLCA structure. All standard SQLCA capabilities are provided.

SQLCA syntax

In the following example, the syntax of the built-in SQLCA function (read) is shown first and the SQLCA statement (write) is shown second:

mantis-variable=SQLCA(element-name)

SQLCA(element-name)=expression



Because the SQLCA is a built-in function, it is not declared. An **INCLUDE SQLCA** statement is not required or recommended.

mantis-variable

Description *Required.* Specifies a value for your SQLCA element, or a variable to receive the value of your SQLCA element (depending on if a statement or function is used).

Format Can be one of the following:

- ◆ MANTIS variable name (possibly subscripted). For an SQLCA function.
- ◆ A valid MANTIS expression. For an SQLCA statement.
- ◆ A literal of the appropriate type. For an SQLCA statement.

Consideration The data type for the MANTIS variable depends on the specified element-name (see the table under “[SQLCA elements](#)” on page 42).

element-name

- Description** *Required.* Is or contains the name of the element to be returned/read (SQLCA function), or set/written (SQLCA statement).
- Format** Valid SQLCA element, as listed in the table under “SQLCA elements” on page 42.
- Consideration** Quotation marks (“ ”) are required when *element-name* is specified as a text literal. For example:

```
IF SQLCA("SQLCODE")<ZERO
.DO ERROR CONDITION ROUTINE
END
```

A text variable containing the *element-name* is also valid. For example:

```
CACODE="SQLCODE"
IF SQLCA(CACODE)<ZERO
.DO ERROR CONDITION ROUTINE
END
```

expression

- Description** *Required.* Specifies any MANTIS expression that is the compatible data type of the SQLCA element that you are storing.

General considerations

- ◆ To read a particular SQLCA field, use the format:
sqlca-element-value=SQLCA(*element-name*)
- ◆ To update a particular SQLCA field, use the format:
SQLCA(*element-name*)=*sqlca-element-value*

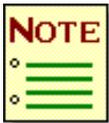
SQLCA elements

The following table lists SQLCA elements, the compatible MANTIS variable type, and usage notes:

Element	MANTIS variable [†]	Usage notes
DBTYPE	TEXT(6)	Unique to MANTIS SQL Support
DBNAME	TEXT(64)	Unique to MANTIS SQL Support
SQLCAID	TEXT(8)	Read only
SQLCABC	NUMERIC	Read only
SQLCODE	NUMERIC	
SQLERRML	NUMERIC	Read only
SQLERRMC	TEXT(70)	Read only
SQLERRP	TEXT(8)	Read only
SQLERRD _{<i>n</i>}	NUMERIC	<i>n</i> ranges from 1–6
SQLWARN _{<i>n</i>}	TEXT(1)	<i>n</i> ranges from 0–F
SQLEXT	TEXT(84)	Read only
MSGTEXT	TEXT(256)	Read only; unique to MANTIS SQL Support

[†] If a data value is moved from an SQLCA element to a MANTIS variable of shorter length (e.g., an 8-character SQLCA element to a 6-character MANTIS variable), the right-most characters are truncated.

SQLCAID, SQLCABC, SQLERRML, SQLERRMC, SQLERRP, SQLEXT, and MSGTEXT are read-only. Although values can be written into the other elements, doing so does not pass any information to SUPRA. In addition, because these elements are used by SUPRA, their contents may be changed when each EXEC_SQL statement is executed. See the following discussion of MSGTEXT for more information.



For portable, nonvolatile MANTIS software, consider the entire SQLCA structure as read-only.

Additional elements added to MANTIS SQL Support are:

DBTYPE

- Description** *Optional.* Specifies the database with which MANTIS SQL Support will communicate.
- Format** 1- to 6-character text value for the current DBTYPE, as explained in “MANTIS EXEC_SQL statement” on page 31.
- Options** SUPRA
- Consideration** All EXEC_SQL statements executed will access the database name specified by DBTYPE.

DBNAME

- Description** *Optional.* Specifies the database name used in all subsequent connections to SUPRA.
- Format** 1- to 64-character text value.
- Consideration** If you do not specify a database name, MANTIS obtains it from the DBNAME environment variable.

MSGTEXT

Description *Optional.* Returns the error message text for the current SQLCODE so it can be manipulated within a MANTIS program, if needed. For example:

```
EXEC_SQL
...
END
IF SQLCA("SQLCODE") < 0
.SHOW SQLCA("MSGTEXT");
END
```

Considerations

- ◆ SQLCA(SQLCODE) is reset to 0 at an implicit COMMIT. An implicit COMMIT is issued at any terminal read operation, unless the program or user explicitly turns off the automatic COMMIT logic by using the COMMIT OFF statement.
- ◆ If, for some reason, your system administrator does not install the text of SUPRA error messages, the following message will be displayed for all SUPRA errors:

```
750 SQLEERROR:nnnn:
```

In this message, the *nnnn* is the number for the SQL error.

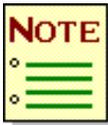
COMMIT and ROLLBACK in SUPRA and COMMIT and RESET in MANTIS SQL Support

In MANTIS SQL Support, SQL COMMIT WORK and ROLLBACK WORK statements have exactly the same effect on SUPRA as the standard MANTIS COMMIT and RESET statements. However, using an SQL COMMIT or ROLLBACK does not imply a MANTIS COMMIT or RESET, but using a MANTIS COMMIT or RESET does imply an SQL COMMIT or ROLLBACK. Executing an SQL COMMIT statement commits only SQL updates, and only for the specified SQL session. MANTIS automatically performs a COMMIT at terminal input for all SUPRA sessions. For more information, refer to the appropriate SUPRA Server SQL Programmer's Guide. MANTIS COMMIT is performed in the following circumstances:

- ◆ When a MANTIS COMMIT statement is encountered.
- ◆ When any terminal input function (CONVERSE, OBTAIN, WAIT, or MORE prompt) is executed by any MANTIS program (including Program Design when reading command lines), unless COMMIT OFF has been specified.
- ◆ During MANTIS main program termination, the COMMIT is performed prior to SUPRA disconnection.

MANTIS RESET is performed in the following circumstances:

- ◆ When a MANTIS RESET statement is encountered.
- ◆ When a MANTIS fault occurs, except for breakpoint faults.



If you are in Program Design and have set a breakpoint, a RESET does not occur when the breakpoint is encountered (the breakpoint is a fault condition). However, unless you have specified a COMMIT OFF, an automatic COMMIT occurs when Program Design prompts you for the next input line. Be aware that this can occur, because it could result in the program behaving differently under breakpoints.

Give careful consideration to using COMMIT and RESET in your MANTIS SQL applications, as well.

Error messages

You can receive messages from three sources: the MANTIS nucleus, MANTIS SQL Support, and SUPRA. MANTIS facility and programming messages are documented in *MANTIS for Windows Language Reference Manual*, P19-2302, and *MANTIS for Windows Facilities Reference Manual*, P19-2301. MANTIS SQL Support messages appear in “MANTIS SQL Support error messages” on page 83 of this manual.

When MANTIS encounters an SQL error from SUPRA, it displays a fault message. The statement where the error occurred, and the text of that line. Messages from SUPRA are prefaced with the 3-character MANTIS fault code 750. A message from the SUPRA contains the SQLCODE value and its associated error message text. The format is as follows:

```
750 SQLERROR:nnnn: ###...
```

In the previous example, *nnnn* is the 3- or 4-digit SQLCODE value and *###...* is the message returned by SUPRA. The following is displayed by MANTIS when SUPRA returns -802 in the SQLCA SQLCODE, and the WHENEVER SQLERROR condition was set to FAULT:

```
750 SQLERROR:-802: INVALID NUMERIC INPUT PARAMETER VALUE
```

4

Dynamic SQL in MANTIS SQL Support

This chapter discusses how dynamic SQL works in MANTIS SQL Support. Dynamic SQL in MANTIS SQL Support differs somewhat from dynamic SQL in other languages. If you are new to dynamic SQL programming, you may want to review this information.

Dynamic SQL is a method for executing SQL statements when data such as SQL statements, tables, or column names are needed, but is not known by the program before program execution begins. For example, if an application requires a user to interactively enter an SQL statement at the terminal during program execution, the application must use dynamic SQL. The I/SQL utility is an example of such an application.

Virtually any statement in a static application can also be executed dynamically. The main statements that enable you to execute SQL statements dynamically are PREPARE, DESCRIBE, EXECUTE, and EXECUTE IMMEDIATE. Alternate forms of DECLARE, OPEN, and FETCH statements are used in dynamic SQL. Communication to and from the database is done using these statements and an SQLDA data structure.

The SQLDA data structure consists of header elements and repeating elements (each repeating element group is called SQLVAR). Each SQLVAR contains metadata (such as data length and data type) about the data going between your program and SUPRA. Consider the SQLDA as a representation and repository of the data being transferred.

Programs using dynamic SQL must procedurally define data about the SQL statements and host variables. In static SQL programs, the SQL preprocessor determines this information. A single program can contain either static SQL statements, dynamic SQL statements, or both.

Execute an SQL statement dynamically

In general, to execute a statement dynamically, you must prepare an SQL statement with a PREPARE statement and then execute it with an EXECUTE statement. If data is being retrieved, inserted, or updated, the program must manipulate an SQLDA between preparation and execution. This manipulation can include allocating and expanding an SQLDA, retrieving metadata from SUPRA with the DESCRIBE statement, and causing data transfer between SUPRA and MANTIS host variables.

The following code provides a sample SELECT and FETCH sequence for dynamic SQL. The syntax for the PREPARE, DESCRIBE, and EXECUTE statements used here appears in the *SUPRA Server SQL Commands Reference Manual*, P26-2420. Sample code for the SQLDA built-in statement and function (described in “[SQLDA structure](#)” on page 50) is also included in this example:

SELECT statement with input parameters

```

50 TEXT DA:DA="sqlda-name"
60 SQLDA(DA)=NEW
70 TEXT SELECT STMT(250)
80 SELECT STMT="SELECT * FROM EMPLOYEE WHERE EMPNO=?"
90 EXEC_SQL:| PREPARE stmt-name INTO sqlda-name FROM :SELECT
  STMT
100 END
110 EXEC_SQL:| DECLARE cursor-name CURSOR FOR stmt-name
120 END
130 SQLDA(DA,"SQLHOSTVAR",I)=input-parameter:| I = 1 TO
  SQLDA(DA,"SQLN")
140 EXEC_SQL:| OPEN cursor-name USING DESCRIPTOR sqlda-name
150 END

```

FETCH statement with output parameters

```

70 EXEC_SQL:| FETCH cursor-name USING DESCRIPTOR sqlda-name
80 END
90 SHOW SQLDA(DA,"SQLHOSTVAR",I):| I = 1 TO SQLDA(DA,"SQLD")

```

Other statements with input parameters

```

20 TEXT UPDATE STMT(250)
30 UPDATE STMT="UPDATE EMPLOYEE SET FIRSTNME=:FIRST NAME WHERE
EMPNO=?"
40 EXEC_SQL:| PREPARE stmt-name FROM :UPDATE STMT
50 END
60 EXEC_SQL:| DESCRIBE stmt-name INTO sqlda-name
70 END
80 SQLDA(DA,"SQLHOSTVAR",I)=input-parameter:| I = 1 TO
SQLDA(DA,"SQLN")
90 EXEC_SQL:| EXECUTE stmt-name USING DESCRIPTOR sqlda-name
100 END

```

Complete code for dynamic insert, update, delete, and select routines are in [“Sample MANTIS SQL programs”](#) on page 65, along with their static equivalents.

SQLDA structure

In dynamic SQL support, data is transferred between your program and SUPRA by using and SQLDA (SQL Descriptor Area). An SQLDA is a data structure that holds information about data (metadata) that is transferred between your program and the database.

The following figure shows the structure of an SQLDA. The first four elements are header elements; they occur once per SQLDA. The next nine elements repeat once per data item. A data item is either one column of an SQL table (output from SUPRA to your program) or the value of a host variable (input to SUPRA from your program). The maximum number of entries is 300. The following table summarizes header and repeating elements:

SQLDAID	SQLMAX	SQLN	SQLD	Header Elements
SQLCOLNAME ₁				Repeating Element ₁
SQLCOLIO ₁		SQLCOLMODE ₁		
SQLCOLTYPE ₁		SQLCOLLENGTH ₁		
SQLCOLFRAC ₁		SQLHOSTFIND ₁		
SQLHOSTVARTY ₁		SQLHOSTVAR ₁		
SQLCOLNAME ₂				Repeating Element ₂
SQLCOLIO ₂		SQLCOLMODE ₂		
SQLCOLTYPE ₂		SQLCOLLENGTH ₂		
SQLCOLFRAC ₂		SQLHOSTFIND ₂		
SQLHOSTVARTY ₂		SQLHOSTVAR ₂		

In SQL in C, you must explicitly declare each SQLDA element as a data area in your program and then access SQLDA elements through programming statements. In MANTIS SQL Support, when you declare an SQLDA, an SQLDA with all the elements above is built for you. The SQLDA contains the default number of repeating elements set by the Master User in the configuration file. This value can be modified by your program with the SQLMAX header element (see “[Move data from your program into an SQLDA header element](#)” on page 54).

The MANTIS SQLDA statement allows your MANTIS program to create and maintain SQLDA data structures which are in turn used with dynamic SQL statements. Your MANTIS program can use the SQLDA statement to create a named SQLDA structure and to place host variable information into the SQLDA. The *sqlda-name* parameter must be a text expression, literal, or the name of a valid MANTIS variable. Your program can also use the SQLDA function to retrieve information from the SQLDA. The SQLDA statement and function are described and illustrated in the sections that follow. Their uses are outlined in the following list:

- ◆ To allocate or deallocate an SQLDA:

```
SQLDA (sqlda-name) =NEW
SQLDA (sqlda-name) =QUIT
```

- ◆ To move data from your program into an SQLDA:

```
SQLDA (sqlda-name, header-element) =expression
SQLDA (sqlda-name, repeating-element, index) =expression
```

- ◆ To move data from an SQLDA into your program:

```
mantis-variable =SQLDA (sqlda-name, header-element)
mantis-variable =SQLDA (sqlda-name, repeating-element, index)
```

In this syntax, *sqlda-name*, *header-element*, *repeating-element*, and *index* refer to standard MANTIS variables, literals, or expressions. MANTIS syntax must be followed with the exception of *sqlda-name* which requires SQL naming conventions. In SQL, a name must begin with an alphabetic character and cannot exceed 18 characters. The only special characters allowed are #, \$, and @. Using special characters in an embedded SQL statement causes MANTIS to generate faults. Header elements and repeating elements are listed in the tables in “[SQLDA header elements](#)” on page 56 and “[SQLDA repeating elements](#)” on page 62. In the examples above, *index* refers to the sequential occurrence of the repeating element group in the SQLDA.

Allocate an SQLDA

To allocate a new SQLDA, use the following SQLDA statement:

SQLDA(*sqlda-name*)=NEW

sqlda-name

Description *Required.* Specifies the name of the SQLDA.

Format Valid text expression evaluating to a valid SQLDA name of 1–18 characters.

Example

```
SQLDA ( "SQLDA1 " ) =NEW
```

General considerations

- ◆ This statement allocates a new, empty SQLDA structure with the default number of repeating elements. The default is a MANTIS option that can be modified by the Master User (refer to *MANTIS for Windows SQL Support for SUPRA Administration Guide*, P19-2308). Within your program you can also modify an SQLDA's size by resetting the value of SQLMAX (see the discussion of SQLMAX in the table under “[SQLDA header elements](#)” on page 56).
- ◆ If you declare an SQLDA of the same name as one that already exists, the second SQLDA statement is ignored.
- ◆ The scope of an SQLDA is the current DO level. For example, you can have two SQLDAs of the same name on different DO levels. Within a DO level, however, you can only access SQLDAs defined for that DO level.

Deallocate an SQLDA

To deallocate an SQLDA, use the following SQLDA statement:

SQLDA(*sqlda-name*)=QUIT

sqlda-name

Description *Required.* Specifies the name of the SQLDA to be deallocated.

Format Valid text expression evaluating to a valid SQLDA name of 1–18 characters.

Example

```
SQLDA ( "SQLDA1 " ) =QUIT
```

General Considerations

- ◆ This statement deallocates an existing SQLDA by name.
- ◆ An SQLDA defined at a DO level is also deallocated when that DO level is exited.
- ◆ SQLDAs are also deallocated in the case of a RUN without a line number.
- ◆ A RUN with a line number may produce unpredictable results if you have modified the program.

Move data from your program into an SQLDA header element

Use the following SQLDA statement to set header or column-name information in the SQLDA:

SQLDA(*sqlda-name*,*header-element*)=*expression*

sqlda-name

- Description** *Required.* Specifies the name of a previously allocated SQLDA.
- Format** Valid text expression evaluating to a valid SQLDA name of 1–18 characters.

header-element

- Description** *Required.* Indicates the name of an SQLDA header element into which you are moving data.

- Options** **SQLN** SQLN is the number of repeating groups (SQLVARs) currently in use. The value can range from 1 through SQLMAX. This value is normally set as a result of a DESCRIBE statement. In some cases, SQLN must be set by your program (e.g., when you are inserting data).

```
SQLDA("DA1", "SQLN")=10
```

SQLMAX SQLMAX is the total number of repeating groups available in the SQLDA. The value can range from 1–300. Setting this number in your program causes the SQLDA to expand or contract by the specified number of repetitions. Once physically expanded, the space occupied by the SQLDA never physically contracts. For example, if an SQLDA named DA1 has 20 repeating occurrences, the following statement reduces the logical occurrences to five; however, physical space for 20 remains (these numbers are arbitrary):

```
SQLDA("DA1", "SQLMAX")=5
```

SQLD SQLD is the total number of output host variables in the SQLDA. The value of SQLD must be less than or equal to SQLN.

```
SQLDA("DA1", "SQLD")=8
```

expression

Description *Required.* Specifies the SQLDA variable count.

Format Standard MANTIS variable, literal, or expression.

Consideration Since all SQLDA header elements that can be set are numeric, the expression must also always be numeric.

Example

```
SQLDA("SQLDA1", "SQLN")=TOTAL NEEDED
```

General considerations

- ◆ In SQL in C, when a DESCRIBE statement is executed, if the SQLDA is too small (SQLMAX is less than the number of items that will be returned as a result of the DESCRIBE), SUPRA sets SQLN to the required number and terminates. The program must then expand the SQLDA accordingly. By contrast, MANTIS SQL Support automatically expands the SQLDA to the required size if the SQLDA is too small to accept the results of a DESCRIBE. You can check the number of occurrences after the DESCRIBE by examining the SQLN value.
- ◆ The other header element illustrated in the previous figure, SQLDAID, is read-only. If you attempt to use a read-only element in this SQLDA statement, you will cause a MANTIS fault.

SQLDA header elements

The following table shows how and when SQL header elements are used, what the results are, and whether they can be updated:

Element	How set/when used	Results	Updatable?
SQLDAID			
Eyecatcher	Set by SQL	Normally contains SQLDA	No
SQLN			
Total number of host variables in SQLDA	Set, as a result of a DESCRIBE, to the total number of host variable parameters in the statement (except for DESCRIBE in FETCH USING DESCRIPTOR where SQLN is set to the number of result table columns)	Number of host variables required	Yes
SQLMAX			
Number of repeating groups in SQLDA	Set using value from Configuration file when SQLDA is allocated; can be modified in program if needed	Number of repeating groups allocated	Yes
SQLD			
Total number of output host variables in SQLDA	Set as a result of a DESCRIBE to the number of output host variables (except for DESCRIBE in FETCH USING DESCRIPTOR where SQLD is set to the number of result table columns)	Number of output MANTIS variables described in SQLDA	Yes

Move data from your program into an SQLDA repeating element

To supply values for input host variables and to set the value of repeating elements, use the following SQLDA statement:

SQLDA(*sqlda-name*,*repeating-element*,*index*)=*expression*

sqlda-name

Description *Required.* Specifies the name of a previously allocated SQLDA.

repeating-element

Description *Required.* Specifies the name of the repeating element into which you are moving data.

Options SQLCOLNAME Provides the column name returned by SUPRA. It can also be set by your program. SQLCOLNAME has a type of TEXT and a length of 18. Note that although you can modify the SQLCOLNAME element, it does not have an effect on the database. In addition, the database writes to this element, so its contents may be destroyed at each EXEC_SQL statement execution.

SQLHOSTIND See “Indicator variables” on page 23 which describes the interpretation of an indicator variable and of the host variable data when a column is NULL (does not exist). Also note the following list of values and meanings for indicator variables:

Value	Meaning
=0	The host variable contains data.
<0	The host variable is NULL (does not exist).
>0	The host variable contains truncated data. The indicator variable value is the original length of the host variable data.

SQLHOSTVAR In SQL in C, this element holds a memory address. This address is used programmatically to access the data item being transferred between the program and the database. In SQL in C, the program must acquire space for the data item and place the space's address in this element. In MANTIS SQL Support, SQLHOSTVAR is used to automatically perform these actions for you. When you are transferring data into the SQLDA, this element automatically:

- ◆ Allocates a data area for the data item or expands the data area, if necessary.
- ◆ Sets the value of SQLHOSTVAR to the address of the data area. This address is used internally by MANTIS SQL Support; your program does not need to manipulate this value.
- ◆ Moves data from the MANTIS host variable into the SQLDA data area.
- ◆ Sets SQLCOLTYPE and SQLCOLLENGTH to match the definition of the MANTIS variable according to the SQLCOLTYPE values in the following table. SQLCOLLENGTH is set to the length of the MANTIS variable.

If you are transferring data out of the SQLDA, this element simply performs the transfer. Note as well that the SQLHOSTVAR element may have a type of numeric or string. When you do not know the type of data you want to retrieve from the database in advance, use the SQLCOLTYPE and SQLHOSTVARTY elements to determine the data type.

index

Description *Required.* Specifies the relative occurrence of the repeating element into which you are moving data. The value should be relative to 1.

expression

Description *Required.* Specifies an SQLVAR element value.

Format Standard MANTIS variable, literal, or expression.

Consideration The expression may be either text or numeric. There are no limitations.

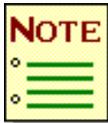
Example

```
SQLDA("SQLDA1", "SQLHOSTVAR", 9) = SALARY
```

General considerations

- ◆ SQLCOLLENGTH and SQLHOSTVARTY will be set to the length and data type of the MANTIS expression.
- ◆ SQLHOSTVARTY is always set to the MANTIS equivalent data type. A data type conversion table is shown in the following example:

SUPRA DRDM		MANTIS SQLDA		MANTIS
Data type	Data code	Data type	Data code	Data type
(SQLCOLTYPE)		(SQLHOSTVARTY)		
Fixed	0	Float (8 bytes long)	3	BIG/SMALL
Float	1	Float (8 bytes long)	3	BIG/SMALL
Character	2	Character string (filled with blanks)	6	TEXT
Date	4	Character string	6	TEXT
Time	5	Character string	6	TEXT
String	6	Character string (with nonprintable characters)	8	TEXT



Certain representations of numeric data in the SQL database may cause conversion errors in the data values. You can avoid this by using the appropriate MANTIS numeric data type (BIG or SMALL) to store the data. If conversion errors appear when you are using a SMALL data type to store a value, try BIG instead, and vice versa.

Move data from an SQLDA header element into your program

Use the following SQLDA function to read header elements:

mantis-variable=SQLDA(sqlda-name,header-element)

mantis-variable

Description *Required.* Specifies the variable into which the SQLDA header element is to be placed.

Format Standard MANTIS variable name.

sqlda-name

Description *Required.* Specifies the name of a previously allocated SQLDA.

header-element

Description *Required.* Provides the name of the header element you are reading.

Example

```
TOTAL NEEDED=SQLDA( "SQLDA1" , "SQLN" )
```

General considerations

- ◆ No index value is permitted.
- ◆ You may read all header elements.

Move data from an SQLDA repeating element into your program

Use the following SQLDA function to transfer data from a repeating element into a MANTIS variable:

mantis-variable=SQLDA(sqlda-name,repeating-element,index)

mantis-variable

Description *Required.* Specifies the variable into which the SQLDA repeating element is to be placed.

Format Standard MANTIS variable name.

sqlda-name

Description *Required.* Specifies the name of a previously allocated SQLDA.

repeating-element

Description *Required.* Specifies the name of the repeating element to be read.

index

Description *Required.* Provides the relative occurrence of the repeating element to be read. This value should be relative to 1.

Example

```
EMPLOYEE NUMBER=SQLDA( "SQLDA1 " , "SQLHOSTVAR" , 1 )
```

General considerations

- ◆ You may read all repeating elements.
- ◆ Data types between repeating elements and MANTIS variables must match.

SQLDA repeating elements

The following table shows how and when SQL repeating elements are used, what the results are, and whether they can be updated:

Element	How set/when used	Results	Updatable?
SQLCOLNAME			
SQL column name	Set by DRDM as the result of a DESCRIBE; can be set by program	Column or header name	Yes
SQLCOLIO			
Indicates whether parameter is input or output	Set by DRDM as the result of a DESCRIBE	0 = Input 1 = Output	No
SQLCOLMODE			
Indicates whether NULL values are allowed	Set as the result of a DESCRIBE	0 = Not allowed <> 0 = Allowed	No
SQLCOLTYPE			
Data type as it resides on the database. Code differs depending on whether it is set by SQLCOLTYPE or SQLHOSTVARTY [†]	Set by DRDM as the result of a DESCRIBE.	SQLCOLTYPE BIG/SMALL 0 Fixed 1 Float TEXT 2 Character TEXT 4 Date 5 Time 6 String	No
SQLCOLLENGTH			
Total number of bytes used to store the data	Set by DRDM as the result of a DESCRIBE or by MANTIS when data is transferred by SQLHOSTVAR.	BIG/SMALL: 8 TEXT: maximum column length when set by DESCRIBE, MANTIS variable current length when set by SQLDA	No

[†] The SQLHOSTVAR element may have a type of numeric or string. When you do not know the type of data you want to retrieve from the database in advance, use the SQLCOLTYPE and SQLHOSTVARTY elements to determine the data type.

Element	How set/when used	Results	Updatable?
SQLCOLFRAC			
Number of decimal positions for FIXED column types	Set by DRDM as the result of a DESCRIBE and not used by MANTIS since all numeric data is floating point	1 for FLOAT column types	No
SQLHOSTIND			
Contains the value of the indicator variable	Used to indicate the presence of NULL variables and truncated data. Set by DRDM during SQL function can be set by your program	= 0 Defined value; no errors < 0 NULL value > 0 Original column length due to truncated value	Yes
SQLHOSTVARTY			
Contains the data type of the data in the SQLDA ^{††}	Set by MANTIS when SQLHOSTVAR is used	SQLHOSTVARTY [†] BIG/SMALL 3 Float TEXT 6 Character string filled with blanks	No
SQLHOSTVAR			
Subfunction that physically transfers data between MANTIS data areas and the SQLDA data areas	Used to transfer value of variable between database and MANTIS	Data stored in MANTIS variable or SQLDA; SQLCOLTYPE and SQLCOLLENGTH set according to value being transferred when data is moved into SQLDA	Yes

[†] These are not the only SQLHOSTVARTY codes, but a list of codes for MANTIS-compatible data types.

^{††} The SQLHOSTVAR element may have a type of numeric or string. When you do not know the type of data you want to retrieve from the database in advance, use the SQLCOLTYPE and SQLHOSTVARTY elements to determine the data type.

A

Sample MANTIS SQL programs

MANTIS HELP includes prompters for the dynamic SQL statements interpreted by MANTIS. These prompters are subject to the syntax rules for SUPRA SQL.

Use the following HELP commands:

- ◆ HELP EXEC_SQL
- ◆ HELP SQL (displays a list of available SUPRA help prompters)
- ◆ HELP SUPRA-DECLARE

Each of the SUPRA-specific HELP prompters includes an example program that is reproduced in the EXAMPLES program library.

For clarity, the following examples do not contain error checking or display logic. Employee information is hardcoded into the programs. Each static example has the same functionality as the dynamic example that follows.

Static insert routine

```
10 ENTRY STATIC INSERT
20 .BIG HIRE DATE , BIRTH DATE , JOB CODE , SALARY , EDUCATION LEVEL
30 .TEXT EMPLOYEE NUMBER(6) , FIRST NAME(20) , MIDDLE INITIAL(1) , LAST NAME(20)
40 .TEXT PHONE NUMBER(4) , WORK DEPARTMENT(3) , SEX(1)
50 .|
60 .EMPLOYEE NUMBER="000120"
70 .FIRST NAME="SEAN"
80 .MIDDLE INITIAL=" "
90 .LAST NAME="O'CONNELL"
100 .BIRTH DATE=421018
110 .HIRE DATE=631205
120 .JOB CODE=58
130 .EDUCATION LEVEL=14
140 .SALARY=29250
150 .PHONE NUMBER="2167"
160 .WORK DEPARTMENT="A00"
170 .SEX="M"
180 .|
190 .EXEC_SQL:| INSERT INTO EMPLOYEE.TABLE
200 ..|                (EMPNO ,
210 ..|                FIRSTNME ,
220 ..|                MIDINIT ,
230 ..|                LASTNAME ,
240 ..|                BRTHDATE ,
250 ..|                HIREDATE ,
260 ..|                JOBCODE ,
270 ..|                EDUCLVL ,
280 ..|                SALARY ,
290 ..|                PHONENO ,
300 ..|                WORKDEPT ,
310 ..|                SEX)
320 ..|                VALUES (:EMPLOYEE NUMBER ,
330 ..|                :FIRST NAME ,
340 ..|                :MIDDLE INITIAL ,
```

```
350 ..|           :LAST NAME ,
360 ..|           :BIRTH DATE ,
370 ..|           :HIRE DATE ,
380 ..|           :JOB CODE ,
390 ..|           :EDUCATION LEVEL ,
400 ..|           :SALARY ,
410 ..|           :PHONE NUMBER ,
420 ..|           :WORK DEPARTMENT ,
430 ..|           :SEX )
440 .END
450 EXIT
```

Dynamic insert routine

```

10 ENTRY DYNAMIC INSERT
20 .BIG HIRE DATE,BIRTH DATE,JOB CODE,SALARY,EDUCATION LEVEL
30 .TEXT EMPLOYEE NUMBER(6),FIRST NAME(20),MIDDLE INITIAL(1),LAST NAME(20)
40 .TEXT PHONE NUMBER(4),WORK DEPARTMENT(3),SEX(1)
50 .TEXT SQL TEXT(254)
60 .|
70 .EMPLOYEE NUMBER="000120"
80 .FIRST NAME="SEAN"
90 .MIDDLE INITIAL=" "
100 .LAST NAME="O'CONNELL"
110 .BIRTH DATE=421018
120 .HIRE DATE=631205
130 .JOB CODE=58
140 .EDUCATION LEVEL=14
150 .SALARY=29250
160 .PHONE NUMBER="2167"
170 .WORK DEPARTMENT="A00"
180 .SEX="M"
190 .|
200 .SQL TEXT="INSERT INTO EMPLOYEE.TABLE"
210 . "(EMPNO,FIRSTNME,MIDINIT,LASTNAME,BRTHDATE",
220 . "HIREDATE,JOBCODE,EDUCLVL,SALARY,PHONENO",
230 . "WORKDEPT,SEX)"
240 . "VALUES (?,?,?,?,?,?,?,?,?,?)"
250 .|
260 .EXEC_SQL:| PREPARE S1 FROM :SQL TEXT
270 .END
280 .|
290 .SQLDA("SQLDA1")=NEW
300 .SQLDA("SQLDA1","SQLMAX")=12
310 .EXEC_SQL:| DESCRIBE S1 INTO SQLDA1
320 .END
330 .SQLDA("SQLDA1","SQLHOSTVAR",1)=EMPLOYEE NUMBER
340 .SQLDA("SQLDA1","SQLHOSTVAR",2)=FIRST NAME

```

```
350 .SQLDA("SQLDA1", "SQLHOSTVAR", 3)=MIDDLE INITIAL
360 .SQLDA("SQLDA1", "SQLHOSTVAR", 4)=LAST NAME
370 .SQLDA("SQLDA1", "SQLHOSTVAR", 5)=BIRTH DATE
380 .SQLDA("SQLDA1", "SQLHOSTVAR", 6)=HIRE DATE
390 .SQLDA("SQLDA1", "SQLHOSTVAR", 7)=JOB CODE
400 .SQLDA("SQLDA1", "SQLHOSTVAR", 8)=EDUCATION LEVEL
410 .SQLDA("SQLDA1", "SQLHOSTVAR", 9)=SALARY
420 .SQLDA("SQLDA1", "SQLHOSTVAR", 10)=PHONE NUMBER
430 .SQLDA("SQLDA1", "SQLHOSTVAR", 11)=WORK DEPARTMENT
440 .SQLDA("SQLDA1", "SQLHOSTVAR", 12)=SEX
450 .|
460 .EXEC_SQL:| EXECUTE S1 USING DESCRIPTOR SQLDA1
470 .END
480 EXIT
```

Static update routine

```
10 ENTRY STATIC UPDATE
20 .BIG HIRE DATE,BIRTH DATE
30 .TEXT EMPLOYEE NUMBER(6)
40 .TEXT FIRST NAME(20),MIDDLE INITIAL(1),LAST NAME(20)
50 .|
60 .EMPLOYEE NUMBER="000120"
70 .FIRST NAME="JOHN"
80 .MIDDLE INITIAL="H"
90 .LAST NAME="DOE"
100 .BIRTH DATE=490113
110 .HIRE DATE=880120
120 .|
130 .EXEC_SQL
140 ..|
150 ..| UPDATE EMPLOYEE.TABLE
160 ..|
170 ..| SET FIRSTNME=:FIRST NAME,
180 ..|     MIDINIT=:MIDDLE INITIAL,
190 ..|     LASTNAME=:LAST NAME,
200 ..|     BRTHDATE=:BIRTH DATE,
210 ..|     HIREDATE=:HIRE DATE
220 ..|
230 ..| WHERE EMPNO=:EMPLOYEE NUMBER
240 .END
250 EXIT
```

Dynamic update routine

```

10 ENTRY DYNAMIC UPDATE
20 .BIG HIRE DATE,BIRTH DATE
30 .TEXT EMPLOYEE NUMBER(6),FIRST NAME(20),MIDDLE INITIAL(1),LAST NAME(20)
40 .TEXT DA(18),DAPARM(8)
50 .TEXT SQL TEXT(254)
60 .|
70 .EMPLOYEE NUMBER="000120"
80 .FIRST NAME="JOHN"
90 .MIDDLE INITIAL="H"
100 .LAST NAME="DOE"
110 .BIRTH DATE=490113
120 .HIRE DATE=880120
130 .|
140 .SQL TEXT="UPDATE EMPLOYEE.TABLE SET "
150 .SQL TEXT=SQL TEXT+"FIRSTNME=?,MIDINIT=?,LASTNAME=?, "
160 .SQL TEXT=SQL TEXT+"BRTHDATE=?,HIREDATE=? "
170 .SQL TEXT=SQL TEXT+"WHERE EMPNO=?"
180 .|
190 .EXEC_SQL:| PREPARE S1 FROM :SQL TEXT
200 .END
210 .|
220 .SQLDA("SQLDA1")=NEW
230 .DA="SQLDA1"
240 .DAPARM="SQLHOSTVAR"
250 .SQLDA(DA,"SQLMAX")=6
260 .EXEC_SQL:| DESCRIBE S1 INTO SQLDA1
270 .END
280 .SQLDA(DA,DAPARM,1)=FIRST NAME
290 .SQLDA(DA,DAPARM,2)=MIDDLE INITIAL
300 .SQLDA(DA,DAPARM,3)=LAST NAME
310 .SQLDA(DA,DAPARM,4)=BIRTH DATE
320 .SQLDA(DA,DAPARM,5)=HIRE DATE
330 .SQLDA(DA,DAPARM,6)=EMPLOYEE NUMBER
340 .|
350 .EXEC_SQL:| EXECUTE S1 USING DESCRIPTOR SQLDA1
360 .END
370 EXIT

```

Static select routine

```
10 ENTRY STATIC SELECT
20 .BIG HIRE DATE,BIRTH DATE,JOB CODE,SALARY,EDUCATION LEVEL
30 .TEXT EMPLOYEE NUMBER(6),FIRST NAME(20),MIDDLE INITIAL(1),LAST NAME(20)
40 .TEXT WORK DEPARTMENT(3),PHONE NUMBER(3),SEX(1)
50 .EMPLOYEE NUMBER="000120"
60 .|
70 .EXEC_SQL:| DECLARE C1 CURSOR FOR
80 ..|           SELECT * FROM EMPLOYEE.TABLE
90 ..|           WHERE EMPNO = :EMPLOYEE NUMBER
100 .END
110 .EXEC_SQL:| OPEN C1
120 .END
130 .EXEC_SQL:| FETCH C1 INTO :EMPLOYEE NUMBER,
140 ..|           :FIRST NAME,
150 ..|           :MIDDLE INITIAL,
160 ..|           :LAST NAME,
170 ..|           :WORK DEPARTMENT,
180 ..|           :PHONE NUMBER,
190 ..|           :HIRE DATE,
200 ..|           :JOB CODE,
210 ..|           :EDUCATION LEVEL,
220 ..|           :SEX,
230 ..|           :BIRTH DATE,
240 ..|           :SALARY
250 .END
260 .EXEC_SQL:| CLOSE C1
270 .END
280 EXIT
```

Dynamic select routine

```

10 ENTRY DYNAMIC SELECT
20 .BIG HIRE DATE , BIRTH DATE , JOB CODE , SALARY , EDUCATION LEVEL
30 .TEXT EMPLOYEE NUMBER(6)
40 .TEXT FIRST NAME(20) , MIDDLE INITIAL(1) , LAST NAME(20)
50 .TEXT WORK DEPARTMENT(3) , PHONE NUMBER(3) , SEX(1)
60 .TEXT SQL TEXT(254)
70 .|
80 .EMPLOYEE NUMBER="000120"
90 .SQL TEXT="SELECT * FROM EMPLOYEE.TABLE "
100 .' "WHERE EMPNO = ?"
110 .|
120 .EXEC_SQL:| PREPARE S1 FROM :SQL TEXT
130 .END
140 .EXEC_SQL:| DECLARE C1 CURSOR FOR S1
150 .END
160 .EXEC_SQL:| OPEN C1 USING :EMPLOYEE NUMBER
170 .END
180 .SQL TEXT="FETCH C1 USING DESCRIPTOR"
190 .EXEC_SQL:| PREPARE S2 FROM :SQL TEXT
200 .END
210 .SQLDA("SQLDA1")=NEW
220 .EXEC_SQL:| DESCRIBE S2 INTO SQLDA1
230 .END
240 .EXEC_SQL:| EXECUTE S2 USING DESCRIPTOR SQLDA1
250 .END
260 .EXEC_SQL:| CLOSE C1

270 .END
280 .|
290 .EMPLOYEE NUMBER=SQLDA("SQLDA1" , "SQLHOSTVAR" , 1)
300 .FIRST NAME=SQLDA("SQLDA1" , "SQLHOSTVAR" , 2)
310 .MIDDLE INITIAL=SQLDA("SQLDA1" , "SQLHOSTVAR" , 3)
320 .LAST NAME=SQLDA("SQLDA1" , "SQLHOSTVAR" , 4)
330 .WORK DEPARTMENT=SQLDA("SQLDA1" , "SQLHOSTVAR" , 5)

```

```
340 .PHONE NUMBER=SQLDA("SQLDA1", "SQLHOSTVAR", 6)
350 .HIRE DATE=SQLDA("SQLDA1", "SQLHOSTVAR", 7)
360 .JOB CODE=SQLDA("SQLDA1", "SQLHOSTVAR", 8)
370 .EDUCATION LEVEL=SQLDA("SQLDA1", "SQLHOSTVAR", 9)
380 .SEX=SQLDA("SQLDA1", "SQLHOSTVAR", 10)
390 .BIRTH DATE=SQLDA("SQLDA1", "SQLHOSTVAR", 11)
400 .SALARY=SQLDA("SQLDA1", "SQLHOSTVAR", 12)
410 EXIT
```

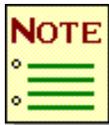
Static delete routine

```

10 ENTRY STATIC DELETE
20 .TEXT EMPLOYEE NUMBER(6)
30 .EMPLOYEE NUMBER="000120"
40 .EXEC_SQL
50 ..|
60 ..| DELETE FROM EMPLOYEE.TABLE
70 ..|
80 ..| WHERE EMPNO = :EMPLOYEE NUMBER
90 .END
100 EXIT

```

Dynamic delete routine



Using an SQLDA is not required because no data is transferred between the database system and the MANTIS program.

```

10 ENTRY DYNAMIC DELETE
20 .TEXT EMPLOYEE NUMBER(6),SQL TEXT(254)
30 .EMPLOYEE NUMBER="000120"
40 .SQL TEXT="DELETE FROM EMPLOYEE.TABLE WHERE EMPNO=?"
50 ..|
60 .EXEC_SQL
70 ..|
80 ..| PREPARE S1 FROM :SQL TEXT
90 ..|
100 .END
110 .EXEC_SQL
120 ..|
130 ..| EXECUTE S1 USING :EMPLOYEE NUMBER
140 ..|
150 .END

```

SQL query function

```

10 ENTRY SQL QUERY
20 .|
30 .| This example illustrates the use of dynamic SQL to perform a
40 .| QUERY-like function.
50 .|
60 .| E.g.:
70 .|         INSERT INTO table VALUES (?,?,...)
80 .|         SELECT * FROM table
90 .|         SELECT * FROM table WHERE column > ?
100 .|        SELECT * FROM table INTO ?,?,... WHERE column = ?
110 .|        COMMIT WORK RELEASE is NOT recommended
120 .|
130 .TEXT STMT(255),TEXT PARM(80),DA CMD,DA FETCH
140 .DA CMD="DA CMD"
150 .DA FETCH="DA FETCH"
160 .SQLDA(DA CMD)=NEW
170 .SQLDA(DA FETCH)=NEW
180 .EXEC_SQL:|WHENEVER SQLERROR DO QHANDLER
190 .END
200 .|
210 .WHILE NOT(FINISHED)
220 ..|
230 ..| Prompt for next SQL statement and execute it
240 ..|
250 ..STMT=""
260 ..SHOW "SQL>";:OBTAIN STMT
270 ..IF KEY<>"ENTER" OR STMT=""
280 ...FINISHED=TRUE
290 ..ELSE
300 ...UNPAD STMT BEFORE
310 ...STMT=UPPERCASE(STMT)
320 ..IF POINT(STMT-"?")=0:|
330 ...SQLDA(DA CMD,"SQLN")=0
340 ...SQLDA(DA CMD,"SQLD")=0
350 ...EXEC_SQL:|EXECUTE IMMEDIATE :STMT
360 ...END
370 ...ELSE:|

```

Any host variables?

Yes, use SQLDA

```
380 ....EXEC_SQL:|PREPARE S1 INTO DA CMD FROM :STMT
390 ....END
400 ....IF SQLDA(DA CMD,"SQLN")>SQLDA(DA CMD,"SQLD")
410 .....DO SQLDA INPUT(DA CMD)
420 ....END
430 ....EXEC_SQL:|EXECUTE S1 USING DESCRIPTOR DA CMD
440 ....END
450 ...END
460 ...IF SQLCA("SQLCODE")=0
470 ....IF STMT(1,6)="SELECT" AND POINT(STMT-"INTO")=0
480 .....|
490 .....| A SELECT without an INTO clause:
500 .....| FETCH all rows from result table
510 .....|
520 .....WHILE SQLCA("SQLCODE")=0
530 .....EXEC_SQL:|FETCH USING DESCRIPTOR DA FETCH
540 .....END
550 .....IF SQLCA("SQLCODE")=0
560 .....DO SQLDA OUTPUT(DA FETCH)
570 .....END
580 ....END
590 ...ELSE:| No result table to process
600 ....IF SQLDA(DA CMD,"SQLD")>0 AND SQLCA("SQLCODE")=0
610 .....DO SQLDA OUTPUT(DA CMD)
620 ....END
630 ...END
640 ..END
650 .END
660 .END
670 EXIT
680 |
690 | Allow continuation after failure to execute SQL statement
700 |
710 ENTRY QHANDLER
720 .SHOW "*** SQL ERROR CODE =";SQLCA("SQLCODE")
730 .SHOW "***";SQLCA("SQLERRMC")
```

```
740 .SHOW "*** Press ENTER to continue, CANCEL to stop";:WAIT
750 .IF KEY<>"ENTER"
760 .. STOP
770 .END
780 EXIT
790 |
800 | Process Input host-variable parameters (?)
810 |
820 ENTRY SQLDA INPUT(DA)
830 .I=0
840 .WHILE I<SQLDA(DA,"SQLN")
850 ..I=I+1
860 ..IF SQLDA(DA,"SQLCOLIO",I)=0:|           An input parameter
870 ...SHOW "Enter Input parameter";I;";";
880 ...WHEN SQLDA(DA,"SQLCOLTYPE",I)<2
890 ....OBTAIN BIG PARM
900 ....SQLDA(DA,"SQLHOSTVAR",I)=BIG PARM
910 ...WHEN SQLDA(DA,"SQLCOLTYPE",I)>2
920 ....TEXT BUFFER
930 ....OBTAIN TEXT PARM
940 ....SQLDA(DA,"SQLHOSTVAR",I)=TEXT PARM
950 ...END
960 ..END
970 .END
980 EXIT
990 |
1000 | Process Output host-variable parameters
1010 |
1020 ENTRY SQLDA OUTPUT(DA)
1030 .I=0
1040 .WHILE I<SQLDA(DA,"SQLN")
1050 ..I=I+1
1060 ..IF SQLDA(DA,"SQLCOLIO",I)=1:|           An output parameter
1070 ...SHOW SQLDA(DA,"SQLHOSTVAR",I),
1080 ..END
1090 .END
1100 .SHOW
1110 EXIT
```

Dynamic column select

```

10 ENTRY SQL LIST COLUMNS
20 .|
30 .| THIS PROGRAM LISTS COLUMNS BASED ON TABLE NAME
40 .|
50 .TEXT TABLE NAME(32)
60 .TEXT SQL FUNCTION(100) Contains the SQL statement
70 .SHOW "PLEASE ENTER TABLE NAME:"
80 .OBTAIN TABLE NAME
90 .SQL FUNCTION="SELECT * FROM "+TABLE NAME Creates the SQL statement
100 .EXEC_SQL
110 ..| EXECUTE IMMEDIATE :SQL FUNCTION Creates the result set
120 .END
130 .SQL FUNCTION="FETCH USING DESCRIPTOR"
140 .EXEC_SQL:| PREPARE S1 FROM :SQL FUNCTION Prepares the SQL statement
150 .END
160 .SQLDA("SQLDA1")=NEW
170 .EXEC_SQL:| DESCRIBE S1 INTO SQLDA1 Returns table column data into
180 .END the SQLDA
190 .EXEC_SQL:| EXECUTE S1 USING DESCRIPTOR SQLDA1 Retrieves first row of data from
200 .END the table
210 .COUNTER=1
220 .SHOW"COLUMN NAME",AT(25),"TYPE",AT(45),"LENGTH",AT(55),"DATA"
230 .WHILE COUNTER<SQLDA("SQLDA1","SQLD")
240 ..SHOW SQLDA("SQLDA1","SQLCOLNAME",COUNTER) Displays returned data
250 ..'AT(25),SQLDA("SQLDA1","SQLCOLTYPE",COUNTER)
260 ..'AT(45),SQLDA("SQLDA1","SQLCOLLENGTH",COUNTER)
270 ..'AT(55),SQLDA("SQLDA1","SQLHOSTVAR",COUNTER)
280 ..COUNTER=COUNTER+1
290 .END
300 .WAIT
310 EXIT

```


B

Features not supported

The following features of SQL are not supported in MANTIS for Windows:

- ◆ Host variables may not be specified in a SELECT list. For example:

```
SELECT A, :VX, C
INTO :VA, :VB, :VC    VX is invalidly used as a host variable.
```

- ◆ Exact line number reference upon the detection of a syntax error is not supported in all cases. Once control is transferred to the database system in the execution of an SQL statement, MANTIS no longer has control and therefore cannot keep track of where the error was encountered. For example:

```
1330 ..X=X+1
1340 ..EXEC_SQL
1350 ...|SELECT A,B,C
1360 ...|INTO :VA,:VB),:VC    ← Error in this line
1370 ...|FROM TABLE.1
1380 ...|WHERE A=1
1390 ..END
1400 ..X=X-VA
```

For unbound programs, MANTIS points to the last line in the program block. For bound programs, MANTIS points to the line before the END statement. For example:

```
1330 ..X=X+1
1340 ..EXEC_SQL
1350 ...|SELECT A,B,C
1360 ...|INTO :VA,:VB),:VC ← Error in this line
1370 ...|FROM TABLE.1
1380 ...|WHERE A=1          ← FAULTS display this line when
                             bound
1390 ..END                 ← FAULTS display this line when
                             unbound
1400 ..X=X-VA
```

- ◆ The contents of one SQLDA structure cannot be implicitly copied into another in a single instruction. The following statement is not permitted:

```
SQLDA("NAME2")=SQLDA("NAME1")
```

However, each element of an SQLDA can be passed individually to the corresponding element of a different SQLDA.

C

MANTIS SQL Support error messages

In a MANTIS SQL Support application, you may receive messages from three sources: MANTIS SQL Support, the MANTIS nucleus, and the database system. This appendix lists the MANTIS SQL Support messages and provides a suggested corrective action.

Messages from the MANTIS nucleus are documented in *MANTIS for Mainframe Error Messages*, P19-5004.

Database system messages are documented in the SUPRA documentation. Messages from the database system start with *-nnnn*.

720

The `SQLDA` name is missing or incorrectly specified.

Explanation The statement or function expects an `SQLDA` name as a parameter. Either the `SQLDA` name is missing or the specified `SQLDA` name is invalid.

Action An `SQLDA` name must evaluate to a nonzero-length text string. Add or correct the `SQLDA` name parameter.

721 The SQLDA element name is missing or incorrectly specified.

Explanation The SQLDA element name is the second SQLDA parameter and must be specified as a TEXT expression that evaluates to one of the following:

SQLDAID	SQLCOLNAME/ SQLNAME	SQLHOSTIND/ SQLIND
SQLMAX	SQLCOLIO	SQLHOSTVARTY/ SQLTYPE
SQLN	SQLCOLMODE	SQLHOSTVAR/ SQLDATA
SQLD	SQLCOLTYPE/ SQLTYPE	SQLCOLFRAC
	SQLCOLLENGTH/ SQLLEN	

The element names separated by a slash (/) are synonymous.

Action Add or correct the SQLDA element name.

722 The SQLDA variable index is missing or incorrectly specified.

Explanation The third SQLDA parameter specifies the index (from 1) into the repeating element (SQLVAR) array of the named SQLDA structure and must be a numeric expression.

Action Add or correct the SQLDA variable index parameter.

723

The SQLCA element name is missing or incorrectly specified.

Explanation The first SQLCA parameter addresses a given field of the SQLCA (global SQL Communications Area). It must be a TEXT expression that evaluates to one of the following:

DBTYPE	SQLERRD1	SQLWARN0
DBNAME	SQLERRD2	SQLWARN1
SQLCAID	SQLERRD3	:
SQLCABC	SQLERRD4	SQLWARN7
SQLCODE	SQLERRD5	SQLWARN8
SQLERRML	SQLERRD6	: SUPRA ONLY
SQLERRMC		SQLWARNF
SQLTEXT		

Action Add or correct the SQLCA element name.

725

An embedded host variable indicator is not numeric.

Explanation You have specified an indicator variable that is not numeric. A numeric variable is required to store the following:

- ◆ = 0 Error free, non-NULL host variable data transfer
- ◆ < 0 Associated host variable is NULL
- ◆ > 0 Original column length when output host variable truncation has occurred (SQLWARNING).

Action Specify a numeric indicator variable.

726

The SQLDA variable index is out of range.

Explanation The third SQLDA parameter is an index into the SQLVAR repeating elements of the SQLDA. The value you have specified is not in the allowed range: 1–SQLMAX.

Action Modify the SQLDA variable index value.

727 The specified host variable is not yet allocated.

Explanation There is no host variable data for the specified SQLVAR entry.

Action The specified SQLVAR entry has not been set by the SQLDA statement or has not been set as a result of an EXECUTE statement.

728 The SQLDA or SQLCA element is read-only.

Explanation You have attempted to assign a value to a read-only field of the SQLCA or SQLDA structure. Certain fields are protected against programmed update because MANTIS SQL Support relies on their unaltered contents for internal consistency. Read-only fields are as follows:

SQLCA		SQLDA
SQLCABC	SQLDAID	SQLCOLTYPE/SQLTYPE
SQLCAID	SQLCOLIO	SQLCOLLENGTH/SQLEN
SQLERRML	SQLABC	SQLHOSTVARTY/SQLTYPE
SQLERRMC	SQLCOLMODE	SQLCOLFRAC
SQLERRP		
SQLEXT		
MSGTEXT		

The element names separated by a slash (/) are synonymous.

Action Omit any assignments for read-only fields.

729

The value of the SQLDA or SQLCA element is the wrong data type.

Explanation The specified data to be assigned to the SQLDA or the SQLCA element does not match the data type of the element. You have tried to assign a text value to a numeric element, or vice versa. The settable element data types are as follows:

SQLCA		SQLDA	
DBTYPE	TEXT(8)	SQLMAX	NUMERIC
DBNAME	TEXT(64)	SQLN	NUMERIC
SQLCODE	NUMERIC	SQLD	NUMERIC
		SQLCOLNAME/ SQLNAME	TEXT(18/30)
SQLWARN _n	TEXT(1)	SQLHOSTVAR/ SQLDATA	NUMERIC or TEXT
SQLERRD _n	NUMERIC	SQLHOSTIND/ SQLIND	NUMERIC

Action Correct the data type you have specified.

730

The value of the SQLDA element is out of range.

Explanation You have attempted to set SQLN, SQLD, or SQLMAX to an illegal value. For IBM SQL support, SQLMAX and SQLN are synonyms for the same field (SQLN) of the SQLDA. These values are restricted as follows:

- ◆ SQLMAX -> 1-300
- ◆ SQLN -> 1-SQLMAX
- ◆ SQLD -> 1-SQLN

Action Correct the value you have specified.

731

The specified SQL DBTYPE is not supported.

Explanation The specified DBTYPE is not supported. The DBTYPE element is used to specify the SQL system when multiple SQL systems are active.

Action Specify SUPRA.

- 732** Substrings for output host variables are not supported in compatibility mode.
- Explanation** You have specified substring subscripts on an output host variable. This is not compatible with MANTIS for the IBM mainframe.
- Action** Use a temporary TEXT variable as the output host variable and perform a MANTIS substring assignment as required.
- 733** The SQL session number is invalid.
- Explanation** The SQL session number must be in the range from one through eight. It is used to specify the session number for multiple concurrent CONNECT's to SUPRA SQL. The default is 1.
- Action** Enter an SQL session number in the range of one through eight.
- 735** The DBNAME parameter is missing or incorrectly specified.
- Explanation** The SQL SET DBNAME statement parameter must be either a quoted string literal or an embedded host variable.
- Action** Correct the SET DBNAME statement.
- 737** A numeric session parameter is not valid when the DBTYPE is not "SUPRA".
- Explanation** You have specified a numeric session parameter when the current or selected SQL DBTYPE is not SUPRA. Only SUPRA SQL supports multiple sessions that can be selected by session number on the EXEC_SQL statement. The current DBTYPE is established at user sign-on time and remains in effect until explicitly changed by an SQLCA or EXEC_SQL statement. The most probable cause of this error occurs when the program is expecting SUPRA as the current DBTYPE, but it has been set to a different value.
- Action** Specify SUPRA as the selected SQL DBTYPE before entering a numeric session parameter.

- 738** The cursor name is missing or incorrectly specified.
- Explanation** The cursor name in the WHERE CURRENT OF clause of an SQL DELETE or UPDATE statement is not specified correctly.
- Action** Correct the DELETE or UPDATE statement. MANTIS expects the CURRENT OF clause to be specified as follows: CURRENT OF *cursor-name*.
- 748** There is no result table for the FETCH with auto-cursor.
- Explanation** The FETCH statement has not specified a cursor-name. This is an auto-cursor, (implicit cursor) request.
- Action** You must execute a SELECT statement in the program before executing a FETCH with auto-cursor.
- 749** An embedded host variable is missing.
- Explanation** The host variable list is incorrectly specified. There is a missing colon after a comma in the list.
- Action** Check the list carefully. Note that every host variable in the list must be preceded by a colon and that indicator variables, if specified, are not preceded by a comma.
- 750** `SQLERROR:nnnn: ###...`
- Explanation** An error has occurred in the SQL system. The SQLCODE shown immediately after the MANTIS fault code is followed by the error message text generated by the SQL system. By default, MANTIS only generates this fault in the case of SQLERROR conditions (when SQLCODE is less than zero). The MANTIS program will continue after SQLWARNING, SQLEXCEPTION, and NOT FOUND conditions. You can use the SQL WHENEVER statement to handle (or fault on) any or all of the four possible SQL error conditions.
- Action** Please consult the appropriate SQL reference manual for probable cause of the error and corrective action.

- 751** No SQL database name has been specified for connection.
- Explanation** An SQL database name is required for connection to the SQL system. If the first SQL statement executed is not a CONNECT statement, MANTIS performs an implicit CONNECT using the values of the following system environment variables: DBUSER, DBPASS, and DBNAME. If DBNAME is not defined, MANTIS uses the DBNAME established by the SUPRA/XUSER command. If connection using DBUSER fails, MANTIS attempts an implicit CONNECT using the names set up by the SUPRA XUSER command.
- Action** Ensure that the DBNAME environment variable identifies a valid SUPRA database, or that you have executed SUPRA XUSER and obtained correct parameters for an implicit CONNECT.
- 752** The SQL statement text is missing or incorrectly coded.
- Explanation** The SQL statement text for the EXEC_SQL statement is missing. An EXEC_SQL statement must be followed by at least one nonblank MANTIS comment line. The most likely cause of this error is when the SQL statement text is coded on the same line as the EXEC_SQL verb and either the colon or comment character is missing (:!).
- Action** Correct the EXEC_SQL statement line in your program.
- 753** The cursor has not been OPENED.
- Explanation** Before a FETCH operation can be executed on a cursor (named result table), the cursor must first be opened by the OPEN statement.
- Action** Use the OPEN statement to open the cursor.
- 754** The cursor name has not been DECLARED.
- Explanation** The cursor named in the OPEN statement has not yet been declared in a DECLARE statement.
- Action** Use the DECLARE statement to declare the cursor.

- 755** The statement name has not been PREPARED.
- Explanation** You must use a PREPARE statement to declare a statement name before you can use that statement in a dynamic DECLARE or EXECUTE statement.
- Action** Check that the execution path of the program includes a PREPARE statement which declares the statement name.
- 756** The named SQLDA has not been allocated.
- Explanation** A named SQLDA must be allocated via the SQLDA statement before it can be used in any other statement.
- Action** Use the SQLDA statement to allocate the named SQLDA. For example:
- ```
SQLDA("sqlda1")=NEW
EXEC_SQL:| PREPARE s1 INTO sqlda1 FROM:STMT
END
```
- 757** The BEGIN/END statement is incorrectly specified.
- Explanation** The BEGIN DECLARE SECTION and END DECLARE SECTION statements, as well as the INCLUDE statement, do not perform any function in MANTIS SQL Support other than documentation.
- Action** Code BEGIN/END DECLARE SECTION as follows:
- ```
EXEC_SQL: | BEGIN DECLARE SECTION
END
EXEC_SQL: | END DECLARE SECTION
END
```

- 758** The named SQLDA was created for a different DBTYPE.
- Explanation** The SQLDA named in the SQL statement was created for another DBTYPE and cannot be used for the current DBTYPE. An SQLDA inherits the DBTYPE that is current when it is created by the SQLDA(*sqlda-name*)=NEW statement.
- Action** Your program may have altered the current DBTYPE before attempting to use the SQLDA. If your program does use multiple SQL systems, it must ensure that named SQLDAs created for a given SQL product are used for that system only. Use the SHOW SQLDA(*sqlda-name*, DBTYPE) statement in your program to prove that this is the case. Place this statement just before the SQLDA statement that creates the named SQLDA and just before the statement at fault.
- 759** An embedded host variable data type is incorrect.
- Explanation** The MANTIS data type of an embedded host variable conflicts with its usage. You have attempted to substitute a text parameter where a numeric value is required, or vice versa.
- Action** Correct the data type.
- 760** The FROM clause is missing or incorrectly specified.
- Explanation** The FROM clause is mandatory in the PREPARE statement. It is either missing or you have incorrectly placed it.
- Action** Add the clause or correct its placement.
- 761** The SQL statement specification is missing or incorrectly specified.
- Explanation** An SQL statement specification occurs with an embedded host variable or as an SQL string literal. It is mandatory in the PREPARE and EXECUTE IMMEDIATE statements.
- Action** Add the SQL statement specification or correct the specification.

- 762** The SQL statement name is missing or incorrectly specified.
- Explanation** An SQL statement name is mandatory in the PREPARE and DESCRIBE statements and expected in the DECLARE statement if the FOR clause does not specify a recognized SQL verb.
- Action** Add the SQL statement name or correct its specification.
- 763** Failure on SQL COMMIT, attempting to ROLLBACK instead.
- Explanation** A MANTIS COMMIT has failed on an SQL database. This could happen because of:
- ◆ Database sign-off (disconnect) as a result of:
 - NEW, LOAD, EDIT, or RUN commands
 - Main program termination in RUN mode, or exiting Program Design
 - CHAIN statement when the MANTIS option to sign-off is enabled
 - ◆ MANTIS COMMIT statement
 - ◆ Implicit COMMIT by MANTIS on a terminal input (unless COMMIT OFF has been executed by the running program)
- Because the only usual remedy for this type of error is to perform an SQL ROLLBACK function, MANTIS tries to ROLLBACK the current transaction for you.
- Action** Correct the cause of the COMMIT failure: usually some database updates that violate integrity rules.

- 764** The MANTIS SQL statement/cursor name limit has been exceeded.
- Explanation** The MANTIS interface to the Database Manager limits both the number of cursors and the number of dynamic SQL statements that can be used. Calculate the number of dynamic SQL statements allowed by summing the following (for the main program and all of its subprograms):
- ◆ The number of SQL statement names
 - ◆ The number of static DECLARE statements
- The current interface supports a total of eight concurrently used cursors and or/prepared statements.
- Action** Simplify your application if possible. Otherwise, contact your local Cincom representative for more information.
- 765** There is more than one result table row for the singleton SELECT (INTO).
- Explanation** The singleton SELECT statement was specified by the INTO clause of the SELECT statement. This is a request to obtain a result table with, at most, one row. The result of the executed SELECT statement has more than one row.
- Action** The WHERE clause of the SELECT statement should be tightened to restrict the number of result rows to one (or none). Otherwise, you must use a nonsingleton SELECT in conjunction with the DECLARE, OPEN, and/or FETCH statements.

766

The `WHENEVER` condition is missing or incorrectly specified.

Explanation The `WHENEVER` condition must be specified as the first parameter. Only the following keywords are valid for the `WHENEVER` condition:

Keyword	SUPRA SQL	IBM SQL
SQLERROR	(SQLCODE<0)	(SQLCODE<0)
SQLEXCEPTION	(SQLCODE>100)	(Not Applicable)
NOT FOUND	(SQLCODE=100)	(SQLCODE=100)
SQLWARNING	(SQLCODE>100 and SQLWARN0=W)	(SQLCODE>0 and SQLCODE<>100 OR SQLWARN0<> “ ”)

Action Add the `WHENEVER` condition or correct its placement.

767

The `WHENEVER` action is missing or incorrectly specified.

Explanation The `WHENEVER` action must be specified as the second parameter. Only the following keywords are valid for the `WHENEVER` action:

- ◆ **CONTINUE.** (Default for `SQLEXCEPTION`, `NOT FOUND`, and `SQLWARNING`)
- ◆ **FAULT.** (Default for `SQLERROR`)
- ◆ **DO.** (Never the default)

Action Add the `WHENEVER` action or correct its placement.

768

The `USING` parameter is missing or incorrectly specified.

Explanation The dynamic `USING` clause in the `OPEN`, `FETCH`, and `EXECUTE` statements can specify either a `DESCRIPTOR` or a host variable list. Neither of these is specified correctly.

Action Add or correct the `USING` parameter.

- 769** The `DECLARE CURSOR` statement is incorrectly specified.
- Explanation** The keyword `FOR` is missing from the `DECLARE CURSOR` statement.
- Action** Add the keyword `FOR` to your `DECLARE CURSOR` statement.
- 770** The `INTO` clause is missing.
- Explanation** The `INTO` clause is mandatory in the `DESCRIBE` statement. It must be specified immediately after the statement name.
- Action** Add the `INTO` clause to your `DESCRIBE` statement.
- 771** The `INTO` parameter is missing or incorrectly specified.
- Explanation** The `INTO` clause on the `FETCH` statement must specify a host variable list.
- Action** Correct the host variable list in the `INTO` clause.
- 772** The host variable is incorrectly specified.
- Explanation** The embedded host variable does not identify a MANTIS variable correctly. This error means that the word following the colon is not a valid MANTIS symbolic name.
- Action** Correct the variable specification.
- 773** The `FETCH` statement is incorrectly specified.
- Explanation** The `FETCH` statement requires either an `INTO` clause or a `USING` clause to specify the output data destination.
- Action** Add the `INTO` or `USING` clause to your `FETCH` statement.
- 775** One of `NEW` or `QUIT` keywords expected but not specified.
- Explanation** An `SQLDA` statement with one parameter can only be used to allocate and deallocate `SQLDA` structures. The only valid values for the right-hand side of the assignment are the keywords `NEW` and `QUIT`.
- Action** Add the appropriate keyword.

- 778** The END statement is missing after the SQL statement text.
- Explanation** The EXEC_SQL comment block is terminated by an END statement. Any other noncommented statement is not allowed.
- Action** Add an END statement to terminate your EXEC_SQL statement block.
- 779** The embedded SQL statement has a syntax error.
- Explanation** This is an internal problem. MANTIS SQL Support has encountered a syntax error in an embedded SQL statement.
- Action** Please contact your local Cincom representative.

D

Differences: MANTIS SQL Support versus SQL in C; MANTIS versus SQL

SQL in MANTIS SQL Support is essentially the same as SQL in C. In this manual, SQL in these non-MANTIS languages is called SQL in C for convenience.

This appendix summarizes the differences between SQL in MANTIS SQL Support and SQL in other languages. More information is provided in the sections specified.

SQL in MANTIS SQL Support versus SQL in C

- ◆ SQL statements are embedded in a MANTIS application program as standard MANTIS comments and delimit each SQL statement with an EXEC_SQL-END block. No MANTIS comments are permitted within the EXEC_SQL-END block. All comments within the block are considered SQL statement text.
- ◆ In the SQL WHENEVER statement:
 - The GOTO clause is replaced by a standard MANTIS DO statement, and STOP is replaced by FAULT.
 - The default for the condition SQLERROR is FAULT; in SQL in C, the default is CONTINUE.
 - WHENEVER settings may have different ranges of applicability in C than they would in SQL.

- ◆ SQLCA elements are accessed through the SQLCA statement/function rather than as items of data.
- ◆ Elements in SQLDAs are accessed through the SQLDA statement/function, rather than as items of data.
- ◆ In a MANTIS SQL Support application, you may receive messages from three sources: the MANTIS nucleus, MANTIS SQL Support, and the database system. MANTIS SQL Support messages are in “MANTIS SQL Support error messages” on page 83 of this manual.
- ◆ MANTIS SQL Support does not support an SQL INCLUDE statement, as INCLUDE denotes a preprocessor action. The SQLCA and SQLDA functions eliminate the need to INCLUDE these structures.
- ◆ DECLARE statements are unnecessary for tables and views.

MANTIS versus SQL

- ◆ In MANTIS, quotation marks (") delimit character-string constants. In SQL, apostrophes (') delimit character-string constants.
- ◆ Permissible data-type conversions between SQL and MANTIS are listed in the table in “Data conversion between MANTIS SQL Support and SUPRA” on page 24.
- ◆ Only data-type codes for MANTIS-compatible data types are returned in the SQLCOLTYPE element in the SQLDA. Valid data types are thus limited to those listed in the table in “Move data from your program into an SQLDA repeating element” on page 57.

Index

A

allocate an SQLDA 52
arrays, referencing values in 22

C

C, SQL in 15, 51, 99
COMMIT statement 30, 39, 45
CONTINUE 36
cursors 28

D

data conversion 24, 60
data types 22, 24
DBNAME 39, 43
DBTYPE 43
deallocate an SQLDA 53
declarative versus interpretive
 WHENEVER statements 38
DESCRIBE 47
differences between SQL in
 MANTIS and SQL in C 15,
 99
disconnecting from SUPRA 30
dynamic SQL
 description of 47
 embedding SQL in MANTIS 14,
 17
 sample programs 48

E

error messages
 how displayed 44
 list of 83
EXEC_SQL statement 31
EXEC_SQL-END 27

EXECUTE 47
EXECUTE IMMEDIATE 47

F

FAULT 36
features not supported 81

H

host variables 21

I

indicator variables 23

M

MANTIS data types vs. SQL data
 types 22
MANTIS SQL Support
 WHENEVER defaults 36
MANTIS vs. SQL 99
maximum number of entries in an
 SQLDA structure 50
messages
 how displayed 44
 list of 83
move data
 from program to SQLDA
 header element 54
 from program to SQLDA
 repeating element 57
 from SQLDA header element to
 program 60
 from SQLDA repeating element
 to program 61
MSGTEXT 44
multiple session support 29

N

NOT FOUND 34

P

- permissible data type
 - conversions 24
- PREPARE 47
- programming considerations 25

R

- release levels supported 13
- RESET statement 45
- ROLLBACK statement 39, 45

S

- sample MANTIS SQL programs
 - 65
- SQL EXCEPTION 34
- SQL in C 13, 51, 99
- SQL SET DBNAME 39

- SQL vs. MANTIS 99
- SQL WARNING 33
- SQL WHENEVER statement 33
- SQLCA 40
 - elements 42
- SQLDA 28, 50
 - header elements 55
 - repeating elements 61
- SQLERROR 33
- SQLHOSTIND 57
- SQLHOSTVAR 58
- SQLTYPE codes 50
- statements 28
- SUPRA
 - connection 29
 - multiple session support 29

W

- WHENEVER statement 33, 38

Reader Comment Sheet

Name: _____

Job title/function: _____

Company name: _____

Address: _____

Telephone number: _____ Date: _____

How often do you use this manual? Daily Weekly Monthly Less

How long have you been using this product? Months Years

Can you find the information you need? Yes No Please comment.

Is the information easy to understand? Yes No Please comment.

Is the information adequate to perform your task? Yes No Please comment.

General comment: _____

WE STRIVE FOR QUALITY

To respond, please fax to Larry Fasse at (513) 612-2000.