

Cincom

AD/ADVANTAGE

MANTIS DB2 Programming
UNIX

P39-1360-00



AD/Advantage
MANTIS DB2 Programming, UNIX
Publication Number P39-1360-00

© 2001 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This document contains unpublished, confidential, and proprietary information of Cincom.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®	iD CinDoc™	MANTIS®
C+A-RE™	iD CinDoc Web™	Socrates®
CINCOM®	iD Consulting™	Socrates® XML
Cincom Encompass®	iD Correspondence™	SPECTRA™
Cincom Smalltalk™	iD Correspondence Express™	SUPRA®
Cincom SupportWeb®	iD Environment™	SUPRA® Server
CINCOM SYSTEMS®	iD Solutions™	Visual Smalltalk®
	intelligent Document Solutions™	VisualWorks®
gOOi™	Intermax™	

All other trademarks are trademarks or registered trademarks of:

Acucobol, Inc.	Micro Focus, Inc.
AT&T	Microsoft Corporation
Compaq Computer Corporation	Systems Center, Inc.
Data General Corporation	TechGnosis International, Inc.
Gupta Technologies, Inc.	The Open Group
International Business Machines Corporation	UNIX System Laboratories, Inc.
JSB Computer Systems Ltd.	

or of their respective companies.

Cincom Systems, Inc.
55 Merchant Street
Cincinnati, Ohio 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 DB2 Programming, UNIX, P39-1360-00, is dated October 30, 2001. This document supports Release 5.5.01 of MANTIS.

We welcome your comments

We encourage critiques concerning the technical content and organization of this manual. At your convenience, please take the [survey](#) provided with the online documentation.

Cincom Technical Support for AD/Advantage

<i>All customers</i>	Web:	http://supportweb.cincom.com
<i>U. S. A. customers</i>	Phone:	1-800-727-3525
	FAX:	(513) 612-2000
	Attn:	AD/Advantage Support
	Mail:	Cincom Systems, Inc. Attn: AD/Advantage Support 55 Merchant Street Cincinnati, OH 45246-3732 U. S. A.
<i>Customers outside U. S. A.</i>	All:	Visit the support links at http://www.cincom.com to find contact information for your nearest Customer Service Center.

Contents

About this book	xi
Using this document	xi
MANTIS overview	xi
Document organization	xii
Conventions	xiii
MANTIS documentation series	xvi
Educational material	xvii
MANTIS SQL Support overview	19
Introduction	19
MANTIS SQL Support and non-SQL MANTIS applications	19
Embedding SQL statements in a MANTIS application program	20
Procedure for embedding an SQL statement	20
Sample code for an embedded SQL statement	20
How MANTIS processes embedded SQL statements	21
Host variables	21
Introduction to host variables	21
Input host variable (variable that provides input to the database)	22
Output host variable (variable that receives data from a database)	23
Host variables as parameters of SQL statements	24
Indicator variables	25
Description	25
Sample code	25
MANTIS SQL Support software requirements	26
Supported DB2 version	26
Binding MANTIS against a DB2 database	26
Differences between SQL in MANTIS and SQL in COBOL	27
Logical names	27
Static and dynamic SQL	28
Security	28

System maintenance	29
Master User facilities.....	29
MANTIS SQL options.....	30
Updating the User Profile.....	30
Specifying the default SQL DBTYPE.....	30
Default SQL DBTYPE and the current SQL DBTYPE.....	30
Current SQL DBTYPE and the MANTIS EXEC_SQL and SQLCA statements	30
Connecting to DB2.....	31
Explicit or implicit connect to DB2.....	31
Examples of explicit connection to DB2.....	31
 Embedding SQL statements in MANTIS programs	 33
Rules for embedding SQL statements in a MANTIS program.....	34
Follow the standard SQL syntax rules	34
Embed an SQL statement as a standard MANTIS comment.....	34
Follow the standard MANTIS comment rules	34
Place an SQL statement within an EXEC_SQL-END block	34
Place only one SQL statement within an EXEC_SQL-END block	35
Apart from the SQL statement, do not include any other MANTIS statements in an EXEC_SQL-END block	36
Do not place a comment or a MANTIS statement on the same line as an SQL statement	37
Within an EXEC_SQL-END block, use a colon to represent a host variable (the colon will not represent a new statement)	37
If desired, split an SQL statement across multiple comment lines within an EXEC_SQL-END block.....	38
In an SQL statement, use as many spaces as desired	39
If you use a colon to attach an SQL statement to an EXEC_SQL statement, that SQL statement exists within the EXEC_SQL-END block	40
If you place a MANTIS statement on the same line as the END in an EXEC_SQL-END block, MANTIS will not execute that statement	41
A MANTIS comment is permitted to be on the same line as the END in an EXEC_SQL-END block.....	41
Using host variables.....	42
Definition of a host variable	42
Sample code	42
Declaring host variables.....	43
Referencing values in a MANTIS array	44
MANTIS data types vs. SQL data types	45

Indicator variables	46
Definition.....	46
Creating an indicator variable.....	46
Interpreting an indicator variable.....	46
Sample code.....	46
Defining indicator variables	47
Data conversion between MANTIS SQL Support and DB2	48
Programming considerations	49
Chapter summary	49
MANTIS SQL Support and SQL statements	51
How to use MANTIS SQL Support	51
How MANTIS SQL Support processes SQL statements	51
SQL statements and cursors as SQL entities	51
Varying line numbers associated with an EXEC_SQL-END block	52
Line number associated with an EXEC_SQL-END block in a unbound program	52
Line number associated with an EXEC_SQL-END block in a bound program	53
Scope of cursors, statements, and SQLDA data structures	54
Basic elements of dynamic SQL programs	54
Local scope for cursor name, statement name, or SQLDA name	54
Mapping MANTIS cursor names and MANTIS statement names onto DB2 cursor names and DB2 statement names.....	54
MANTIS cursors and real cursor names	55
DB2 connection and disconnection	56
Connection to DB2	56
Disconnection from DB2.....	57
MANTIS EXEC_SQL statement, used for multiple session support.....	58
Syntax definition for the EXEC_SQL statement.....	59
SQL WHENEVER statement.....	61
Differences between WHENEVER statement in MANTIS SQL Support and WHENEVER statement in SQL in COBOL	61
Syntax for MANTIS SQL Support WHENEVER statement.....	61
Quick reference for WHENEVER conditions and default actions	66
Sample code for DO, FAULT, and CONTINUE	66
Declarative (COBOL) vs. interpretive (MANTIS) WHENEVER statements ..	67
Scope of the WHENEVER statement	68
SQLCA function and SQLCA statement	69
SQLCA in SQL in COBOL vs. SQLCA in MANTIS SQL Support	69
Syntax for SQLCA function and SQLCA statement	69
DBTYPE, an additional element for the SQLCA statement	73
SQLCA elements.....	74

COMMIT/ROLLBACK and COMMIT/RESET	75
Error messages.....	77
Message sources.....	77
Information that MANTIS generally displays for an error.....	77
Information that MANTIS displays for an error from the database system....	78
Dynamic SQL in MANTIS SQL Support	79
Who should read this chapter?	79
Dynamic SQL overview.....	79
Definition of dynamic SQL	79
Example of when to use dynamic SQL.....	79
Principal statements for dynamically executing SQL statements	80
SQLDA data structure.....	80
Defining data about SQL statements and host variables	81
SQL statements that you cannot execute dynamically.....	82
Auto-cursor FETCH statements.....	83
Executing a statement dynamically	83
Dynamic SQL sample code for creating an SQLDA data structure.....	84
SQLDA data structure.....	87
Figure representing SQLDA data structure	87
Data item.....	88
SQLDA names	88
Declaring SQLDA elements in other programming languages vs. declaring them in MANTIS SQL Support.....	88
Number of repeating elements in an SQLDA data structure	88
Built-in SQLDA statement and built-in SQLDA function	89
Sample code for the SQLDA statement and SQLDA function.....	90
Cursors for prepared statements	108
MANTIS SQL sample programs	109
Introduction	109
INSERT routines.....	110
Static INSERT routine.....	110
Dynamic INSERT routine.....	112
UPDATE routines.....	114
Static UPDATE routine	114
Dynamic UPDATE routine	115
SELECT routines	117
Static SELECT routine.....	117
Dynamic SELECT routine.....	118
DELETE routines	120
Static DELETE routine.....	120
Dynamic DELETE routine.....	121
Dynamic QUERY-like function.....	122
Dynamic column select.....	124

SQL features that are not supported for DB2 SQL	127
SQL features that are not supported for DB2 SQL	127
MANTIS SQL Support vs. SQL in COBOL	129
Differences between MANTIS SQL Support and SQL in COBOL	130
MANTIS vs. SQL	133
Differences between MANTIS and SQL.....	133
Index	135

About this book

Using this document

This guide describes MANTIS SQL Support for the version of DB2 that runs under UNIX. MANTIS SQL Support is an extended version of MANTIS that enables you to create MANTIS applications that use SQL to access database systems.

MANTIS overview

MANTIS[®] is an application development system that consists of the following:

- ◆ **A programming language.**
- ◆ **Design facilities.** For example:
 - Screens
 - Files

The MANTIS system is designed to increase your productivity in all areas of application development, from initial design through production and maintenance. MANTIS is a part of AD/Advantage, which offers additional tools for application development.

Document organization

Here are summaries for each chapter in this guide:

Chapter 1—MANTIS SQL Support overview

Provides an overview of MANTIS SQL Support. Describes using it to create MANTIS applications that use SQL.

Chapter 2—System maintenance

Supplements the information found in *MANTIS Administration, OpenVMS/UNIX*, P39-1320, with information specific to DB2. Provides information on Master User facilities, MANTIS SQL options, updating the User Profile, and signing on to DB2.

Chapter 3—Embedding SQL statements in MANTIS programs

Describes the rules that you must follow when embedding SQL statements in a MANTIS program.

Chapter 4—Programming considerations

Describes the program design implications resulting from the interpretive nature of MANTIS SQL Support.

Chapter 5—Dynamic SQL in MANTIS SQL Support

Discusses how dynamic SQL works in MANTIS SQL Support.

Appendix A—Sample MANTIS SQL programs

Provides examples of dynamic MANTIS SQL programs and their equivalent static MANTIS SQL programs.

Appendix B—Features not supported for DB2 SQL

Lists the SQL features that are not supported for DB2 SQL.

Appendix C—MANTIS SQL Support vs. SQL in COBOL

Summarizes the differences between MANTIS SQL Support and SQL in COBOL.

Appendix D—MANTIS vs. SQL

Summarizes the differences between MANTIS and SQL.

Index

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 <i>begin</i> LAST } </pre>
<p><u>Underlining</u> (In syntax)</p>	<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>	<pre> [<u>ON</u> OFF [<i>row</i>][<i>col</i>]] </pre>
	<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> <u>PROTECTED</u> </pre>
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> (<i>argument</i>, ...) </pre>

Convention	Description	Example
UPPERCASE	<p>Indicates MANTIS reserved words. You must enter them exactly as they appear.</p> <p>The example indicates that you must enter CONVERSE exactly as it appears.</p>	CONVERSE <i>name</i>
<i>Italics</i>	<p>Indicate variables you replace with a value, a column name, a file name, and so on.</p> <p>The example indicates that you can supply a name for the program.</p>	COMPOSE [<i>program-name</i>]
Punctuation marks	<p>Indicate required syntax that you must code exactly as presented.</p> <p>() parentheses . period , comma : colon ; semicolon ' single quotation mark " " double quotation marks</p>	$[\text{LET}]_v \left[\begin{matrix} (i) \\ (i, j) \end{matrix} \right] [\text{ROUNDED}(n)] = e1 [, e2, e3\dots]$

MANTIS documentation series

The manuals offered with MANTIS in the OpenVMS™ and UNIX® environments are listed below:

Getting started

- ◆ *MANTIS 2.8.01 Installation and Startup, OpenVMS/UNIX*, P39-0027*

General use

- ◆ *MANTIS Facilities, OpenVMS/UNIX*, P39-1300*
- ◆ *MANTIS Language, OpenVMS/UNIX*, P39-1310
- ◆ *MANTIS Messages and Codes, OpenVMS/UNIX*, P39-1330
- ◆ *MANTIS Application Development Tutorial, OpenVMS/UNIX*, P39-1340
- ◆ *MANTIS SUPRA SQL Programming, OpenVMS/UNIX*, P39-1345
- ◆ *AD/Advantage Programming*, P39-7001
- ◆ *AD/Advantage MANTIS Entity Transformers*, P39-0013
- ◆ *AD/Advantage Component Management Facility*, P19-2131
- ◆ *MANTIS Oracle Programming, UNIX*, P39-1355
- ◆ *MANTIS DB2 Programming, UNIX*, P39-1360
- ◆ *MANTIS WebSphere MQ Programming*, P39-1365



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

Master User tasks

- ◆ *MANTIS Facilities, OpenVMS/UNIX*, P39-1300*
- ◆ *MANTIS Administration, OpenVMS/UNIX*, P39-1320
- ◆ *MANTIS 2.8.01 Installation and Startup, OpenVMS/UNIX*, P39-0027*



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

1

MANTIS SQL Support overview

Introduction

MANTIS is an application development system that enables you to interactively perform the following for applications:

- ◆ Develop
- ◆ Test
- ◆ Execute
- ◆ Document

MANTIS SQL Support, an extended version of MANTIS, enables you to create MANTIS applications that use SQL to access database systems.

MANTIS SQL Support and non-SQL MANTIS applications

The presence of MANTIS SQL Support does not affect non-SQL MANTIS applications. Therefore, MANTIS SQL Support programs can run concurrently with, or in conjunction with, non-SQL MANTIS programs, without them affecting each other.

Embedding SQL statements in a MANTIS application program

Procedure for embedding an SQL statement

Embed SQL statements in a MANTIS application program as standard MANTIS comments. To embed an SQL statement, perform the following:

1. Using a text editor, open a MANTIS application program to which you would like to add an embedded SQL statement.
2. On a new line, enter an EXEC_SQL statement.
3. On the following line, enter a desired SQL statement.



Although you can enter only one SQL statement in an EXEC_SQL-END block, you can split this statement across multiple lines. Begin each line of your SQL statement with the comment character: a vertical bar (|).

4. On the following line, enter an END statement.

Sample code for an embedded SQL statement



The appearance of a MANTIS program with embedded SQL statements is similar to the appearance of the preprocessor output for SQL statements embedded in other host languages, such as COBOL.

See the following sample code for an embedded SQL statement:

```

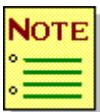
4580 TEXT EMP_NAME(30)
4590 BIT EMP_SAL
4600 WHILE SQLCA("SQLCODE") <> 1403
4610 . EXEC_SQL
4620 .. | SELECT SALARY
4630 .. | INTO :EMPL_SAL
4640 .. | FROM EMPLOYEE_TABLE
4650 .. | WHERE NAME = :EMP_NAME
4660 . END
4670 END
4680 DO BONUS_ROUTINE

```

How MANTIS processes embedded SQL statements

As MANTIS SQL Support encounters each SQL statement in a MANTIS program, MANTIS SQL Support performs the following:

1. Transparently prepares the SQL statement for execution.
2. Executes the SQL statement.



For further information on embedding SQL statements in MANTIS programs, see [“Embedding SQL statements in MANTIS programs”](#) on page 33.

Host variables

Introduction to host variables

MANTIS variables in SQL statements are called host variables. In an SQL statement, a colon always precedes a host variable. There are three kinds of host variables:

- ◆ **Input host variables.** See [“Input host variable \(variable that provides input to the database\)”](#) on page 22.
- ◆ **Output host variables.** See [“Output host variable \(variable that receives data from a database\)”](#) on page 23.
- ◆ **Host variables as parameters of SQL statements.** See [“Host variables as parameters of SQL statements”](#) on page 24.

Input host variable (variable that provides input to the database)

Description

An input host variable is a MANTIS variable that is:

- ◆ Passed to SQL
- ◆ Used to perform one of the following:
 - Select data
 - Insert data
 - Delete data
 - Update data.

The “input” in “input host variable” is from the perspective of the DB2 database, not the MANTIS program. Therefore, the input host variable provides output from the MANTIS program and input to the DB2 database.

Sample code

In the following sample code, “:EMP_NAME” is an input host variable:

```
4580 TEXT EMP_NAME(30)
4590 BIT EMP_SAL
4600 WHILE SQLCA("SQLCODE") <> 1403
4610 .EXEC_SQL
4620 .. | SELECT SALARY
4630 .. | INTO :EMPL_SAL
4640 .. | FROM EMPLOYEE_TABLE
4650 .. | WHERE NAME= :EMP_NAME
4660 .END
4670 END
4680 DO BONUS_ROUTINE
```

Output host variable (variable that receives data from a database)

Description

An output host variable is a MANTIS variable that receives data from a database. “Output” in “output host variable” is from the perspective of the DB2 database, not the MANTIS program. Therefore, the output host variable provides output from the DB2 database and input to the MANTIS program.

Sample code

In the following sample code, “:EMPL_SAL” is an output host variable.

```
4580 TEXT EMP_NAME(30)
4590 BIT EMP_SAL
4600 WHILE SQLCA("SQLCODE") <> 1403
4610 .EXEC_SQL
4620 .. | SELECT SALARY
4630 .. | INTO :EMPL_SAL
4640 .. | FROM EMPLOYEE_TABLE
4650 .. | WHERE NAME = :EMP_NAME
4660 .END
4670 END
4680 DO BONUS_ROUTINE
```

Host variables as parameters of SQL statements

Description

You can use host variables as parameters of SQL statements.

Sample code

In the following sample code, “:EMP_NAME” is a parameter of an SQL statement:

```
4580 TEXT EMP_NAME(30)
4590 BIT EMP_SAL
4600 WHILE SQLCA("SQLCODE") <> 1403
4610 .EXEC_SQL
4620 .. | SELECT SALARY
4630 .. | INTO :EMPL_SAL
4640 .. | FROM EMPLOYEE_TABLE
4650 .. | WHERE NAME= :EMP_NAME
4660 .END
4670 END
4680 DO BONUS_ROUTINE
```

Indicator variables

Description

Optionally, you can specify an indicator variable along with a host variable. The database system sets the indicator variable to indicate one of the following:

- ◆ Null value
- ◆ Value was truncated



For more information on indicator variables, see “[Indicator variables](#)” on page 46.

Sample code

In the following sample code, “:EMPLIV” and “:NAMEIV” are indicator variables:

```
EXEC_SQL: | SELECT EMPLNO, EMPLNA  
. | INTO :EMPLIV, :NAMEIV  
. | FROM EMPLOYEES WHERE DEPT = 17  
END
```

MANTIS SQL Support software requirements

Supported DB2 version

MANTIS SQL Support for DB2 works against DB2 UDB Release 7.1 databases.

Binding MANTIS against a DB2 database

Before you run the MANTIS program, the MANTIS DB2 administrator must perform a binding process in order to store a MANTIS package in the database.

Bind file information for the MANTIS DB2 administrator

Below is the required bind file information for the MANTIS DB2 administrator:

- ◆ **Name:** db2_int.bnd
- ◆ **Directory in which it is located:** \$MAN_ROOT/libdb2

Sample code

Below is sample code for binding MANTIS against a DB2 sample database:

```
CD $MAN_ROOT/libdb2
DB2 CONNECT TO SAMPLE USER DB2INST1 USING IBMDB2
DB2 BIND DB2_INT.BND
```



For detailed information about the DB2 BIND command, refer to *Command Reference* for DB2.

Differences between SQL in MANTIS and SQL in COBOL

The SQL implementation in MANTIS SQL Support is similar to the SQL implementations in third-generation languages like FORTRAN and COBOL.



For convenience, this guide refers to all third-generation SQL implementations as “SQL in COBOL.”

There are some differences between MANTIS SQL Support and SQL in COBOL. These differences are mostly due to the interpretive (as opposed to compiled) nature of MANTIS. For a summary of these differences, see “[MANTIS SQL Support vs. SQL in COBOL](#)” on page 129.

Logical names



In this guide, “logical name” refers to an identifier or variable which stands for another name or value.

OpenVMS and UNIX differ in their logical name implementations:

- ◆ **OpenVMS.** OpenVMS logical names correspond directly to MANTIS logical names.
- ◆ **UNIX.** UNIX implements logical names as environment variables. Ensure that any shell variables that must affect MANTIS are exported or inherited into the environment in which MANTIS is executing.

Static and dynamic SQL

A MANTIS SQL Support application can be either static or dynamic:

- ◆ **Static.** All SQL statements are defined before run time.
- ◆ **Dynamic.** All SQL statements are defined at run time. SQL statements are specified during program execution.

Security

Together, the database system and MANTIS handle security in MANTIS SQL Support. Ensure that users have authorized access to the views that they require.

2

System maintenance



The considerations in this chapter assume that you have already installed MANTIS. This chapter supplements *MANTIS Administration, OpenVMS/UNIX*, P39-1320.

Master User facilities

As the Master User, you alone can access certain facilities and information. When you sign on as MASTER, your Facility Selection menu appears as shown below:

```

M A N T I S

FACILITY SELECTION

Run A Program ..... 1      Sign On As Another User .... 11
Display A Prompter ..... 2  Transfer Facility ..... 12
Design A Program ..... 3    Edit MANTIS Messages ..... 13
Design A Screen ..... 4     Directory Facility ..... 14
Design A File Profile ..... 5 Universal Export Facility .. 15
Design A Prompter ..... 6   Update Shared Entity List .. 16
Design A User Profile ..... 7 Update Language Codes ..... 17
Design An Interface ..... 8  MANTIS Maintenance ..... 18
Design An Ultra File View .. 9 Spectra ..... 19
Design An External File View 10 Search Facility..... 20
                                List of Current Mantis Users. 21
                                Exit MANTIS ..... CANCEL

                                :      :
```

MANTIS SQL options

Two MANTIS options affect SQL support:

- ◆ SQLSSNINC
- ◆ SQLVARINC

For information on these options, refer to *MANTIS Facilities, OpenVMS/UNIX*, P39-1300.

Updating the User Profile

Specifying the default SQL DBTYPE

Specify the default SQL DBTYPE in the MANTIS User Profile's Default SQL DBTYPE field. In this field, specify one of the following SQL DBTYPEs:

- ◆ DB2
- ◆ ORACLE
- ◆ RDB
- ◆ SUPRA

Default SQL DBTYPE and the current SQL DBTYPE

When the user signs on to MANTIS, the default DBTYPE sets the user's current SQL DBTYPE. The current DBTYPE always determines which SQL database system MANTIS uses.

Current SQL DBTYPE and the MANTIS EXEC_SQL and SQLCA statements

The MANTIS EXEC_SQL and SQLCA statements can change the current DBTYPE at any time.



Not all DBTYPEs are supported on all platforms. For example:

- ◆ Rdb/VMS runs only in OpenVMS environments.
 - ◆ DB2 runs only in AIX environments.
-

Connecting to DB2



For considerations on connecting to DB2, and for information on disconnection from DB2, see “[DB2 connection and disconnection](#)” on page 56.

Explicit or implicit connect to DB2

You can connect to DB2 either explicitly or implicitly:

- ◆ **Explicit connect.** Occurs through the SQL CONNECT statement:

```
CONNECT [[TO {server - name}
         {host - variable}]

[USER {authorization - name}
      {host - variable} USING {password}
      {host - variable}]
```

- ◆ **Implicit connect.** Occurs when the first SQL statement is executed and there is no CONNECT statement. MANTIS for UNIX supports the implicit sign-on facility provided by DB2.

Examples of explicit connection to DB2

There are several possible variations on explicit connection:

Connecting to a DB2 server using all default values

Description. In this example, you connect to the DB2 server using default values for the following:

- ◆ Server
- ◆ Username
- ◆ Password

Sample code. See the following sample code:

```
EXEC_SQL
. | CONNECT
END
```

Connecting to a DB2 server called “SAMPLE”

Description. In this example, you connect to the DB2 server called “SAMPLE” using default values for the following:

- ◆ Username
- ◆ Password

Sample code. See the following sample code:

```
EXEC_SQL
. | CONNECT TO SAMPLE
END
```

Connecting to a DB2 server using MANTIS variables for username and password

Description. In this example, you connect to the default DB2 server using MANTIS variables for the following:

- ◆ Username
- ◆ Password

Sample code. See the following sample code:

```
EXEC_SQL
. | CONNECT USER :AIXUSER USING :PASSWORD
END
```

3

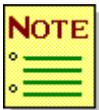
Embedding SQL statements in MANTIS programs



This chapter assumes a working knowledge of MANTIS, DB2, and SQL.

This chapter describes the following:

- ◆ **Rules for embedding SQL statements in a MANTIS program.** See “[Rules for embedding SQL statements in a MANTIS program](#)” on page 34.
- ◆ **How to reference host variables.** See “[Using host variables](#)” on page 42.
- ◆ **How to use indicator variables.** See “[Indicator variables](#)” on page 46.
- ◆ **How the database system converts data values between MANTIS SQL Support and the DB2 database.** “[Data conversion between MANTIS SQL Support and DB2](#)” on page 48.



For more information on MANTIS language conventions, refer to [MANTIS Language, OpenVMS/UNIX](#), P39-1310.

Rules for embedding SQL statements in a MANTIS program

Follow the standard SQL syntax rules

Follow the standard SQL syntax rules for SQL statements embedded within a MANTIS program. For these rules, refer to the *SQL Reference DB2* guide.

Embed an SQL statement as a standard MANTIS comment

Embed an SQL statement in a MANTIS application program as a standard MANTIS comment. That is, precede the SQL statement with a vertical bar (|).

Follow the standard MANTIS comment rules

For SQL statements embedded within a MANTIS program, follow the MANTIS comment rules. To find these rules, refer to *MANTIS Language, OpenVMS/UNIX*, P39-1310.

Place an SQL statement within an EXEC_SQL-END block

Rule

Place an SQL statement within an EXEC_SQL-END block.



Statements within the EXEC_SQL-END block are indented.

Sample code

See the following sample code:

```
EXEC_SQL
.| OPEN C1
END
```

Place only one SQL statement within an EXEC_SQL-END block

Rule

Place only one SQL statement within an EXEC_SQL-END block.



You can divide a single SQL statement across several lines.

Sample invalid code (what to avoid)

The following sample code is invalid because there are three separate SQL statements in the EXEC_SQL-END block, and only one is allowed.

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

Sample valid code

The following sample code is valid because it contains a single SQL statement that is divided across several lines:

```
4610 .EXEC_SQL
4620 .. | SELECT SALARY
4630 .. | INTO :EMPL_SAL
4640 .. | FROM EMPLOYEE_TABLE
4650 .. | WHERE NAME=:EMP_NAME
4660 .END
```

Apart from the SQL statement, do not include any other MANTIS statements in an EXEC_SQL-END block

Rule

Apart from the SQL statement, do not include any other MANTIS statements in an EXEC_SQL-END block. MANTIS SQL Support permits only the SQL statement within the SQL statement's EXEC_SQL-END block.

Sample invalid code (what to avoid)

In the following sample code, a MANTIS statement other than the SQL statement is located in the EXEC_SQL-END block:

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

Do not place a comment or a MANTIS statement on the same line as an SQL statement

Rule

Do not place a comment or another MANTIS statement on the same line as an SQL statement. Once MANTIS SQL Support encounters a vertical bar, MANTIS SQL Support considers the rest of the physical line to be a single SQL statement.

Sample invalid code (what to avoid)

See the following invalid sample code:

- ◆ **(Invalid code).** In the following sample code, a MANTIS statement is appended to a valid SQL statement:

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

- ◆ **(Invalid code).** In the following sample code, a MANTIS comment is appended to a valid SQL statement:

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

Within an EXEC_SQL-END block, use a colon to represent a host variable (the colon will not represent a new statement)

Rule

Use a colon within an EXEC_SQL-END block to represent a host variable. The colon will not represent a new statement, even though, outside of an EXEC_SQL-END block, the colon is the MANTIS statement-separator character.

Sample code

See the following sample code. In this code, “C1” is an SQL entity and “A” is a host variable.

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

If desired, split an SQL statement across multiple comment lines within an EXEC_SQL-END block

Rule

If desired, split an SQL statement across multiple comment lines within an EXEC_SQL-END block. MANTIS reads the text in two or more consecutive comment lines within the same EXEC_SQL-END block as if it were separated by a single blank. That is, all of the text lines, together, compose one statement.



SQL text literals (characters between apostrophes) may not span multiple comment lines.

Sample code

The following two pieces of sample code are equivalent:

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

In an SQL statement, use as many spaces as desired

Rule

In an SQL statement, use as many spaces as desired. MANTIS SQL Support treats multiple blanks at the beginning or end of an SQL statement, or spaces between words on the same line, as a single blank.

Sample code

Consider the usage of spaces in the following sample code:

- ◆ The following two pieces of sample code are equivalent:

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

- ◆ The following two pieces of sample code are equivalent:

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

If you use a colon to attach an SQL statement to an EXEC_SQL statement, that SQL statement exists within the EXEC_SQL-END block

Rule

If you use a colon to attach an SQL statement to an EXEC_SQL statement, that SQL statement exists within the EXEC_SQL-END block. The SQL statement is considered to all or part of the single SQL statement that is allowed within the EXEC_SQL-END block.

Sample code

Consider the following sample code, in which the first part of the SQL statement is attached to the EXEC_SQL statement and the second part of the SQL statement is within the EXEC_SQL-END block:

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

If you place a MANTIS statement on the same line as the END in an EXEC_SQL-END block, MANTIS will not execute that statement

Rule

If you place a MANTIS statement on the same line as the END in an EXEC_SQL-END block, MANTIS will not execute that statement. This is consistent with the way MANTIS treats a MANTIS statement added to an END in each of the following kinds of MANTIS block:

- ◆ IF-END
- ◆ WHILE-END
- ◆ FOR-END
- ◆ WHEN-END
- ◆ UNTIL-END

Sample code

See the following sample code, in which MANTIS disregards “OPENED=TRUE”:

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

A MANTIS comment is permitted to be on the same line as the END in an EXEC_SQL-END block

Rule

A MANTIS comment is permitted to be on the same line as the END in an EXEC_SQL-END block.

Sample code

Consider the following sample code, in which the comment is on the same line as the END:

```
EXEC_SQL
.| OPEN C1
END : | C1 IDENTIFIES TAG FILE ENTRIES
```

Using host variables

Definition of a host variable

A “host variable” is a MANTIS variable in an SQL statement that provides input to, or receives output from, the connected database. Create a host variable within an SQL statement by prefixing a variable name with a colon.

Sample code

See the following sample code, in which “:SALARY” is a host variable:

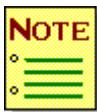
```
EXEC_SQL
.| INSERT INTO OWNER.TAB (COLA)
.| VALUES (:SALARY * 1.1)
END
```

Declaring host variables

Description

You may define a variable explicitly or implicitly:

- ◆ **Explicit definition.** You may explicitly declare a host variable before it appears in the SQL statement, and you can declare this variable to be any type.
- ◆ **Implicit definition.** If you do not explicitly declare a host variable, MANTIS implicitly declares the host variable the first time MANTIS uses it (as MANTIS does with other MANTIS variables). MANTIS automatically declares an implicitly-defined host variable in an SQL statement as a MANTIS BIG variable.



A MANTIS BIG variable is a numeric floating-point variable that is 16 digits long.

Sample code

See the following sample code for explicit and implicit definition. The two pieces of sample code are equivalent.

- ◆ **Explicit definition.** See the following sample code:

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

- ◆ **Implicit definition.** See the following sample code:

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

Referencing values in a MANTIS array

Description

You can use a host variable as an item in a MANTIS array. Use arithmetic expressions and MANTIS functions to specify subscripts of host variables.

Sample code

In the following sample code, a subscript of a host variable refers to an item in a MANTIS array:



In the following sample code, all code that comes after the colon must conform to MANTIS syntax. MANTIS rules apply to subscripting, even though the subscript is located in an SQL statement.



You can prefix only the host variable with a colon. You cannot use a colon to prefix any of the other MANTIS variables to which you refer in subscript expressions. In the following sample code, a colon does not prefix the variables N and T, and these variables are assumed to be MANTIS variables.

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

MANTIS data types vs. SQL data types

Database system to MANTIS translation

When the connected database system transfers data from one of its variables to a host variable, MANTIS causes the database system to automatically convert the data's type from an SQL type to a MANTIS type.

MANTIS to database system translation

When MANTIS transfers data from one of its host variables to the connected database system, MANTIS causes the database to automatically convert the data's type from a MANTIS type to an SQL type.

More information on data type translation

For a summary of how database systems convert to and from MANTIS data types, see [“Data conversion between MANTIS SQL Support and DB2”](#) on page 48.

Risks of data type translation

During data-type-translation from a database variable to an SQL variable, any of the following problems may occur:

- ◆ Rounding
- ◆ Truncation
- ◆ Overflow

Indicator variables

Definition

Optionally, you can include an indicator variable along with a host variable in an SQL statement. As its name implies, the indicator variable indicates whether the host variable:

- ◆ Contains a real value
- ◆ Is NULL or MISSING

Creating an indicator variable

Prefix an indicator variable name with a colon. Place the indicator variable immediately after the corresponding host variable or subscript expression.

Interpreting an indicator variable

Interpret an indicator variable as follows:

- ◆ **If the indicator variable contains zero:** The host variable contains a defined value. No error has occurred.
- ◆ **If the indicator variable contains a value that is less than zero:** The host variable contains a NULL or MISSING value.

Sample code

In the following sample code, EMPLIV and NAMEIV are indicator variables:

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

Defining indicator variables

Explicit vs. implicit definition

Like host variables, indicator variables can be defined explicitly or implicitly:

- ◆ **Explicit definition.** Consider the following:
 - When you explicitly define a variable, you can use only a numeric variable as an indicator variable.



For descriptions of variable types, see “[Data conversion between MANTIS SQL Support and DB2](#)” on page 48.

- If you explicitly define an indicator variable as a floating point value, DB2 still interprets it as an integer. Therefore, a value of -0.9 in an indicator variable will not specify a NULL or MISSING value (that is, a value that is less than zero). Instead, the value will be converted to zero before it is interpreted.
- ◆ **Implicit definition.** By default, MANTIS defines the indicator variable as a MANTIS BIG variable.

Supplying indicator values for any columns that may contain NULL values

When using SELECT or FETCH to read data from the database, supply an indicator variable for each column that may contain a NULL value. Check the value of the indicator variable before examining the host variable data. If the indicator variable shows that the column is NULL, the value of the host variable is undefined.

Data conversion between MANTIS SQL Support and DB2

DB2 performs data conversion between all data types (including between numeric and string data types). Be sure to match host variable data types with database columns so that you obtain correct results.

The following table shows valid data type conversions:

SQL data type	MANTIS data type(s)	Considerations
DECIMAL	BIG, SMALL	Consider the following: <ul style="list-style-type: none"> ◆ When converting from SQL to MANTIS, loss of precision may occur. ◆ When converting from MANTIS to SQL, overflow may occur.
INTEGER	BIG, SMALL	Consider the following: <ul style="list-style-type: none"> ◆ When converting from SQL to MANTIS, loss of precision may occur. ◆ When converting from MANTIS to SQL, overflow may occur.
SMALLINT	BIG, SMALL	When converting from MANTIS to SQL, overflow may occur.
FLOAT	BIG, SMALL	When converting from an SQL FLOAT variable to a MANTIS SMALL variable, overflow and/or loss of precision may occur.
CHAR VARCHAR	TEXT	Consider the following: <ul style="list-style-type: none"> ◆ When converting from SQL to MANTIS, truncation may occur. ◆ When converting from MANTIS to SQL, truncation may occur. ◆ The string can have a variable length, from 1–65535 characters.
DATE	TEXT	If TEXT size is less than 10, truncation may occur.
TIME	TEXT	If TEXT size is less than 8, truncation may occur.
TIMESTAMP	TEXT	If TEXT size is less than 26, truncation may occur.

4

Programming considerations

MANTIS SQL Support's interpretive nature has several program design implications. Before you begin writing MANTIS SQL Support programs, learn about these implications, which are described throughout this chapter.

Chapter summary

This chapter contains the following sections:

- ◆ **MANTIS SQL Support and SQL statements.** Briefly describes several important aspects of MANTIS SQL statements. See [“MANTIS SQL Support and SQL statements”](#) on page 51.
- ◆ **Varying line numbers associated with an EXEC_SQL-END block.** Internally, MANTIS stores an EXEC_SQL-END block as a single line of text. This section describes how MANTIS associates various line numbers, contained within the block, with this single line of text. See [“Varying line numbers associated with an EXEC_SQL-END block”](#) on page 52.
- ◆ **Scope of cursors, statements, and named SQLDA data structures.** Describes the local scope of cursors, statements, and SQLDA data structures, as well as the mapping of global real cursor names to MANTIS cursor names. See [“Scope of cursors, statements, and SQLDA data structures”](#) on page 54.
- ◆ **DB2 connection and disconnection.** Lists considerations for using MANTIS SQL Support to connect to, and disconnect from, DB2. See [“DB2 connection and disconnection”](#) on page 56.

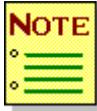
- ◆ **MANTIS EXEC_SQL statement, used for multiple session support.** Describes the EXEC_SQL statement syntax and describes how to use the MANTIS EXEC_SQL statement for multiple session support. See “[MANTIS EXEC_SQL statement, used for multiple session support](#)” on page 58.
- ◆ **SQL WHENEVER statement.** Lists considerations for using the WHENEVER statement. Also lists differences between the WHENEVER statement in MANTIS and in other languages. See “[SQL WHENEVER statement](#)” on page 61.
- ◆ **SQLCA function and SQLCA statement.** Describes differences between SQLCA access in COBOL and SQLCA access in MANTIS SQL Support. Also describes the elements that compose the SQLCA function and the SQLCA statement. See “[SQLCA function and SQLCA statement](#)” on page 69.
- ◆ **COMMIT/ROLLBACK and COMMIT/RESET.** Lists considerations for MANTIS’s COMMIT/ROLLBACK statement and MANTIS SQL Support’s COMMIT/RESET statement. See “[COMMIT/ROLLBACK and COMMIT/RESET](#)” on page 75.
- ◆ **Error messages.** This section provides general information about all MANTIS error messages and more specific information about MANTIS error messages from a database. See “[Error messages](#)” on page 77.

MANTIS SQL Support and SQL statements

How to use MANTIS SQL Support

To use MANTIS SQL Support, perform the following:

1. Place an appropriate SQL statement in a MANTIS comment.



Identify a line as a MANTIS comment by beginning the line with a vertical bar (|).

2. Enclose the MANTIS comment in an EXEC_SQL-END block.
3. Place the EXEC_SQL-END block, containing the MANTIS comment, in your MANTIS application program.
4. Repeat steps 1-3 for as many SQL statements as you require in your MANTIS application program.

How MANTIS SQL Support processes SQL statements

As MANTIS SQL Support encounters each SQL statement, it prepares the SQL statement for execution and then executes the SQL statement. In effect, it performs the same steps (preprocess, compile, link, and load before the run) that are performed for a COBOL program containing embedded SQL statements.

Unlike a COBOL program, the MANTIS program (including its SQL statements) can be modified and then, through the RUN command, immediately re-executed.

SQL statements and cursors as SQL entities

Both cursors and SQL statements are SQL entities and not MANTIS entities. Because of this, you cannot perform the following:

- ◆ Pass them as parameters
- ◆ Use them in non-SQL MANTIS statements

Varying line numbers associated with an EXEC_SQL-END block

An EXEC_SQL-END block may span several program lines. However, when you execute the EXEC_SQL-END block, MANTIS SQL Support stores the block internally as a single line of text.

Line number associated with an EXEC_SQL-END block in an unbound program

Description

MANTIS SQL Support stores an EXEC_SQL-END block internally as a single line of text. If the program is unbound, MANTIS associates this single line with different program lines for different purposes:

- ◆ **For executing the program:** For the purpose of program execution, MANTIS associates the EXEC_SQL-END block with the first line in the EXEC_SQL-END block. For example, to execute the following sample code, enter “RUN 10”.
- ◆ **For labeling an error message:** For the purpose of labeling an error-message, MANTIS associates the EXEC-SQL-END block with the last line in the EXEC_SQL-END block. Therefore, if MANTIS encounters an error in the following sample code, it returns the appropriate error message and displays “40” for line 40.

Sample code

See the following sample code:

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

Line number associated with an EXEC_SQL-END block in a bound program

Description

MANTIS SQL Support stores an EXEC_SQL-END block internally as a single line of text. If the program is bound, MANTIS associates this single line with the line number of the last SQL statement (the line immediately preceding the END statement).

For example, to execute the following sample code, enter "RUN 30." If MANTIS encounters an error in the program block, it returns the appropriate error message and displays "30" for line 30.

Sample code

See the following sample code:

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

Scope of cursors, statements, and SQLDA data structures

Basic elements of dynamic SQL programs

The following are identified by names and are basic to dynamic SQL programs:

- ◆ Cursors
- ◆ Statements
- ◆ SQLDA data structures

Local scope for cursor name, statement name, or SQLDA name

The scope of a cursor name, statement name, or SQLDA data structure name is local. That is, it is limited to the program or external subprogram context in which the name is declared. When a cursor name, statement name, or SQLDA name is redeclared in an external subprogram, the name refers to a different structure.

Mapping MANTIS cursor names and MANTIS statement names onto DB2 cursor names and DB2 statement names

MANTIS maps cursor names and statement names onto DB2 cursor names and statement names. The scope of the DB2 cursor names and statement names is one of the following:

- ◆ MANTIS main program
- ◆ Entire connect time to the DB2 database

MANTIS cursors and real cursor names

The mapping of cursor names requires further explanation, since certain error conditions relating to open cursors can cause error messages to display real cursor names.

As each MANTIS cursor is opened, the MANTIS nucleus allocates and maps a global real cursor name of the form “Cnnn” (where “nnn” is a three-digit number allocated by MANTIS) to the MANTIS cursor name. Therefore, each MANTIS external subprogram maps a range of real cursor names.

When the EXIT subprogram executes, MANTIS closes all real cursors that are open. This frees up that range of real cursor names for the next subprogram to be called or for the same subprogram to be invoked again.

The limited scope of a MANTIS cursor name is illustrated by the following mapping:

- ◆ MANTIS cursor C1 in main program MAIN might be mapped to real cursor C001.
- ◆ MANTIS cursor C1 in external subprogram SUB might be mapped to real cursor C006.

DB2 connection and disconnection



For the difference between explicit and implicit connection, and for examples of explicit connection to DB2, see “[Connecting to DB2](#)” on page 31.

Connection to DB2

MANTIS supports the DB2 type 1 database connection.



For more information on the DB2 type 1 database connection, refer to the *SQL Reference DB2* manual.

Connection considerations

Consider the following for a MANTIS connection to DB2:

- ◆ **One concurrent connection.** A MANTIS program can only connect to one DB2 application server at a time.
- ◆ **“DB2 implicit connect” and implicit connection.** If a “DB2 implicit connect” is available, the MANTIS program implicitly connects to the default DB2 application server.
- ◆ **Explicit connection.** A MANTIS program can explicitly connect to a server that is different from the currently-connected DB2 server.

Disconnection from DB2

SQL CONNECT RESET statement

MANTIS detaches from the DB2 database by executing the SQL CONNECT RESET statement. MANTIS can execute the CONNECT RESET statement in two ways:

- ◆ **Explicitly.** Performed by a MANTIS program.
- ◆ **Implicitly.** Performed by the MANTIS nucleus.

Possible causes of disconnection

Possible causes of disconnection are:

- ◆ **A MANTIS program executes an SQL CONNECT RESET statement.** The format for this statement is:

```
EXEC_SQL: | CONNECT RESET END
```
- ◆ **MANTIS main program context cleanup.** This occurs in the following circumstances:
 - The MANTIS nucleus releases the current TEST program context in Program Design as the result of a NEW, LOAD, EDIT, or RUN command.
 - The user exits the Program Design Facility.
 - A main program terminates in RUN mode, when that main program is not under Program Design's control.
- ◆ **A MANTIS program executes a MANTIS CHAIN statement.** Disconnection occurs only if the MANTIS option for database sign-off on a CHAIN statement is enabled.
- ◆ **A CONNECT TO statement changes the DB2 application server that is connected to the MANTIS program.**

MANTIS EXEC_SQL statement, used for multiple session support

This section discusses the use of EXEC_SQL for multiple session support.



This guide's examples have already demonstrated the EXEC_SQL statement's common usage. See the following:

- ◆ “Embedding SQL statements in MANTIS programs” on page 33
 - ◆ “Place an SQL statement within an EXEC_SQL-END block” on page 34
-

Syntax definition for the EXEC_SQL statement

Below is the syntax definition for the EXEC_SQL statement:

```
EXEC_SQL [ (exp1 [, exp2]) ] [: | sql - statement ]
```

```
[
    sql - statement continued
]
```

```
END
```

exp1

Description When you require multiple DB2 database connections, specify one of the following:

- ◆ Database subsystem type (DBTYPE)
- ◆ DB2 SQL CONNECT session number

Format One of the following:

- ◆ **Text expression.** One of these text expressions:
 - SUPRA
 - RDB
 - ORACLE
 - DB2
- ◆ **Numeric expression.** A session number ranging from one through five. (If you use a numeric session number, the current DBTYPE must be DB2.)

exp2

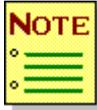
Description *Optional.* When you have used *exp1* to specify SUPRA or ORACLE as the DBTYPE, use *exp2* to specify the connect session number.

Format A numeric expression, equal to a session number ranging from one through five.

General considerations

Consider the following:

- ◆ **Multiple session support.** This refers to the following:
 - The ability to connect to different SQL databases concurrently; for example, to both SUPRA and ORACLE databases.
 - The ability to have multiple connections to a single database. You can only do this with certain databases, such as SUPRA.



You can only connect to one Rdb/VMS or DB2 database at a time.

- ◆ **Current DBTYPE.** The current DBTYPE, an element of the EXEC_SQL statement, is the default DBTYPE that is used when a DBTYPE is not specified in an EXEC_SQL statement. The current DBTYPE can be changed in two ways:
 - Explicitly—Performed by another EXEC_SQL or SQLCA statement.
 - Implicitly—Performed when you sign on to another MANTIS user.

The Master User can specify your default current DBTYPE in your MANTIS user profile; if he or she does not, MANTIS uses SUPRA as the default current DBTYPE.



Not all DBTYPEs are supported on all platforms. For example:

- ◆ Rdb/VMS support is only available on the OpenVMS environment.
 - ◆ DB2 support is only available on the AIX platform.
-

SQL WHENEVER statement

Differences between WHENEVER statement in MANTIS SQL Support and WHENEVER statement in SQL in COBOL

The WHENEVER statement in MANTIS SQL Support differs from the one in SQL in COBOL in four ways:

- ◆ It is interpretive and not compiled.
- ◆ MANTIS DO replaces GOTO.
- ◆ FAULT is an extra WHENEVER action that allows program termination upon occurrence of a specified condition.
- ◆ The default for SQLERROR is FAULT, not CONTINUE.

Syntax for MANTIS SQL Support WHENEVER statement

Below is the syntax for the MANTIS SQL Support WHENEVER statement:



Any action (DO, FAULT, or CONTINUE) can be selected for any condition (SQLERROR, SQLWARNING, or NOT FOUND).

WHENEVER condition action

condition

Description *Required.* Specifies the condition for which you are checking.

Options Valid conditions are:

- ◆ SQLERROR
- ◆ SQLWARNING
- ◆ NOT FOUND

These conditions are explained below:

SQLERROR

Description *Optional.* Indicates the following:

- ◆ The database returned an error code as the result of an SQL statement.
- ◆ SQLCODE < 0.

Default action FAULT

SQLWARNING

Description *Optional.* Indicates the following:

- ◆ SQLCA("SQLWARN0") = "W"
- ◆ SQLCODE = 0

Default action CONTINUE

NOT FOUND

Description *Optional.* Indicates just one of the following:

- ◆ The database cannot find a row to satisfy your SQL statement.
- ◆ There are no more rows to fetch (SQLCODE = 100).

Default action CONTINUE

action

Description *Required.* Specifies the action to be taken when the specified condition is met.

Options Valid actions are the following:

- ◆ DO entry-name[(parms)]
- ◆ FAULT
- ◆ CONTINUE.

DO entry-name[(parms)]

Description *Optional.* Specifies one of the following:

- ◆ Standard MANTIS internal DO
- ◆ Standard MANTIS external DO

Corresponds to the WHENEVER-GOTO SQL statement of SQL in COBOL's WHENEVER-DO statement. Whenever the named condition is encountered, DO transfers control to the specified internal subroutine or external program.

Considerations

- ◆ The following steps are performed:
 1. WHENEVER-DO transfers control from the calling program to an internal subroutine or to an external program. In the process, WHENEVER-DO passes the DO argument values (current at the execution time of the EXEC_SQL statement preceding the DO statement) to the internal subroutine or to the external program.
 2. The internal subroutine or external program executes its logic. This logic can include any MANTIS logic, such as CHAIN, EXIT, or STOP statements.
 3. The EXIT subroutine returns control to the calling program. The calling program resumes at the first statement following the EXEC_SQL-END block in which the DO is located.
- ◆ The WHENEVER-DO statement resembles the existing functionality of the SET TRAP statement in MANTIS. If the DO portion of a WHENEVER-DO statement contains an error, MANTIS returns a MANTIS error message associated with the DO statement, not an SQL WHENEVER-type error. With the error message, MANTIS displays the number of the subroutine line that contains the error. If any part of a WHENEVER statement contains an error, MANTIS will detect that error whether the WHENEVER statement is in the execution path or not.



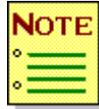
DO is executed as a result of an SQL statement raising the condition with which the DO action is associated.

FAULT

Description

Optional. Performs the following:

- ◆ Terminates program execution.
- ◆ Displays the generated database system message in the form of a MANTIS fault (error) message.



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

CONTINUE

Description

Optional. When the named condition occurs, CONTINUE permits program execution to proceed without interruption. After the named condition occurs, your program should check SQLCODE for the results of each EXEC_SQL.

Quick reference for WHENEVER conditions and default actions

The following table provides a quick reference for each WHENEVER condition and its associated default action.

WHENEVER Condition	Default action
SQLERROR	FAULT
SQLWARNING	CONTINUE
NOT FOUND	CONTINUE

Sample code for DO, FAULT, and CONTINUE

Consider the following sample code for DO, FAULT, and CONTINUE:

```
200 |  
210 | SET 'WHENEVER' SETTINGS TO DESIRED VALUES  
220 |  
230 EXEC_SQL  
235 . | WHENEVER SQLERROR DO DO_ROUTINE(PARM1, PARM2, PARM3)  
240 END  
250 EXEC_SQL: | WHENEVER SQLWARNING FAULT  
260 END  
270 EXEC_SQL: | WHENEVER NOT FOUND CONTINUE  
280 END
```

Declarative (COBOL) vs. interpretive (MANTIS) *WHENEVER* statements

Declarative *WHENEVER* statement for SQL embedded in COBOL

When SQL is embedded in COBOL, *WHENEVER* is a declarative statement. *WHENEVER* is processed when the user precompiles the program, rather than when the user executes the program. Thus, in a COBOL program, the current “*WHENEVER* setting”—that is, the combination of condition (such as *SQLWARNING*) and action (such as *CONTINUE*) specified by the *WHENEVER* statement—is determined by sequential position, regardless of the execution path at runtime.



A *WHENEVER* statement can have three settings:

- ◆ One setting for the condition *SQLERROR*
 - ◆ One setting for the condition *SQLWARNING*
 - ◆ One setting for the condition *NOT FOUND*
-

Interpretive *WHENEVER* statement for SQL embedded in MANTIS

When SQL is embedded in MANTIS, *WHENEVER* is an interpretive statement. The last-executed *WHENEVER* statement is in effect, regardless of its position in the program sequence.

When the difference between MANTIS SQL Support and SQL in COBOL is important

The difference between MANTIS SQL Support (last-executed WHENEVER being in effect) and SQL in COBOL (most recent WHENEVER in program sequence being in effect) is important when you use a WHENEVER statement with conditional statements.

The following figure shows the different effects of a declarative vs. interpretive WHENEVER statement:

SQL in COBOL 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*

SQL in COBOL: Since the setting is established before run time, the setting remains unchanged regardless of whether the program executes lines 50–70.

MANTIS SQL Support: The first time the program executes line 40, the setting is C1; the setting is thereafter C2. * If the WHILE condition is not true the first time the program executes line 40, C1 remains the setting through line 80 because the program did not execute line 50.



The consideration regarding WHENEVER in MANTIS SQL Support vs. WHENEVER in SQL in COBOL also apply to the following MANTIS structures:

- ◆ FOR
- ◆ UNTIL
- ◆ WHEN
- ◆ IF

Scope of the WHENEVER statement

The scope of the WHENEVER statement is one of the following:

- ◆ The current MANTIS DOLEVEL
- ◆ Every EXEC_SQL until a new WHENEVER is executed

If you do not want the default WHENEVER settings, have each externally-done program issue WHENEVER.

SQLCA function and SQLCA statement

SQLCA in SQL in COBOL vs. SQLCA in MANTIS SQL Support

SQLCA in SQL in COBOL

In SQL in COBOL, the SQLCA (SQL Communications Area) is a data structure. An SQL in COBOL application accesses elements in the SQLCA as data items.

SQLCA in MANTIS SQL Support

In MANTIS SQL Support, the following two items provide all standard SQLCA capabilities:

- ◆ **SQLCA function.** For reading SQLCA structure elements.
- ◆ **SQLCA statement.** For writing SQLCA structure elements.

Syntax for SQLCA function and SQLCA statement

Syntax overview

Below is the syntax for:

- ◆ **SQLCA function.** For reading SQLCA structure elements.

sqlca-element = SQLCA(element_name)

- ◆ **SQLCA statement.** For writing SQLCA structure elements.

SQLCA(element_name) = sqlca-element-value

Syntax elements for the SQLCA function and SQLCA statement

Consider the following elements:

sqlca-element

Description *Required.* Specifies a MANTIS variable or array element to receive the value of your SQLCA element.

Format Valid MANTIS variable reference:

- ◆ Scalar variable
- ◆ Subscripted array
- ◆ Substring reference

Consideration The data type of *sqlca-element* must be compatible with the data type of the SQLCA element referenced as *element_name* in the SQLCA function.

element_name

- Description** *Required.* Specifies one of the following:
- ◆ A text literal that is the name of the element to be returned or read.
 - ◆ A text variable containing the name of the element to be returned or read.

Format Text expression evaluating to a valid SQLCA element, as listed in “SQLCA elements” on page 74.

Consideration You can specify *element_name* in two different ways:

- ◆ **As a text literal.** When you specify *element_name* as a text literal, you must use quotation marks (“”). For example:

```
.IF SQLCA ("SQLCODE") < ZERO
..DO ERROR_CONDITION_ROUTINE
.END
```

- ◆ **In a text variable.** For example:

```
.CACODE="SQLCODE"
.IF SQLCA (CACODE) < ZERO
..DO ERROR_CONDITION_ROUTINE
.END
```



Because the SQLCA function is built in, it is not declared. Cincom does not require or recommend an INCLUDE SQLCA statement.

sqlca-element-value

Description *Required.* Specifies a value to be assigned to your SQLCA element.

Format Text or numeric MANTIS expression.

Consideration The data type of *sqlca-element-value* must be compatible with the data type of the SQLCA element referenced as *element_name* in the SQLCA statement.

Cincom has added an additional element, DBTYPE, to MANTIS SQL Support. The following section describes it.

DBTYPE, an additional element for the SQLCA statement

The DBTYPE element for the SQLCA statement enables you to specify the database with which MANTIS SQL Support will communicate.

Syntax

Below is the syntax for the SQLCA statement's additional element:

SQLCA("DBTYPE")

Syntax element

Below is the additional element for the SQLCA statement:

DBTYPE

Description Specifies a text value, one to six characters in length, for the current DBTYPE.



For more information on the DBTYPE, see ["Updating the User Profile"](#) on page 30.

Format The following DBTYPE values are valid:

- ◆ SUPRA
- ◆ ORACLE
- ◆ RDB
- ◆ DB2

Consideration By default, after execution of the SQLCA("DBTYPE") statement, all executed EXEC_SQL statements may access the relational database implied by the DBTYPE.

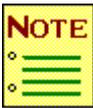
SQLCA elements

The following table lists the compatible MANTIS data type for each SQLCA element name.



If you move a data value from an SQLCA element to a MANTIS variable of shorter length, some of the right-hand characters are truncated. For example, this occurs if you move an eight-character SQLCA element to a six-character MANTIS variable.

SQLCA element name	Compatible MANTIS data type	Element contents	Element considerations
SQLCAID	TEXT(8)		Eyecatcher. Set by SQL.
SQLCABC	BIG	Length of SQLCA.	Set by SQL.
SQLCODE	BIG	Code indicating results of SQL statement execution.	
SQLERRMC	TEXT(70)	Tokens for insertion into SQL error message text.	
SQLERRP	TEXT(8)	SQL diagnostic data.	
SQLERRDn	BIG	SQL diagnostic data.	<i>n</i> ranges from 1–6.
SQLWARNn	TEXT(1)	SQL warning flags.	<i>n</i> ranges from 0–A.
SQLSTATE	TEXT(5)	Indicates the result of SQL statement execution.	

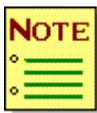


For portable, nonvolatile MANTIS software, consider the entire SQLCA structure to be read-only.

COMMIT/ROLLBACK and COMMIT/RESET

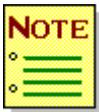
Regarding SQL's COMMIT/ROLLBACK statement and MANTIS SQL Support's COMMIT/RESET statement, consider the following:

- ◆ **RELEASE parameter for COMMIT/ROLLBACK.** Cincom recommends that you do not use the COMMIT/ROLLBACK RELEASE parameter. This parameter is a request to disconnect from the database upon successful completion of the COMMIT or ROLLBACK. Rather than the RELEASE parameter, use the SQL CONNECT RESET statement to disconnect from the database.
- ◆ **DB2 and COMMIT.** By default, the MANTIS nucleus automatically issues COMMIT every time MANTIS reads from the terminal for a CONVERSE, OBTAIN, or WAIT statement. Be careful when using COMMIT and RESET in applications with embedded SQL.
- ◆ **COMMIT/ROLLBACK vs. COMMIT/RESET.** Regarding COMMIT/ROLLBACK vs. COMMIT/RESET, consider the following:
 - MANTIS SQL Support's COMMIT/ROLLBACK statement has exactly the same effect on the database as the MANTIS COMMIT/RESET statement.
 - An executed COMMIT/ROLLBACK does not imply a COMMIT/RESET. However, an executed COMMIT/RESET implies a COMMIT/ROLLBACK.
 - The MANTIS COMMIT/RESET statements COMMIT/ROLLBACK the current SQL transaction. Embedded SQL COMMIT/ROLLBACK statements affect only the SQL database.
- ◆ **Scope of an SQL COMMIT statement.** Execution of an SQL COMMIT statement commits only SQL, and only for the specified SQL session.
- ◆ **COMMIT and terminal input.** MANTIS automatically performs a COMMIT at terminal input.



COMMIT/RESET can affect SQL cursor position. For more information on cursor positioning, refer to the *SQL Reference DB2* guide.

- ◆ **When MANTIS COMMIT executes.** MANTIS COMMIT executes in the following circumstances:
 - MANTIS encounters a MANTIS COMMIT statement.
 - Any MANTIS program (including Program Design, when you are reading command lines) encounters any terminal input function (CONVERSE, OBTAIN, WAIT, or MORE prompt), unless you have specified COMMIT OFF.
 - When MANTIS runs the main program cleanup, before database disconnection.
- ◆ **When MANTIS RESET executes.** MANTIS RESET executes in the following circumstances:
 - MANTIS encounters a MANTIS RESET statement.
 - A MANTIS FAULT (excluding a breakpoint fault) occurs, .
- ◆ **Be careful of a terminal input request by Program Design.** When Program Design makes a terminal input request, beware of the request's effect. Your MANTIS program may go into a resource wait state because it is attempting to update a table that another user is reading (via SELECT).
- ◆ **Be careful of COMMIT/RESET.** Carefully consider the use of COMMIT and RESET in your embedded SQL applications.



If you are in Program Design and have set a breakpoint, a RESET will not occur when program execution encounters a breakpoint (the breakpoint is a FAULT condition). However, unless you have performed a COMMIT OFF, an automatic COMMIT will occur when Program Design prompts you for the next input line. This could result in unintentional COMMITs of DB2 transactions.



When a MANTIS COMMIT fails to COMMIT DB2, MANTIS automatically attempts to ROLLBACK only the DB2 database.

Error messages

Message sources

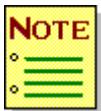
You can receive messages from three sources:

- ◆ MANTIS nucleus
- ◆ MANTIS SQL Support
- ◆ Database system

Information that MANTIS generally displays for an error

When MANTIS encounters an error, it generally displays the following:

- ◆ Fault message
- ◆ Statement in which the error occurred
- ◆ Text of the line containing the statement in which the error occurred



For details on all of MANTIS's error messages, refer to *MANTIS Messages and Codes, OpenVMS/UNIX*, P39-1330.

Information that MANTIS displays for an error from the database system

Description

An error message from the database system consists of the following:

- ◆ **The following three-character code:** 750.
- ◆ **The following text string:** SQLERROR.
- ◆ **The three- or four-digit SQLCODE value.** In the following format sample, *nnnn* represents this.
- ◆ **The SQLCODE value's associated text message.** In the following format sample, *###...* represents this.

Format sample

In the following sample:

- ◆ *nnnn* is the SQLCODE value
- ◆ *###...* is the SQLCODE value's associated text message

Here is the format sample:

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

5

Dynamic SQL in MANTIS SQL Support

Who should read this chapter?

Read this chapter if you fall into one of the following categories:

- ◆ You are new to dynamic SQL programming.
- ◆ You are new to dynamic SQL programming in MANTIS SQL Support, but have experience with dynamic SQL programming in other languages (dynamic SQL in MANTIS SQL Support differs from dynamic SQL in other languages).

Dynamic SQL overview

Definition of dynamic SQL

Dynamic SQL is a method for executing SQL statements when a program needs the following information, but does not possess this information before the program executes:

- ◆ SQL statements
- ◆ Tables
- ◆ Column names

Example of when to use dynamic SQL

An application must use dynamic SQL if, during program execution, it requires a user to interactively enter an SQL statement at the terminal.

Principal statements for dynamically executing SQL statements

You can dynamically execute almost any statement that you would find in a static application. The principal statements enabling you to dynamically execute SQL statements are:

- ◆ PREPARE
- ◆ DESCRIBE
- ◆ EXECUTE
- ◆ EXECUTE IMMEDIATE.
- ◆ DECLARE (alternate form)
- ◆ OPEN (alternate form)
- ◆ FETCH (alternate form)

You accomplish communication to and from the database using the statements listed above and an SQLDA data structure.

SQLDA data structure

The SQLDA data structure used in database communication is a representation and repository of the data being transferred.

The SQLDA data structure used in database communication consists of:

- ◆ **Header elements.**
- ◆ **Repeating elements.** Each repeating element group is sometimes called an SQLVAR.

The SQLDA data structure contains metadata (for example, data length and data type) about the data passing between your program and the database.

Defining data about SQL statements and host variables

A single program can contain static SQL statements, dynamic SQL statements, or both:

- ◆ **Only static SQL statements.** The MANTIS program procedurally defines data about its SQL statements and host variables.
- ◆ **Only dynamic SQL statements.** The SQL preprocessor defines the data about the MANTIS program's SQL statements and host variables.
- ◆ **Both static and dynamic SQL statements.** The MANTIS program procedurally defines data about its static SQL statements and host variables, and the SQL preprocessor defines data about the MANTIS program's dynamic SQL statements and host variables.

SQL statements that you cannot execute dynamically

You cannot execute all DB2 statements dynamically.

Example: **SELECT** statement

An example of a statement that you cannot execute dynamically is the **SELECT** statement. Because MANTIS SQL Support executes all embedded SQL statements dynamically, MANTIS must overcome this limitation on the **SELECT** statement. It does this by implicitly performing **DECLARE** and **OPEN** statements for each static **SELECT** statement.

All statements that you cannot execute dynamically

You cannot execute the following SQL statements dynamically through the dynamic **PREPARE** and **EXECUTE** statements:

- ◆ **CLOSE**
- ◆ **CREATE VIEW**
- ◆ **DECLARE CURSOR**
- ◆ **DESCRIBE**
- ◆ **EXECUTE**
- ◆ **FETCH**
- ◆ **OPEN**
- ◆ **PREPARE**
- ◆ **RELEASE**
- ◆ **SELECT [INTO]**
- ◆ **WHENEVER**

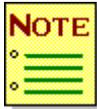
Auto-cursor FETCH statements

To support dynamic SQL, MANTIS uses auto-cursor FETCH statements. In the embedded SQL FETCH statement, the cursor-name is optional. An auto-cursor FETCH statement always applies to the most recently executed static SELECT statement (which may be in a calling program context).

Executing a statement dynamically

To execute a statement dynamically, you usually perform the following:

1. Use the PREPARE statement to prepare the SQL statement.
2. If you are retrieving, inserting, or updating data, use program logic to manipulate the SQLDA data structure. This SQLDA manipulation can include:
 - ◆ Allocating and expanding an SQLDA data structure.
 - ◆ Using the DESCRIBE statement to retrieve metadata from SQL.
 - ◆ Causing data transfer between SQL and MANTIS variables.



For an illustration of how dynamically executed SQL statements and the SQLDA data structure work together in dynamic routines, see “[Dynamic SQL sample](#)” on page 84.

3. Use the EXECUTE statement to execute the SQL statement.

Dynamic SQL sample code for creating an SQLDA data structure

Introduction

The following sample code provides an example of how to create an SQLDA data structure. This code includes examples of the built-in SQLDA statement and SQLDA function (described in “[SQLDA data structure](#)” on page 87).



For details on the syntax for the PREPARE, DESCRIBE, and EXECUTE statements used in this sample code, refer to the *SQL Reference DB2* guide.

Sample code

Below is sample code for creating an SQLDA data structure:

```

10 TEXT UPDATE_STMT (250)
15 TEXT DA: DA="sqlda-name"
16 SQLDA(DA)=NEW
20 EXEC_SQL: | PREPARE stmt-name FROM :UPDATE_STMT
30 END
40 EXEC_SQL: | DESCRIBE stmt-name INTO sqlda-name
50 END
60 FOR I = 1 TO SQLDA(DA,"SQLD")
70 SQLDA(DA,"SQLHOSTVAR",I)=input-parameter
80 END
90 EXEC_SQL: | EXECUTE stmt-name USING DESCRIPTOR sqlda-name
100 END

```

Explanation of the sample code

In the preceding sample code, the following occur:

1. The first SQLDA statement allocates the SQLDA data structure.
2. The PREPARE statement dynamically compiles the SQL statement.
3. The DESCRIBE statement returns metadata about the results of the SQL statement in the SQLDA data structure.
4. *(This statement is optional)* The second SQLDA statement supplies values for input host variables.
5. The EXECUTE statement tells the processor to execute the named statement.



At the end of the program, you may include the SQLDA built-in function to transfer data from a host variable.

Additional sample code

To find dynamic and static sample code for the following, see “[MANTIS SQL sample programs](#)” on page 109:

- ◆ INSERT
- ◆ UPDATE
- ◆ SELECT
- ◆ DELETE
- ◆ A dynamic QUERY-like routine
- ◆ A dynamic column-select routine

SQLDA data structure

In dynamic SQL, SQL communicates with your program via an SQLDA (SQL Descriptor Area) data structure. An SQLDA data structure holds “information about data” (known as metadata) that is transferred between your program and the database.

Figure representing SQLDA data structure

The following figure represents the structure of an SQLDA data structure. It contains the following elements:

- ◆ **Header elements.** These are the first four elements in the figure below. They occur once per SQLDA data structure.
- ◆ **Repeating elements.** These are the next six elements in the figure below. They repeat once per data item.



For more information on header elements and repeating elements, see “[SQLDA header elements](#)” on page 96 and “[SQLDA repeating elements](#)” on page 105.

SQLDAID	SQLDABC	SQLDN	SQLD	
	SQLNAME ₁		SQLTYPE ₁	Header Elements
	SQLLEN ₁		SQLFRAC ₁	
	SQLIND ₁		SQLDATA ₁	Repeating Element 1
	SQLNAME ₂		SQLTYPE ₂	
	SQLLEN ₂		SQLFRAC ₂	
	SQLIND ₂		SQLDATA ₂	Repeating Elements 2
		:		
		:		

Data item

A data item is one of the following:

- ◆ **One column of an SQL table.** This is the output from SQL to your program.
- ◆ **The value of a host variable.** This is the input to SQL from your program.

The maximum number of entries in a data item is 300.

SQLDA names

SQLDA names must follow the rules for MANTIS variable names, so that the MANTIS parser can recognize them in embedded SQL statements.

Declaring SQLDA elements in other programming languages vs. declaring them in MANTIS SQL Support

MANTIS SQL Support differs from other programming languages in SQLDA element declaration and access:

- ◆ **Other programming languages.** To declare and access SQLDA elements, you must perform the following:
 1. Explicitly declare each SQLDA element as a data area in your program.
 2. Access the SQLDA elements through programming statements.
- ◆ **MANTIS SQL Support.** When you declare an SQLDA data structure, MANTIS SQL Support automatically builds an SQLDA data structure with all the elements shown in the preceding figure (see [“Figure representing SQLDA data structure”](#) on page 87).

Number of repeating elements in an SQLDA data structure

An SQLDA data structure contains the default number of repeating elements (your Master User set this in your `SQLVARINC` MANTIS Options). Your program can modify this value.

Built-in SQLDA statement and built-in SQLDA function

The built-in SQLDA statement and built-in SQLDA function enable your MANTIS program to create and maintain SQLDA data structures. You can then use programmed dynamic SQL statements with these SQLDA data structures.

Use the SQLDA statement and function for different purposes:

- ◆ **SQLDA statement.** Your MANTIS program can use the SQLDA statement to perform the following:

- Create a named SQLDA data structure.
- Send information (input host variable information) to SQL.

In the SQLDA statement, the *sqlda-name* parameter must include a text expression or literal that contains the name of a valid MANTIS variable.

- ◆ **SQLDA function.** Your program can use the SQLDA function to retrieve information (output host variable information) about SQL table columns.

Sample code for the SLQDA statement and SQLDA function

Introduction

Following sections. The following sections provide format samples for various uses of the SQLDA statement and SQLDA function:

- ◆ Using the SLQDA statement:
 - “Allocating an SQLDA data structure” on page 91
 - “Deallocating an SQLDA data structure” on page 92
 - “Moving data from your program into an SQLDA header element” on page 93
 - “Moving data from your program into an SQLDA repeating element” on page 97
- ◆ Using the SQLDA function:
 - “Moving data from an SQLDA header element into your program” on page 102
 - “Moving data from an SQLDA repeating element into your program” on page 103

Items to replace in the syntax. In the syntax described by the following sections, you must replace the following with standard MANTIS variables, literals, or expressions:

- ◆ *sqlda-name*
- ◆ *header-element*



For a list of header elements, see “SQLDA header elements” on page 96.

- ◆ *repeating-element*



For a list of repeating elements, see “SQLDA repeating elements” on page 105.

- ◆ *index* (refers to the sequential occurrence of the repeating element group in the SQLDA data structure)

Allocating an SQLDA data structure

Description. Use this SQLDA statement to allocate a new SQLDA data structure.

Syntax. See the following syntax:

SQLDA(*sqlda-name*) = NEW

Syntax element. See the following syntax element:

sqlda-name

Description *Required.* Specifies the name of the new SQLDA data structure.

Format MANTIS text expression, 1–18 characters in length.

Consideration The expression result must be a valid MANTIS symbolic name with a length of 1–18 characters.

General considerations

Consider the following:

- ◆ This statement allocates a new, empty SQLDA data structure with the default number of repeating elements. (At installation, your Master User set the default number of repeating elements as one of your MANTIS Options.)
- ◆ Within your program, you can modify an SQLDA data structure's size by resetting the value of SQLMAX (see the [SQLMAX discussion](#)).
- ◆ If you use an SQLDA statement to declare a second SQLDA data structure with the same name as a first, pre-existing SQLDA data structure, MANTIS ignores your SQLDA statement.
- ◆ The scope of an SQLDA data structure is the current DO level. You can create two SQLDA data structures, both with the same name, on two different DO levels. However, within a single DO level, you can only access SQLDA data structures defined for that DO level.

Example

See the following sample code:

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

Deallocating an SQLDA data structure

Description. Use this SQLDA statement to deallocate an SQLDA data structure.

Syntax. See the following syntax:

SQLDA(*sqlda-name*) = QUIT

Syntax element. See the following syntax element:

sqlda-name

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

Format MANTIS text expression.

Consideration The text expression must specify the name of a previously-allocated SQLDA data structure.

General considerations

Consider the following:

- ◆ This statement deallocates an existing SQLDA data structure by name.
- ◆ When a DO level is exited, MANTIS automatically deallocates all SQLDA data structures that were defined at that DO level.
- ◆ If you execute a RUN without specifying a line number, MANTIS automatically deallocates all SQLDA data structures.



A RUN with a line number may produce unpredictable results if you have modified the program in which you are running the specific line.

Example

See the following sample code:

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

Moving data from your program into an SQLDA header element

Use this SQLDA statement to set header or column-name information in the SQLDA data structure. See the following syntax:

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

sqlda-name

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

header-element

- Description** *Required.* Specifies the name of the SQLDA header element into which you are moving data.
- Format** MANTIS text expression.
- Options** You may set only the following three header elements:
- ◆ **SQLN.** (In DB2 support, SQLN and SQLMAX are the same. See SQLMAX in the following bullet.)


```
SQLDA("DA1", "SQLN") = 10
```
 - ◆ **SQLMAX.** The number of repeating groups in the physical SQLDA data structure. This value can range from 1–300. Setting this number in your program causes the SQLDA data structure to expand or contract by the specified number of repetitions. Once physically expanded, the space occupied by the SQLDA data structure will never physically contract. For example, if an SQLDA data structure named DA1 has 20 repeating occurrences, the following statement will reduce the logical occurrences to five; however, physical space for 20 remains (these numbers are arbitrary).


```
SQLDA("DA1", "SQLMAX") = 5
```
 - ◆ **SQLD.** The number of repeating groups currently in use. SQL sets it as the result of a PREPARE INTO or DESCRIBE statement. However, when necessary, the program can set it.


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

Considerations

Consider the following:

- ◆ For more information on possible *header-element* values, see the table under [SQLDA header elements](#).
- ◆ The other header element illustrated in “[SQLDA data structure](#)” on page 87, SQLDAID, is read-only. Attempting to use a read-only element in this SQLDA statement generates a fault.

expression

Description *Required.* Specifies the SQLDA variable count.

Format MANTIS numeric expression.

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

General consideration

Consider the following:

- ◆ A third-generation language like FORTRAN or COBOL handles the execution of a DESCRIBE statement differently than MANTIS SQL Support:
 - Third-generation language—If the SQLDA is too small (that is, if SQLN is less than the number of items that will be returned as a result of the PREPARE INTO or DESCRIBE statement), SQL sets SQLN to the required number and terminates. The program must then expand the SQLDA data structure to the required size.
 - MANTIS SQL Support—If the SQLDA data structure is too small to accept the results of a DESCRIBE statement, MANTIS SQL Support automatically expands the SQLDA data structure to the required size. To check the number of repeating elements after MANTIS executes the DESCRIBE statement, examine the SQLD value.

Example

Consider the following sample code:

```
SQLDA("SQLDA1", "SQLN") = TOTAL_NEEDED
```

SQLDA header elements. The following table lists the SQLDA header elements. Use this table to complete *header-element* in the SQLDA statement under “Moving data from your program into an SQLDA header element” on page 93.

Element	How it is set	Results	Updateable?
SQLN/SQLMAX. Total number of host variables in the SQLDA data structure.	Can be set in the following ways: <ul style="list-style-type: none"> ◆ Automatically set when MANTIS allocates SQLDA, using a value from the MANTIS Options. ◆ Manually set by the SQLDA statement. 	Number of repeating groups allocated.	Yes.
SQLD. Current number of columns described in the SQLDA data structure	Can be set in the following ways: <ul style="list-style-type: none"> ◆ Automatically set as a result of a PREPARE INTO or DESCRIBE statement. ◆ Manually set by the SQLDA statement. 	Number of input or output variables described in SQLDA.	Yes.
SQLDAID.	Set by DB2.	Normally contains “SQLDA”.	No.

Moving data from your program into an SQLDA repeating element

Use this SQLDA statement to supply values for input host variables, setting the values of repeating elements. See the following syntax:

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.

Format Mantis text expression.

Options You may set the following three repeating elements:

- ◆ **SQLNAME.** Specifies the column name returned by SQL (your program can also set the column name). SQLNAME has a type of TEXT and a length of 18. Although you can modify the SQLNAME element, your modifications do not affect the database. In addition, the database writes to the SQLNAME element, so the SQLNAME element's contents may be destroyed each time MANTIS executes an EXEC_SQL statement.

- ◆ **SQLIND.** Specifies the indicator value. The indicator value indicates what the host variable contains:
 - A real value
 - NULL
 - MISSING



For more information on indicator variables, see “[Indicator variables](#)” on page 46

Possible indicator values and their meanings are:

- Less than zero—The host variable data is NULL or MISSING.
 - Greater than or equal to zero—The host variable contains real values.
- ◆ **SQLDATA.** Has different behavior in third-generation languages and in MANTIS SQL Support:
 - Third-generation languages—The SQLDATA element holds a four-byte binary address that a program and DB2 use to access the data item being transferred between the program and DB2. A third-generation program must acquire space for the data item and place the space’s address in this element.
 - MANTIS SQL Support—The MANTIS nucleus uses the SQLDATA element to automatically perform the following actions when you transfer data into the SQLDA data structure:
 1. Allocate a data area for the data item, if necessary.
 2. Expand the data area, if necessary.
 3. Set the value of SQLDATA to the address of the data area.

MANTIS SQL Support uses this address internally, and your program does not need to manipulate this value.
 4. Move data from the MANTIS host variable into the SQLDA data area.
 5. Set SQLTYPE and SQLLEN to match the definition of the MANTIS variable, according to the SQLTYPE values in “[MANTIS SQL Support data type conversion](#)” on page 101. The MANTIS program sets SQLLEN to the length of the MANTIS variable.

Considerations

Consider the following:

- ◆ If you are transferring data out of the SQLDA, the SQLDATA element simply performs the transfer.
- ◆ The SQLDATA element may have a type of numeric or string. When you do not know, in advance, the type of data you want to retrieve from the database, use the SQLTYPE elements to determine the data type.

index

Description *Required.* Specifies the sequential occurrence of the repeating element into which you are moving data. The occurrence is relative to one rather than zero; for example, the eighth occurrence of the repeating element has an index value of eight.

Format Mantis numeric expression.

expression

Description *Required.* Specifies an SQLVAR element value.

Format MANTIS text or numeric expression.

Consideration Consider the following:

- ◆ The expression may be either text or numeric.
- ◆ There are no limitations on the expression.

General considerations

Consider the following:

- ◆ MANTIS sets:
 - SQLLEN to the length of the MANTIS expression.
 - SQLTYPE to the equivalent data type in MANTIS (a data type conversion table appears below).
- ◆ For compatibility with MANTIS support for SUPRA SQL, MANTIS recognizes the SUPRA SQL element names as equivalent to the DB2 element names. Use the DB2 names as follows:

DB2 SQL	SUPRA SQL
SQLNAME	SQLCOLNAME
SQLIND	SQLHOSTIND
SQLDATA	SQLHOSTVAR
SQLLEN	SQLCOLLENGTH
SQLTYPE	SQLCOLTYPE/SQLHOSTVARTY

- ◆ MANTIS SQL Support uses the SQLTYPE element to perform the following:
 - Describe the data type in the DB2 database.
 - Receive the host variable data type from MANTIS.

Whenever you use the SQLDA statement to set SQLDATA, MANTIS sets:

- SQLLEN to the length of the MANTIS expression
- SQLTYPE to the MANTIS data type

At this point, previous contents of the elements, which a previous DESCRIBE statement may have set, may be lost.

Example

See the following sample code:

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

MANTIS SQL Support data type conversion. You can use the SQLDA statement to assign either string or numeric MANTIS data to an SQLDA data structure's repeating element. MANTIS sets the SQLTYPE to one of the following:

- ◆ 1
- ◆ 2
- ◆ 12

If you do not use the SQLDA statement to assign string or numeric MANTIS data to an SQLDA data structure's repeating element, MANTIS assigns SQLTYPEs as shown in the following table, replacing the existing values of SQLTYPE.

SQL data type	Description	SQL type (documented in DB2 SQL Reference manual)	SQL type set by MANTIS	MANTIS type
DATE	Calendar Date	384/385	448/449	TEXT
TIME	Time	388/389	448/449	TEXT
TIMESTAMP	Timestamp	392/393	448/449	TEXT
VARCHAR	Variable String	448/449	448/449	TEXT
CHAR	Fixed length string	452/453	448/449	TEXT
FLOAT	Floating point number	480/481	480/481	BIG
DECIMAL	Packed decimal number	484/485	480/481	BIG
INTEGER	Long integer	496/497	480/481	BIG
SMALLINT	Short integer	500/501	480/481	BIG

Moving data from an SQLDA header element into your program

Use this SQLDA function to read header elements. See the following syntax:

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

mantis-variable

- | | |
|--------------------|--|
| Description | <i>Required.</i> Specifies the name into which the SQLDA header element is to be placed. |
| Format | MANTIS variable or array occurrence. |

sqlda-name

- | | |
|--------------------|---|
| Description | <i>Required.</i> Specifies the name of a previously-allocated SQLDA data structure. |
| Format | MANTIS text expression. |

header-element

- | | |
|--------------------|--|
| Description | <i>Required.</i> Specifies the name of the header element you are reading. |
| Format | MANTIS text expression. |

General considerations

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

Example

Consider the following sample code:

```
TOTAL_NEEDED = SQLDA("SQLDA1", "SQLN")
```

Moving data from an SQLDA repeating element into your program

Use this SQLDA function to transfer data from a repeating element into a MANTIS variable in your program. See the following syntax:

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

mantis-variable

Description	<i>Required.</i> Specifies the destination into which the SQLDA repeating element is to be placed.
Format	MANTIS variable or array element reference.

sqlda-name

Description	<i>Required.</i> Specifies the name of a previously-allocated SQLDA.
Format	MANTIS text expression.

repeating-element

Description	<i>Required.</i> Specifies the name of the repeating element to be read.
Format	MANTIS text expression.
Consideration	To fill out this element, see the table under SQLDA repeating elements .

index

Description *Required.* Specifies the sequential occurrence of the repeating element to be read. The occurrence is relative to one, rather than zero; for example, the eighth occurrence of the repeating element has an index value of eight.

Format MANTIS numeric expression.

General considerations

Consider the following:

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

Example

Consider the following sample code:

```
EMPLOYEE_NUMBER = SQLDA( "SQLDA1", "SQLHOSTVAR", 1)
```

SQLDA repeating elements. The following table lists the SQLDA repeating elements. Use this table to complete *repeating-element* in the SQLDA statement under “[Moving data from an SQLDA repeating element into your program](#)” on page 103.

Element	How it is set	Results	Updateable?
SQLNAME / SQLCOLNAME. SQL column name.	Can be set in the following ways: <ul style="list-style-type: none"> ◆ By DB2, as the result of a PREPARE INTO statement. ◆ By DB2, as the result of a DESCRIBE statement. 	Column name.	Yes
SQLTYPE / SQLCOLTYPE. One of the following: <ul style="list-style-type: none"> ◆ Data type in the database. ◆ Host variable data type. 	Can be set in the following ways: <ul style="list-style-type: none"> ◆ By DB2, as the result of a PREPARE INTO statement. ◆ By DB2, as the result of a DESCRIBE statement. ◆ By MANTIS, prior to an EXECUTE statement. ◆ By MANTIS, prior to an OPEN statement. ◆ By MANTIS, prior to a FETCH statement. ◆ By the MANTIS SQLDA statement. 	See “ MANTIS SQL Support data type conversion ” on page 101.	No

Element	How it is set	Results	Updateable?
<p>SQLLEN / SQLCOLLENGTH. One of the following:</p> <ul style="list-style-type: none"> ◆ Maximum number of bytes for a column in the database. ◆ Actual number of host variable bytes. 	<p>Can be set in the following ways:</p> <ul style="list-style-type: none"> ◆ By DB2, as the result of a PREPARE INTO statement. ◆ By DB2, as the result of a DESCRIBE statement. ◆ By MANTIS, prior to an EXECUTE statement. ◆ By MANTIS, prior to an OPEN statement. ◆ By MANTIS, prior to a FETCH statement. ◆ By the MANTIS SQLDA statement. 	<p>Results in one of the following:</p> <ul style="list-style-type: none"> ◆ 4 for numeric columns. ◆ 8 for numeric columns. ◆ Maximum. ◆ Actual string length. 	<p>No</p>
<p>SQLFRAC / SQLCOLFRAC. Number of decimal places for DECIMAL and FLOAT.</p>	<p>Can be set in the following ways:</p> <ul style="list-style-type: none"> ◆ By DB2, as the result of a PREPARE INTO statement. ◆ By DB2, as the result of a DESCRIBE statement. 	<p>Number of places specified in the CREATE TABLE statement</p>	<p>No</p>

Element	How it is set	Results	Updateable?
SQLIND / SQLHOSTIND. The value of the NULL or MISSING indicator variable.	Can be set in the following ways: <ul style="list-style-type: none"> ◆ By DB2, as the result of a FETCH statement. ◆ By the MANTIS SQLDA statement. 	Results in one of the following: <ul style="list-style-type: none"> ◆ A value that is less than or equal to -1. ◆ NULL.. ◆ A value that is defined and addressed by SQLDATA. 	Yes
SQLDATA / SQLHOSTVAR. Address host variable data.	Can be set in the following ways: <ul style="list-style-type: none"> ◆ By MANTIS, prior to an EXECUTE statement. ◆ By MANTIS, prior to an OPEN statement. ◆ By MANTIS, prior to a FETCH statement. ◆ By the MANTIS SQLDA statement. 	Results in one of the following: <ul style="list-style-type: none"> ◆ Address of MANTIS variable in the MANTIS data work area. ◆ Address of MANTIS data after being copied or converted into work buffer. 	Yes

Cursors for prepared statements

For MANTIS Dynamic SQL Support, cursor names are not known at the time of the PREPARE. (This is because MANTIS executes interpretively). MANTIS resolves this problem by declaring cursors for all prepared statements that could possibly require them. These statements include:

- ◆ DECLARE
- ◆ Static SELECT
- ◆ Statements that are parameters to PREPARE

A

MANTIS SQL sample programs

Introduction

Regarding the sample programs in this appendix, consider the following:

- ◆ These programs do not contain
 - Error-checking logic
 - Display logic
- ◆ These programs contain hard-coded employee information.
- ◆ Each static program has the same functionality as the dynamic program that follows it.

INSERT routines

Static INSERT routine

Consider the following sample code for a static INSERT routine:

```
10 ENTRY STATIC_INSERT
20 .|
30 .| THIS PROGRAM IS AN EXAMPLE OF A STATIC SQL "INSERT"
40 .| STATEMENT. IT INSERTS ONE EMPLOYEE INTO AN EMPLOYEE
50 .| TABLE.
60 .|
70 .BIG HIRE_DATE, BIRTH_DATE, JOB_CODE,SALARY, EDUCATION_LEVEL
80 .TEXT EMPLOYEE_NUMBER(6),FIRST_NAME(20)
85 .'MIDDLE_INITIAL(1),LAST_NAME(20)
90 .TEXT PHONE_NUMBER(4),WORK_DEPARTMENT(3),SEX(1)
100 .|
110 .EMPLOYEE_NUMBER = "000120"
120 .FIRST_NAME="SEAN"
130 .MIDDLE_INITIAL= " "
140 .LAST_NAME = "O'CONNELL"
150 .BIRTH_DATE=421018
160 .HIRE_DATE=631205
170 .JOB_CODE=58
180 .EDUCATION_LEVEL=14
```

```
190 .SALARY=29250
200 .PHONE_NUMBER="2167"
210 .WORK_DEPARTMENT="A00"
220 .SEX="M"
230 .|
240 .EXEC_SQL: | INSERT INTO FRED.TEMPL
250 ..|                (EMPNO,
260 ..|                FIRSTNME,
270 ..|                MIDINIT,
280 ..|                LASTNAME,
290 ..|                BRTHDATE,
300 ..|                HIREDATE,
310 ..|                JOBCODE,
320 ..|                EDUCLVL,
330 ..|                SALARY,
340 ..|                PHONENO,
350 ..|                WORKDEPT,
360 ..|                SEX)
370 ..|                VALUES (:EMPLOYEE_NUMBER,
380 ..|                :FIRST_NAME,
390 ..|                :MIDDLE_INITIAL,
400 ..|                :LAST_NAME,
410 ..|                :BIRTH_DATE,
420 ..|                :HIRE_DATE,
430 ..|                :JOB_CODE,
440 ..|                :EDUCATION_LEVEL,
450 ..|                :SALARY,
460 ..|                :PHONE_NUMBER,
470 ..|                :WORK_DEPARTMENT,
480 ..|                :SEX)
490 .END
500 EXIT
```

Dynamic INSERT routine

Consider the following sample code for a dynamic INSERT routine:

```

10 ENTRY DYNAMIC_INSERT
20 .|
30 .| THIS PROGRAM IS AN EXAMPLE OF A DYNAMIC SQL "INSERT"
40 .| STATEMENT. IT INSERTS ONE EMPLOYEE INTO AN EMPLOYEE
50 .| TABLE.
60 .|
70 .BIG HIRE_DATE,BIRTH_DATE,JOB_CODE,SALARY
75 .BIG EDUCATION_LEVEL
80 .TEXT EMPLOYEE_NUMBER(6),FIRST_NAME(20)
85 .TEXT MIDDLE_INITIAL(1),LAST_NAME(20)
90 .TEXT PHONE_NUMBER(4),WORK_DEPARTMENT(3),SEX(1)
100 .TEXT SQL_TEXT(254)
110 .|
120 .EMPLOYEE_NUMBER="000120"
130 .FIRST_NAME="SEAN"
140 .MIDDLE_INITIAL=" "
150 .LAST_NAME="O'CONNELL"
160 .BIRTH_DATE=421018
170 .HIRE_DATE=631205
180 .JOB_CODE=58
190 .EDUCATION_LEVEL=14
200 .SALARY=29250
210 .PHONE_NUMBER="2167"
220 .WORK_DEPARTMENT="A00"
230 .SEX="M"
240 .|
250 .SQL_TEXT="INSERT INTO FRED.TEMPL"
260 .'"(EMPNO, FIRSTNME, MIDINIT, LASTNAME, BRTHDATE",
270 .'"HIREDATE, JOBCODE, EDUCLVL, SALARY, PHONENO",
280 .'"WORKDEPT, SEX)"
290 .'"VALUES (?,?,?,?,?, ?, ?, ?, ?, ?, ?)"

```

```
300 .|
310 .EXEC_SQL:| PREPARE S1 FROM :SQL_TEXT
320 .END
330 .|
340 .SQLDA("SQLDA1") = NEW
350 .SQLDA("SQLDA1", "SQLMAX")=12
360 .EXEC_SQL:| DESCRIBE S1 INTO SQLDA1
370 .END
380 .SQLDA("SQLDA1", "SQLDATA", 1)=EMPLOYEE_NUMBER
390 .SQLDA("SQLDA1", "SQLDATA", 2)=FIRST_NAME
400 .SQLDA("SQLDA1", "SQLDATA", 3)=MIDDLE_INITIAL
410 .SQLDA("SQLDA1", "SQLDATA", 4)=LAST_NAME
420 .SQLDA("SQLDA1", "SQLDATA", 5)=BIRTH_DATE
430 .SQLDA("SQLDA1", "SQLDATA", 6)=HIRE_DATE
440 .SQLDA("SQLDA1", "SQLDATA", 7)=JOB_CODE
450 .SQLDA("SQLDA1", "SQLDATA", 8)=EDUCATION_LEVEL
460 .SQLDA("SQLDA1", "SQLDATA", 9)=SALARY
470 .SQLDA("SQLDA1", "SQLDATA", 10)=PHONE_NUMBER
480 .SQLDA("SQLDA1", "SQLDATA", 11)=WORK_DEPARTMENT
490 .SQLDA("SQLDA1", "SQLDATA", 12)=SEX
500 .|
510 .EXEC_SQL:| EXECUTE S1 USING DESCRIPTOR SQLDA1
520 .END
530 EXIT
```

UPDATE routines

Static UPDATE routine

Consider the following sample code for a static UPDATE routine:

```
10 ENTRY STATIC_UPDATE
20 .|
30 .| THIS PROGRAM IS AN EXAMPLE OF A STATIC SQL "UPDATE"
40 .| STATEMENT. IT UPDATES ONE EMPLOYEE FROM AN EMPLOYEE
50 .| TABLE.
60 .|
70 .BIG HIRE_DATE,BIRTH_DATE
80 .TEXT EMPLOYEE_NUMBER(6)
90 .TEXT FIRST_NAME(20),MIDDLE_INITIAL(1),LAST_NAME(20)
100 .|
110 .EMPLOYEE_NUMBER="000120"
120 .FIRST_NAME="JOHN"
130 .MIDDLE_INITIAL="H"
140 .LAST_NAME="DOE"
150 .BIRTH_DATE=490113
160 .HIRE_DATE=880120
170 .|
180 .EXEC_SQL
190 ..|
200 ..| UPDATE FRED.TEMPL
210 ..|
220 ..|     SET  FIRSTNME = :FIRST_NAME,
230 ..|         MIDINIT  = :MIDDLE_INITIAL,
240 ..|         LASTNAME  = :LAST_NAME,
250 ..|         BRTHDATE  = :BIRTH_DATE,
260 ..|         HIREDATE  = :HIRE_DATE
270 ..|
280 ..|     WHERE EMPNO = :EMPLOYEE_NUMBER
290 .END
300 EXIT
```

Dynamic UPDATE routine

Consider the following sample code for a dynamic UPDATE routine:

```

10 ENTRY DYNAMIC_UPDATE
20 .|
30 .| THIS PROGRAM IS AN EXAMPLE OF A DYNAMIC SQL "UPDATE"
40 .| STATEMENT. IT UPDATES ONE EMPLOYEE FROM AN EMPLOYEE
50 .| TABLE.
60 .|
70 .BIG HIRE_DATE,BIRTH_DATE
80 .TEXT EMPLOYEE_NUMBER(6),FIRST_NAME(20),
85 .'MIDDLE_INITIAL(1),LAST_NAME(20)
90 .TEXT DA(18),DAPARM(8)
100 .TEXT SQL_TEXT(254)
110 .|
120 .EMPLOYEE_NUMBER="000120"
130 .FIRST_NAME="JOHN"
140 .MIDDLE_INITIAL="H"
150 .LAST_NAME="DOE"
160 .BIRTH_DATE=490113
170 .HIRE_DATE=880120
180 .|
190 .SQL_TEXT="UPDATE FRED.TEMPL SET"
200 .'"FIRSTNME = ?, MIDINIT = ?, LASTNAME = ?,"
210 .'"BRTHDATE = ?, HIREDATE = ?"
220 .'"WHERE EMPNO = ?"
230 .|
240 .EXEC_SQL:| PREPARE S1 FROM :SQL_TEXT
250 .END
260 .|

```

```
270 .SQLDA( "SQLDA1" )=NEW
280 .DA="SQLDA1 "
290 .DAPARM="SQLDATA"
300 .SQLDA(DA, "SQLD" )=6
310 .EXEC_SQL: | DESCRIBE S1 INTO SQLDA1
320 .END
330 .SQLDA(DA, DAPARM, 1)=FIRST_NAME
340 .SQLDA(DA, DAPARM, 2)=MIDDLE_INITIAL
350 .SQLDA(DA, DAPARM, 3)=LAST_NAME
360 .SQLDA(DA, DAPARM, 4)=BIRTH_DATE
370 .SQLDA(DA, DAPARM, 5)=HIRE_DATE
380 .SQLDA(DA, DAPARM, 6)=EMPLOYEE_NUMBER
390 . |
400 .EXEC_SQL: | EXECUTE S1 USING DESCRIPTOR SQLDA1
410 .END
420 EXIT
```

SELECT routines

Static SELECT routine

Consider the following sample code for a static SELECT routine:

```

10 ENTRY STATIC_SELECT
20 .|
30 .| THIS PROGRAM IS AN EXAMPLE OF A STATIC SQL "SELECT"
40 .| STATEMENT. IT RETRIEVES EMPLOYEE INFORMATION FOR ONE
50 .| EMPLOYEE FROM AN EMPLOYEE TABLE.
60 .|
70 .BIG HIRE_DATE,BIRTH_DATE,JOB_CODE,SALARY
75 .BIG EDUCATION_LEVEL
80 .TEXT EMPLOYEE_NUMBER(6),FIRST_NAME(20)
85 .TEXT MIDDLE_INITIAL(1),LAST_NAME(20)
90 .TEXT WORK_DEPARTMENT(3),PHONE_NUMBER(3),SEX(1)
100 .EMPLOYEE_NUMBER="000120"
110 .|
120 .EXEC_SQL:| DECLARE C1 CURSOR FOR
130 ..|           SELECT * FROM FRED.TEMPL
140 ..|           WHERE EMPNO = :EMPLOYEE_NUMBER
150 .END
160 .EXEC_SQL:| OPEN C1
170 .END
180 .EXEC_SQL:| FETCH C1 INTO :EMPLOYEE_NUMBER,
190 ..|           :FIRST_NAME,
200 ..|           :MIDDLE_INITIAL,
210 ..|           :LAST_NAME,
220 ..|           :WORK_DEPARTMENT,
230 ..|           :PHONE_NUMBER,
240 ..|           :HIRE_DATE,
250 ..|           :JOB_CODE,
260 ..|           :EDUCATION_LEVEL,
270 ..|           :SEX,
280 ..|           :BIRTH_DATE,
290 ..|           :SALARY
300 .END
310 .EXEC_SQL:| CLOSE C1
320 .END
330 EXIT

```

Dynamic SELECT routine

Consider the following sample code for a dynamic SELECT routine:

```
10 ENTRY DYNAMIC_SELECT
20 .|
30 .| THIS PROGRAM IS AN EXAMPLE OF A DYNAMIC SQL "SELECT"
40 .| STATEMENT. IT RETRIEVES EMPLOYEE INFORMATION FOR ONE
50 .| EMPLOYEE FROM AN EMPLOYEE TABLE.
60 .|
70 .BIG HIRE_DATE,BIRTH_DATE,JOB_CODE,SALARY
75 .BIG EDUCATION_LEVEL
80 .TEXT EMPLOYEE_NUMBER(6)
90 .TEXT FIRST_NAME(20),MIDDLE_INITIAL(1),LAST_NAME(20)
100 .TEXT WORK_DEPARTMENT(3),PHONE_NUMBER(3),SEX(1)
110 .TEXT SQL_TEXT(254)
120 .|
130 .EMPLOYEE_NUMBER="000120"
140 .SQL_TEXT="SELECT * FROM FRED.TEMPL"
150 . "WHERE EMPNO = ?"
160 .|
170 .EXEC_SQL:| PREPARE S1 FROM :SQL_TEXT
180 .END
190 .EXEC_SQL:| DECLARE C1 CURSOR FOR S1
200 .END
210 .EXEC_SQL:| OPEN C1 USING :EMPLOYEE_NUMBER
220 .END
230 .SQL_TEXT="FETCH C1 USING DESCRIPTOR"
240 .EXEC_SQL:| PREPARE S2 FROM :SQL_TEXT
250 .END
260 .SQLDA("SQLDA1")=NEW
270 .EXEC_SQL:| DESCRIBE S2 INTO SQLDA1
280 .END
290 .EXEC_SQL:| EXECUTE S2 USING DESCRIPTOR SQLDA1
```

```
300 .END
310 .EXEC_SQL: | CLOSE C1
320 .END
330 . |
340 .EMPLOYEE_NUMBER=SQLDA("SQLDA1", "SQLDATA", 1)
350 .FIRST_NAME=SQLDA("SQLDA1", "SQLDATA", 2)
360 .MIDDLE_INITIAL=SQLDA("SQLDA1", "SQLDATA", 3)
370 .LAST_NAME=SQLDA("SQLDA1", "SQLDATA", 4)
380 .WORK_DEPARTMENT=SQLDA("SQLDA1", "SQLDATA", 5)
390 .PHONE_NUMBER=SQLDA("SQLDA1", "SQLDATA", 6)
400 .HIRE_DATE=SQLDA("SQLDA1", "SQLDATA", 7)
410 .JOB_CODE=SQLDA("SQLDA1", "SQLDATA", 8)
420 .EDUCATION_LEVEL=SQLDA("SQLDA1", "SQLDATA", 9)
430 .SEX=SQLDA("SQLDA1", "SQLDATA", 10)
440 .BIRTH_DATE=SQLDA("SQLDA1", "SQLDATA", 11)
450 .SALARY=SQLDA("SQLDA1", "SQLDATA", 12)
460 EXIT
```

DELETE routines

Static DELETE routine

Consider the following sample code for a static DELETE routine:

```
10 ENTRY STATIC_DELETE
20 .|
30 .| THIS PROGRAM IS AN EXAMPLE OF A STATIC SQL "DELETE"
40 .| STATEMENT. IT DELETES ONE EMPLOYEE FROM AN EMPLOYEE
50 .| TABLE.
60 .|
70 .TEXT EMPLOYEE_NUMBER(6)
80 .EMPLOYEE_NUMBER "000120"
90 .EXEC_SQL
100 ..|
110 ..| DELETE FROM FRED.TEMPL
120 ..|
130 ..| WHERE EMPNO = :EMPLOYEE_NUMBER
140 .END
150 EXIT
```

Dynamic DELETE routine

Consider the following sample code for a dynamic DELETE routine:



Using an SQLDA is unnecessary because no data is transferred between the database system and the MANTIS program.

```
10 ENTRY DYNAMIC_DELETE
20 .|
30 .| THIS PROGRAM IS AN EXAMPLE OF A DYNAMIC SQL "DELETE"
40 .| STATEMENT. IT DELETES ONE EMPLOYEE FROM AN EMPLOYEE
50 .| TABLE.
60 .|
70 .TEXT EMPLOYEE_NUMBER(6),SQL_TEXT(254)
80 .EMPLOYEE_NUMBER "000120"
90 .SQL_TEXT="DELETE FROM FRED.TEMPL WHERE EMPNO = ?"
100 .|
110 .EXEC_SQL
120 ..|
130 ..| PREPARE S1 FROM :SQL_TEXT
140 ..|
150 .END
160 .EXEC_SQL
170 ..|
180 ..| EXECUTE S1 USING :EMPLOYEE_NUMBER
190 ..|
200 .END
```

Dynamic QUERY-like function

This program enables you to perform the following:

1. Interactively execute SQL statements.
2. At your terminal, display the column data resulting from the execution of those statements.

See the following sample code for a dynamic QUERY-like function:

```
10 ENTRY DB2_QUERY
20 CLEAR
30 HEAD "DYNAMIC EXAMPLE"
40 |
50 | This example illustrates the use of dynamic SQL to
60 | perform a QUERY-like function.
70 |
80 TEXT STMT(80),TMP(6),DA
90 PROGRAM DB2_SHOW_TABLE("EXAMPLES:DB2_SHOW_TABLE","CASINO")
100 DA="DA"
110 SQLDA(DA)=NEW
120 EXEC_SQL:| WHENEVER SQLERROR DO QHANDLER
130 END
```

```

140 WHILE NOT(FINISHED):| Prompt SQL statements
150   STMT="":ERROR=FALSE
160   SHOW "SQL>";:OBTAIN STMT
170   IF KEY<>"ENTER"OR STMT=""
180     FINISHED=TRUE
190   ELSE
200     UNPAD STMT BEFORE
210     TMP=UPPERCASE(STMT(1,6))
220     IF TMP<>"SELECT"
230       EXEC_SQL:| EXECUTE IMMEDIATE :STMT
240       END
250     ELSE
260       EXEC_SQL:| PREPARE S1 FROM :STMT
270       END
280     IF NOT(ERROR)
290       EXEC_SQL:| DESCRIBE S1 INTO DA
300       END
310       IF SQLCA("SQLCODE")=0
320         DO DB2_SHOW_TABLE(STMT)
330         END
340       END
350     END
360   END
370 END
380 EXIT
390 |
400 | Allow continuation after failure to execute SQL statement
410 |
420 ENTRY QHANDLER
430 SHOW "*** SQL ERROR CODE =";SQLCA("SQLCODE")
440 SHOW "*** SQL MESSAGE   =";SQLCA("SQLERRMC")
450 ERROR=TRUE
460 SHOW "*** Press RETURN to continue";:WAIT
470 EXIT

```

Dynamic column select

This program uses dynamically-executed statements to retrieve the following information from a table specified by the user:

- ◆ Column name
- ◆ Type
- ◆ Length
- ◆ First row of the column

See the following sample code for 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)
70 .COUNTER=1
80 .SHOW "PLEASE ENTER TABLE NAME:"
90 .OBTAIN TABLE_NAME
100 .SQL_FUNCTION="SELECT * FROM"+TABLE_NAME
110 .EXEC_SQL
120 ..| EXECUTE IMMEDIATE :SQL_FUNCTION
130 .END
140 .SQL_FUNCTION="FETCH USING DESCRIPTOR"
150 .EXEC_SQL:| PREPARE S1 FROM :SQL_FUNCTION
160 .END
170 .SQLDA("SQLDA1")=NEW

```

```
180 .EXEC_SQL: | DESCRIBE S1 INTO SQLDA1
190 .END
200 .EXEC_SQL: | EXECUTE S1 USING DESCRIPTOR SQLDA1
210 .END
220 .COUNTER=COUNTER+1
230 .SHOW"COLUMN NAME",AT(25),"TYPE",AT(45),
232 .`"LENGTH",AT(55),"DATA"
240 .WHILE COUNTER<SQLDA("SQLDA1","SQLD")
250 ..SHOW SQLDA("SQLDA1","SQLCOLNAME",COUNTER)
260 ..'AT(25),SQLDA("SQLDA1","SQLTYPE",COUNTER)
270 ..'AT(45),SQLDA("SQLDA1","SQLELENGTH",COUNTER)
280 ..'AT(55),SQLDA("SQLDA1","SQLDATA",COUNTER)
290 ..COUNTER=COUNTER+1
300 .END
310 .WAIT
320 EXIT
```


B

SQL features that are not supported for DB2 SQL

SQL features that are not supported for DB2 SQL

The following features of SQL are not supported for DB2 SQL:

- ◆ **Specifying a host variable in a SELECT list.** Consider the following example, which is invalid because it specifies VX as a host variable in a SELECT list:

```
SELECT A, :VX, C
INTO :VA, :VB, :VC
```

- ◆ **In some cases, exact line number referencing upon syntax error detection.** In the execution of an SQL statement, once control is transferred to the database system, MANTIS cannot keep track of where the error was encountered. For the rest of this bullet point, consider the following sample code:

```
1330 ..X=X+1
1340 ..EXEC SQL
1350 ... |SELECT A,B,C
1360 ... |INTO :VA,:VB),:VC
1370 ... |FROM TABLE.1
1380 ... |WHERE A=1
1390 ..END
1400 ..X=X-VA
```

An error occurs in the INTO clause (line 1360) of the preceding sample code. In this INTO clause, there should not be a right parenthesis. MANTIS SQL Support reports this error differently for a bound program than for an unbound program:

- For a bound program—In the FAULT, MANTIS points to the line of the last SQL statement (the line immediately preceding the END statement). In the preceding sample code, this is line 1380.
- For an unbound program—In the FAULT, MANTIS points to the last line in the program block. In the preceding sample code, this is line 1390.

- ◆ **Using a single instruction, implicitly copying the contents of one SQLDA data structure into another.** You can individually pass each element of one SQLDA data structure to the corresponding element of a different SQLDA data structure. However, MANTIS SQL Support does not permit the following statement:

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

- ◆ **Porting MANTIS programs containing dynamic SQL statements between MANTIS SQL Support for the IBM mainframe and MANTIS SQL Support for OpenVMS and UNIX.** Although you cannot port programs containing dynamic SQL statements between the two systems, be aware that you are able to port programs containing static embedded SQL between the two systems.

C

MANTIS SQL Support vs. SQL in COBOL



For convenience, this guide refers to all third-generation SQL implementations as “SQL in COBOL.”

SQL in MANTIS SQL Support is very similar to SQL in COBOL. However, there are some differences, which this appendix summarizes.

Differences between MANTIS SQL Support and SQL in COBOL

Consider the following differences between MANTIS SQL Support and SQL in COBOL:

- ◆ **MANTIS SQL Support permits no MANTIS comments (that are not all or part of an SQL statement) within an EXEC_SQL-END block.** In a MANTIS program, you must embed an SQL statement as a standard MANTIS comment and surround the SQL statement with an EXEC_SQL-END block. MANTIS SQL Support considers all comments within this block to be SQL statement text. For example, consider the following sample code, in which lines 4620 through 4650 are within the EXEC_SQL-END block:

```
4610 EXEC_SQL
      4620 . | SELECT SALARY
      4630 . | INTO :EMPL_SAL
      4640 . | FROM EMPLOYEE_TABLE
      4650 . | WHERE NAME=:EMP_NAME
4660 END
```

COBOL, however, permits comments within an EXEC_SQL-END block.

- ◆ **WHENEVER settings.** Due to the interpretive nature of MANTIS, WHENEVER settings, when used with conditional statements, may have different effects than they would under SQL in COBOL.
- ◆ **DO statement vs. GOTO statement.** In an SQL WHENEVER statement, MANTIS SQL Support uses the MANTIS DO statement where MANTIS in COBOL would use a GOTO clause. For more information, see “[SQL WHENEVER statement](#)” on page 61.
- ◆ **STOP statement vs. FAULT statement.** In an SQL WHENEVER statement, MANTIS SQL Support uses the FAULT statement where MANTIS in COBOL would use a STOP statement. For more information, see “[SQL WHENEVER statement](#)” on page 61.
- ◆ **Default for the SQLERROR condition.** In an SQL WHENEVER statement, MANTIS SQL Support uses a default of FAULT for the SQLERROR condition, while SQL in COBOL uses a default of CONTINUE for the SQLERROR condition.

- ◆ **Ranges of applicability for WHENEVER settings.** In an SQL WHENEVER statement, MANTIS SQL Support's WHENEVER settings may have different ranges of applicability than the WHENEVER settings in SQL in COBOL. For more information, see "[Scope of the WHENEVER statement](#)" on page 61.
- ◆ **Accessing SQLCA elements.** MANTIS SQL Support accesses SQLCA elements through the SQLCA statement and the SQLCA function, while SQL in COBOL accesses SQLCA elements as data items. For more information, see "[SQLCA function and SQLCA statement](#)" on page 69.
- ◆ **Accessing elements in SQLDA data structures.** MANTIS SQL Support accesses SQLDA elements through the SQLDA statement and the SQLDA function, while SQL in COBOL accesses SQLDA elements as data items. For more information, see "[Scope of cursors, statements, and SQLDA data structures](#)" on page 54.
- ◆ **Message sources.** In a MANTIS SQL Support application, messages come to you from three sources:
 - MANTIS nucleus
 - MANTIS SQL Support
 - Database system

To find detailed explanations for, and actions to take in response to, MANTIS SQL Support messages, refer to [MANTIS Messages and Codes, OpenVMS/UNIX](#), P39-1330.

- ◆ **SQLCA and SQLDA functions vs. SQL INCLUDE statements.** MANTIS SQL Support uses the SQLCA and SQLDA functions where SQL in COBOL would use SQL INCLUDE statements. (MANTIS SQL Support does not support an SQL INCLUDE statement; INCLUDE denotes a preprocessor action.) For more information on:
 - SQLCA—See "[SQLCA function and SQLCA statement](#)" on page 69.
 - SQLDA—See "[Scope of cursors, statements, and SQLDA data structures](#)" on page 54.
- ◆ **DECLARE statements.** In MANTIS SQL Support, unlike in SQL in COBOL, DECLARE statements are unnecessary for tables and views.

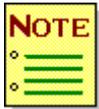
D

MANTIS vs. SQL

Differences between MANTIS and SQL

Consider the following differences between MANTIS and SQL:

- ◆ **Character-string-constant delimiters.** In MANTIS, double quotes (") delimit character-string constants. However, in SQL, apostrophes (') delimit character-string constants.
- ◆ **Data types.** SQL uses different data types than MANTIS. This requires conversion between the two kinds of data types. Permitted data type conversions are listed in [“Data conversion between MANTIS SQL Support and DB2”](#) on page 48.



Only data type codes for MANTIS-compatible data types are returned in the SQLCOLTYPE element of an SQLDA data structure. Valid data types are therefore limited to those listed in [“SQLCA elements”](#) on page 74. SQL supports many more data types than MANTIS.

Index

A

allocating an SQLDA 91

C

COBOL SQL, differences
from MANTIS SQL
Support 27
column select, dynamic 124
COMMIT, using in
embedded SQL
applications 75
CONNECT 31
connection to and
disconnection from
DB2 56
cursor scope 54
cursors for prepared
statements 108

D

data type conversion
between MANTIS vs.
DB2 48
DB2
connection and
disconnection 56
data conversion 48
signing on to 31
SQL features that are not
supported for DB2 SQL
127
DBTYPE 30
DELETE routines 120
disconnection from and
connection to DB2 56
dynamic SQL
cursor names 108
definition 28
overview 79

E

embedding SQL statements
description 20
rules for 34
environment variables 27
error messages 77
EXEC_SQL-END block
line numbers associated
with 52
syntax for 58
explicit sign-on to DB2 31

G

GOTO, MANTIS SQL
Support equivalent 61

H

host variables
description 21
using 42

I

implicit sign-on to DB2 31
indicator variables
description 25
using in SQL statements
46
input host variables 22
INSERT routines 110

L

logical names 27

M

MANTIS

- differences from SQL 133
- variables. *See also* host variables

MANTIS SQL Options 30

MANTIS SQL Support

- data type conversion 48
- differences from SQL in COBOL 27, 130

- processing of SQL statements 20

programs

- considerations for writing 49

- rules for embedding SQL statements 34

- running with non-SQL

 - MANTIS programs 19

- security 28

- software requirements 26

- SQL statements and 51

Master User facilities 29

N

non-SQL MANTIS programs,
using with MANTIS
SQL programs 19

O

output host variable 23

P

parameter of SQL statement
24

processing SQL statements
20

Q

QUERY-like function,
dynamic 122

R

requirements for MANTIS
SQL Support software
26

RESET, using in embedded
SQL applications 75

ROLLBACK, using in
embedded SQL
statements 75

rules for COMMIT and
RESET in embedded
SQL applications 75

S

sample code
 DELETE routines 120
 dynamic column select 124
 dynamic QUERY-like
 function 122
 INSERT routines 110
 SELECT routines 117
 UPDATE routines 114
 scope of cursors,
 statements, and named
 SQLDA data structures
 54
 security in MANTIS SQL
 Support 28
 SELECT routines 117
 sign-off. *See* disconnection
 sign-on to DB2 31
 SQL
 differences from MANTIS
 133
 features that are not
 supported for DB2 SQL
 127
 statements
 CONNECT 31
 dynamic execution of 80
 embedding in MANTIS
 programs 20
 EXEC_SQL 58
 invalid for dynamic
 execution 82
 MANTIS processing of 20
 rules for embedding in a
 MANTIS program 34
 using indicator variables
 in 46
 WHENEVER 61

SQLCA function and SQLCA
 statement 69
 SQLDA
 allocating 91
 data structure scope 54
 repeating elements 104
 structure of 87
 SQLSSNINC 30
 SQLVARINC 30
 statement scope scope 54
 static SQL 28
 Super User. *See* Master
 User
 syntax
 EXEC_SQL-END 58
 WHENEVER 61

U

UPDATE routines 114
 updating MANTIS user
 profile 30

V

variables. *See* MANTIS
 variables or host
 variables

W

WHENEVER 61

