

Cincom

AD/ADVANTAGE

MANTIS Rdb Programming OpenVMS

P39-1350-00



AD/Advantage[®] MANTIS Rdb Programming OpenVMS

Publication Number P39-1350-00

© 1993, 1994, 1997, 1998, 1999, 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This document contains unpublished, confidential, and proprietary information of Cincom.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 [™]	Mindspeed [™]
CINCOM [®]	iD Consulting [™]	MindspeedXML [™]
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 [™]		

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

The *AD/Advantage MANTIS Rdb Programming OpenVMS*, P39-1350-00, is dated February 12, 2001. This document supports Release 2.8 of MANTIS.

We welcome your comments

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

Cincom Technical Support for AD/Advantage

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

E-mail: helpna@cincom.com

Phone: 1-800-727-3525

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



Contents

About this book	ix
Using this document.....	ix
Document organization.....	ix
Revisions to this manual.....	x
Conventions.....	xi
MANTIS documentation series.....	xiv
Educational material.....	xv
MANTIS SQL support overview	17
Software requirements for MANTIS SQL support.....	19
Differences between SQL in MANTIS and SQL in COBOL.....	19
Logical names.....	20
Static and dynamic SQL.....	20
Security.....	20
System maintenance	21
MANTIS SQL options.....	22
Update user profile.....	22
Signing on.....	23
Embedding SQL statements in MANTIS programs	25
Embedding rules.....	26
Using host variables.....	29
Referencing values in a MANTIS array.....	30
MANTIS versus SQL data types.....	30
Indicator variables.....	31
Data conversion between MANTIS SQL support and the Rdb/VMS database.....	32
MANTIS interface to Rdb.....	34
Programming considerations	35
Running an EXEC_SQL-END block.....	37
The scope of cursors, statements and SQLDAs.....	38

Connection to the Rdb/VMS database.....	39
Disconnection from the Rdb/VMS database.....	40
The MANTIS EXEC_SQL statement	41
The SQL WHENEVER statement.....	43
Declarative versus interpretive WHENEVER statements	47
Scope of the WHENEVER statement	48
The SQL COMMIT/ROLLBACK statement.....	48
The SQL SET DBNAME statement	48
The SQLCA in MANTIS SQL support	49
SQLCA syntax.....	49
SQLCA elements	51
COMMIT and ROLLBACK and MANTIS SQL support's COMMIT and RESET	53
Error messages.....	55
Dynamic SQL in MANTIS SQL support	57
An overview of dynamic SQL.....	57
Executing a statement dynamically.....	59
Code sequence for dynamic SQL	59
The SQLDA structure.....	61
Allocate an SQLDA	63
Deallocate an SQLDA.....	64
Move data from your program into an SQLDA header element.....	65
Move data from your program into an SQLDA repeating element.....	69
Move data from an SQLDA header element into your program.....	73
Move data from an SQLDA repeating element into your program.....	74
Sample MANTIS SQL programs	77
A static insert routine.....	78
A dynamic insert routine.....	80
A static update routine	82
A dynamic update routine	83
A static select routine.....	85
A dynamic select routine	86
A static delete routine.....	88
A dynamic delete routine.....	89
A dynamic query-like function	90
A dynamic column select	93
Features not supported	95
Differences: MANTIS SQL support versus SQL in COBOL; MANTIS versus SQL	97
SQL in MANTIS SQL support versus SQL in COBOL.....	97

MANTIS versus SQL	99
Index	101

About this book

Using this document

MANTIS[®] is an application development system that consists of design facilities (for example, screens and files) and a programming language. This manual describes MANTIS SQL Support for Rdb/VMS[™].

Document organization

The information in this manual is organized as follows:

Chapter 1—MANTIS SQL support overview

Provides an overview of MANTIS SQL support and how to use it to create MANTIS applications that use SQL.

Chapter 2—System maintenance

Provides supplemental information to MANTIS administration and installation documents for the Master User.

Chapter 3—Embedding SQL statements in MANTIS programs

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

Chapter 4—Programming considerations

Describes implications for program design 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 static and dynamic MANTIS SQL programs.

Appendix B—Features not supported

Lists the features of SQL that are not supported for Rdb SQL.

Appendix C—Differences: MANTIS SQL support versus SQL in COBOL; MANTIS versus SQL

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

Index

Revisions to this manual

The following changes have been made for this release:

- ◆ Updated Publication Release Number from P25-1350-03 to P39-1350-00.
- ◆ Updated publication titles and numbers under “**MANTIS documentation series**” on page xiv and the entire document.
- ◆ Updated “**Software requirements for MANTIS SQL support**” on page 19.
- ◆ Updated Facility Selection screen in “**System maintenance**” on page 21.

Conventions

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

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

Convention	Description	Example
Braces { }	<p>Indicate selection of parameters. (Do not attempt to enter braces or to stack parameters.) Braces surrounding stacked items represent alternatives, one of which you must select.</p> <p>The example indicates that you must enter FIRST, LAST, or a value for <i>begin</i>.</p>	<pre> { FIRST <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> <p>Underlining also indicates an allowable abbreviation or the shortest truncation allowed.</p> <p>The example indicates that you can enter either PRO or PROTECTED.</p>	<pre> SCROLL [ON OFF [row][,col]] </pre> <p><u>PROTECTED</u></p>
Ellipsis points...	<p>Indicate that the preceding item can be repeated.</p> <p>The example indicates that you can enter (A), (A,B), (A,B,C), or some other argument in the same pattern.</p>	<pre> (argument, ...) </pre>

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 ' ' single quotation marks</p>	$[\text{LET}]_p \begin{bmatrix} (i) \\ (i, j) \end{bmatrix} [\text{ROUNDED}(n)] = e1 [e2, e3\dots]$
<div style="border: 1px solid black; padding: 2px; display: inline-block; margin-bottom: 5px;">UNIX</div> <div style="border: 1px solid black; padding: 2px; display: inline-block;">OpenVMS</div>	<p>Information specific to a certain operating system is flagged by a symbol in a shadowed box (e.g., UNIX) indicating which operating system is being discussed. Skip any information that does not pertain to your environment.</p>	<div style="border: 1px solid black; padding: 2px; display: inline-block; margin-bottom: 5px;">UNIX</div> DBA will run on any terminal that supports the cursor library.

MANTIS documentation series

MANTIS is an application development system designed to increase productivity in all areas of application development, from initial design through production and maintenance. MANTIS is part of AD/Advantage[®], which offers additional tools for application development. Below are listed the manuals offered with MANTIS in the OpenVMS[™] and UNIX[®] environments, organized by task. You may not have all the manuals that are listed here. For a synopsis of each manual, refer to the *AD/Advantage MANTIS Application Development Tutorial OpenVMS/UNIX*, P39-1340.

Getting started

- ◆ *AD/Advantage MANTIS 2.8.x Installation and Startup OpenVMS/UNIX*, P39-0027*

General use

- ◆ *AD/Advantage MANTIS Facilities OpenVMS/UNIX*, P39-1300*
- ◆ *AD/Advantage MANTIS Language OpenVMS/UNIX*, P39-1310
- ◆ *AD/Advantage MANTIS Messages and Codes OpenVMS/UNIX*, P39-1330*
- ◆ *AD/Advantage MANTIS Application Development Tutorial OpenVMS/UNIX*, P39-1340
- ◆ *AD/Advantage MANTIS SUPRA SQL Programming OpenVMS/UNIX*, P39-1345
- ◆ *AD/Advantage MANTIS Rdb Programming UNIX*, P39-1350
- ◆ *AD/Advantage MANTIS Oracle Programming UNIX*, P39-1355



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

Master User tasks

- ◆ *AD/Advantage MANTIS Facilities OpenVMS/UNIX*, P39-1300*
- ◆ *AD/Advantage MANTIS Administration OpenVMS/UNIX*, P39-1320
- ◆ *AD/Advantage MANTIS 2.8.x Installation and Startup OpenVMS/UNIX*, P39-0027*



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

Educational material

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

1

MANTIS SQL support overview

MANTIS is an application development system for developing, testing, executing, and documenting applications interactively. MANTIS SQL Support is an extended version of MANTIS. It enables you to create MANTIS applications that access database systems with SQL.

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

You embed SQL statements in a MANTIS application program as standard MANTIS comments. As MANTIS SQL Support encounters each SQL statement, it transparently prepares it for execution and executes it. The embedding of SQL in MANTIS looks similar to the preprocessor output for SQL statements embedded in other host languages such as COBOL.

Precede each SQL statement with an EXEC_SQL statement and follow it with an END statement, as shown below. The vertical bar (|) is the MANTIS comment character. MANTIS automatically sets the indentation level (number of preceding periods for the EXEC_SQL structure).

```
4580 TEXT EMP_NAME(30)
4590 X=N*RATE
4600 EXEC_SQL
4610 .| SELECT EMPLNAME
4620 .| INTO :EMPL_NAME
4630 .| FROM EMPLOYEE.TABLE
4640 .| WHERE EMPLNAME="SMITH"
4650 END
4660 DO CLEAN_UP
```

MANTIS variables in SQL statements are called host variables. Syntactically, a colon always precedes a host variable in an SQL statement. An input host variable is a MANTIS variable passed to SQL and is used to select, insert, delete, or update data. A MANTIS variable which receives data from the database is called an output host variable. Host variables are also used as parameters of the SQL statements. Optionally, you can specify an indicator variable along with a host variable. The database system sets the indicator variable to indicate null values or to signal that a value was truncated. [“Embedding SQL statements in MANTIS programs”](#) on page 25 describes embedding SQL statements in MANTIS programs.

Software requirements for MANTIS SQL support

Software requirements for MANTIS SQL Support for Oracle/Rdb/ include Open/VMS Release 6.2 and above. It runs against the Oracle Rdb/ 7.0 databases.

Differences between SQL in MANTIS and SQL in COBOL

SQL in MANTIS SQL Support is essentially the same as its implementations in third-generation languages, such as FORTRAN and COBOL. (For convenience, in MANTIS documentation these implementations are generalized as “SQL in COBOL.”) Some differences do exist between MANTIS SQL Support and SQL in COBOL, mostly due to the interpretive, rather than compiled, nature of MANTIS. These differences are noted in the appropriate chapters of this manual and are summarized in “[Differences: MANTIS SQL support versus SQL in COBOL; MANTIS versus SQL](#)” on page 97 . In brief, they are as follows:

- ◆ In the SQL WHENEVER statement: WHENEVER settings may have different effects when used with conditional statements than they would in SQL in COBOL due to the interpretive nature of MANTIS.
- ◆ The GOTO clause is replaced by a standard MANTIS DO statement, and STOP is replaced by FAULT.
- ◆ The default for the condition SQLERROR is FAULT; in SQL in COBOL, the default is CONTINUE.
- ◆ WHENEVER settings may have different ranges of applicability than they would in SQL in COBOL.
- ◆ SQLCA elements are accessed through the SQLCA function.
- ◆ Elements in SQLDAs are accessed through the SQLDA function.
- ◆ In a MANTIS SQL Support application, messages come from three sources: the MANTIS nucleus, MANTIS SQL Support, and the database system. For explanations and actions for MANTIS SQL Support messages, refer to [AD/Advantage MANTIS Messages and Codes OpenVMS/UNIX](#), P39-1330.

Logical names

In this manual, the term logical name refers to an identifier or variable which stands for another name or value.

In MANTIS for OpenVMS, the logical names used by MANTIS correspond directly to OpenVMS logical names.

In MANTIS for UNIX, logical names are implemented as environment variables. You must ensure that any shell variables you want to 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. In a static application, all SQL statements are defined before run time. In a dynamic application, SQL statements are not defined until run time; they are specified during program execution.

Security

Security in MANTIS SQL Support is handled solely through the database system and MANTIS. Make sure that users have authorized access to the views they require.

2

System maintenance

The following considerations assume that you have installed MANTIS. If you are installing MANTIS or MANTIS SQL Support, refer to *AD/Advantage MANTIS Administration OpenVMS/UNIX*, P-39-1320, and *AD/Advantage MANTIS 2.8.x Installation and Startup OpenVMS/UNIX*, P39-0027. This chapter supplements those publications.

As the Master User, you have certain facilities and information that are available to you alone. When you sign on as Master User, your Facility Selection menu appears as shown in the following screen illustration.

```

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 and SQLVARINC. For information on these options, refer to *AD/Advantage MANTIS Administration OpenVMS/UNIX*, P-39-1320.

Update user profile

The MANTIS User Profile contains a field in which you specify the default SQL DBTYPE. Either SUPRA, RDB, or ORACLE can be specified in this field.

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

The current DBTYPE can be changed at any time by the MANTIS EXEC_SQL and SQLCA statements.



Not all DBTYPEs are supported on all platforms. For example, Rdb/VMS only runs in the OpenVMS environments.

Signing on

The MANTIS RDI interface performs automatic attachment to the Rdb/VMS database identified by the logical name SQL\$DATABASE. This method of attachment is the same as that used by third-generation language programs (such as FORTRAN or COBOL) which are processed by the Rdb precompilers.

MANTIS does not support explicit database connection by any embedded SQL statement (such as the SUPRA CONNECT statement). MANTIS does support the setting of the logical name SQL\$DATABASE in several ways, as specified in “[Connection to the Rdb/VMS database](#)” on page 39 .

Consideration

The MANTIS logical MANTIS_RDI_IF must be defined in order to connect to the RDB SQL database. This logical is defined in the MANTIS_CLASS_XXXX_INIT.COM procedure where XXXX is the class name.

3

Embedding SQL statements in MANTIS programs

This chapter describes the rules you must follow when embedding SQL statements in a MANTIS program. It also explains how to reference host variables and MANTIS entities, and how the database system converts data values between SQL and MANTIS. A general working knowledge of MANTIS, of Rdb/VMS, and of SQL is assumed. For more information on MANTIS language conventions, refer to *AD/Advantage MANTIS Language OpenVMS/UNIX*, P39-1310.

Embedding rules

You embed SQL statements in a MANTIS application program as standard MANTIS comments (preceded by a vertical bar). Standard SQL syntax rules apply. Each SQL statement is bracketed with an EXEC_SQL-END block, as shown below. As the examples show, EXEC_SQL causes the statements that follow it to be indented.

Follow these rules when embedding SQL statements in a MANTIS application program. Standard rules for using MANTIS comments apply.

- ◆ Only one SQL statement can be within an EXEC_SQL-END block.

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

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

- ◆ Any text between EXEC_SQL and END must be part of an SQL statement and must be preceded by a vertical bar (|). Once MANTIS SQL Support encounters a vertical bar, the rest of the physical line is considered a single SQL statement. Other MANTIS statements or comments are not permitted.

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

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

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

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

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

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

- ◆ A colon within an EXEC_SQL-END block identifies a MANTIS host variable, not a new statement.

```
..EXEC_SQL
...  FETCH C1 INTO :A           C1 is an SQL entity;
..END                          A is a MANTIS host variable
```

- ◆ An SQL statement in an EXEC_SQL-END block can be broken into multiple lines. MANTIS reads the text on two consecutive comment lines in an EXEC_SQL-END block as if it were separated by a single blank (one statement).

```
..EXEC_SQL           is equivalent to           ..EXEC_SQL
...  OPEN           ..EXEC_SQL
...  C1             ..EXEC_SQL
..END               ..END
```

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

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

```
..EXEC_SQL           is equivalent to           ..EXEC_SQL
...|   OPEN C1      ..EXEC_SQL
..END               ..END
```

Multiple spaces between words in statements are compressed.

```
..EXEC_SQL           is equivalent to           ..EXEC_SQL
...|   OPEN C1      ..EXEC_SQL
...|   OPEN         ..EXEC_SQL
...|   C1           ..EXEC_SQL
...
..END
```

- ◆ An SQL statement “attached” to an EXEC_SQL statement with a colon (the MANTIS statement-separator character) is part of the SQL statement; it is considered to be within the EXEC_SQL-END block.

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

- ◆ A MANTIS statement on the same line as the END in an EXEC_SQL-END block is not executed. This rule is consistent with the rules for using END with MANTIS IF, WHILE, FOR, WHEN, and UNTIL statements. MANTIS comments are permitted.

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

Using host variables

A MANTIS variable used to provide input or receive output from the database is called a host variable. A host variable is identified within an SQL statement by a colon prefix. In the following example, EMPL is a host (MANTIS) variable.

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

Like other MANTIS variables, host variables are implicitly declared when they are first used if they are not explicitly declared before appearing in the EXEC_SQL-END block. Any previously undefined MANTIS variable referred to in an SQL statement is automatically declared as a MANTIS BIG variable. (A MANTIS BIG variable is a numeric floating-point variable 16 digits long.)

```
..BIG A                                is equivalent to ..EXEC_SQL
..EXEC_SQL                                ... FETCH C1 INTO :A
... FETCH C1 INTO :A                        ..END
..END
```

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

Referencing values in a MANTIS array

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

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



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

You may use host variables in SQL expressions. Each host variable must be preceded by a colon, as shown in the following example:

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

MANTIS versus SQL data types

When the database system transfers data to or from a host variable, MANTIS causes the database system to automatically convert the data's type from an SQL type to a MANTIS type, and vice versa. See [“Data conversion between MANTIS SQL support and the Rdb/VMS database”](#) on page 32 for a summary of how types are converted. Be sure to note that truncation, overflow, and rounding may occur.

Indicator variables

Optionally, you can include an indicator variable along with a host variable in SQL statements.

As its name implies, the indicator variable indicates whether the host variable contains a real value or is NULL or MISSING. Indicator variables are interpreted as follows:

Value	Meaning
=0	The host variable is a defined value, no error.
<0	The host variable is a NULL or MISSING value.

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

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

Like host variables, indicator variables can be explicitly or implicitly defined. Only numeric variables can be used as indicator variables. The default in implicit declaration is a MANTIS BIG variable. Variable types are described in “[Data conversion between MANTIS SQL support and the Rdb/VMS database](#)” on page 32. Rdb/VMS interprets indicator values as integers whereas they may be specified as floating point values in MANTIS. Therefore a value of -0.9 will not specify a NULL value because it is converted to zero before being interpreted.

When reading data from the database (SELECT/FETCH), you should supply indicator variables for any columns that may contain NULL values.



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

Data conversion between MANTIS SQL support and the Rdb/VMS database

Rdb performs data conversion between all data types, including conversions between numeric and string data types. Be sure to match host variable data types with database columns so that correct results are obtained.



Rdb does not flag truncation of data as a warning.

The following Rdb data types can be returned in the SQLDA.

	SQLTYPE	Description
449	VARCHAR	Variable length string up to length 65535
453	CHAR	Space padded character string
481	FLOAT	4 or 8 byte floating point, depending on SQLLEN
485	DECIMAL	Packed-decimal with precision and scale specified by SQLLEN
497	INTEGER	4 byte integer
501	SMALLINT	2 byte integer
503	DATE	64-bit number in OpenVMS absolute date-time format
505	QUADWORD	8 byte integer

MANTIS data types can only be INTEGER, SMALL, BIG, DECIMAL or TEXT, but can be mapped to any Rdb/VMS data type, so data transfer between MANTIS host variables and database columns may frequently involve conversions. The following conversion error conditions apply:

- ◆ MANTIS string → Rdb/VMS numeric
 - The MANTIS string is not a valid number in decimal or scientific E-notation.
 - The MANTIS string contains commas.
 - The magnitude of the converted MANTIS string is greater than the magnitude supported by the target number.
- ◆ MANTIS numeric ↔ Rdb/VMS number
 - The magnitude of the source number is greater than the magnitude supported by the target number.
- ◆ MANTIS string → Rdb/VMS DATE

(Note: string format should be “YYYYMMDDHHMMSSNN.”)

 - The MANTIS string is less than 8 characters in length.
 - The MANTIS string does not specify a valid date and time (up to its length or to 16 places, whichever is less).

MANTIS interface to Rdb

MANTIS interfaces to Rdb and Rdb/VMS via a dynamically loaded shareable image. This image is identified by the logical name MANTIS_RDI_IF, and is referred to as the RDI interface. This interface imposes certain limitations and restrictions on the Rdb support in MANTIS. The primary limitation is that MANTIS executes all embedded SQL statements dynamically, and MANTIS dynamic SQL support is merely another layer over its underlying use of dynamic SQL.

Another limitation involves the relationship between embedded SQL statements and cursor names. MANTIS must PREPARE all embedded SQL statements before executing them. The RDI interface requires that when a statement is prepared, the cursor name associated with that statement must also be nominated on the same call.

However, for MANTIS dynamic SQL support, cursor names are not known at the time of the PREPARE (since 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

In this manual the term 'real cursor' is used to refer to cursors declared internally by MANTIS, to distinguish them from cursor names declared in a MANTIS program. When a real cursor is created for a non-SELECT dynamic statement, it is (effectively) a wasted resource.



The RDI interface supplied on the MANTIS release tape supports up to 100 cursors with no limit on the number of statements.

4

Programming considerations

To use MANTIS SQL Support, you simply embed the appropriate SQL statements in your MANTIS application program as standard MANTIS comments, enclosed in EXEC_SQL-END blocks. As MANTIS SQL Support encounters each SQL statement, it prepares it for execution and then executes it, in effect performing the same steps (preprocess, compile, link, and load before the run) that are executed with a COBOL program that contains embedded SQL statements. However, unlike COBOL, the MANTIS program can be modified, including the SQL statements, and then immediately re-executed by issuing the RUN command.

The fact that MANTIS SQL Support is interpretive has several implications for program design, as you will see in this chapter. Before you begin writing MANTIS SQL Support programs, you should be aware of these implications and other programming considerations. Briefly, they are as follows:

- ◆ MANTIS stores an EXEC_SQL-END block as a single line of text internally and associates the line number of the last program line in the block with this single line. There are minor distinctions between bound and unbound versions of a program.
- ◆ The scope of a cursor or SQL statement is local. Since both are SQL entities and not MANTIS entities, you cannot pass them as parameters or use them in non-SQL MANTIS statements.
- ◆ The WHENEVER statement in MANTIS SQL Support differs slightly from the WHENEVER statement used in other languages. It has different syntax, defaults, and possibly effects.
- ◆ Elements in the SQLCA are accessed through a MANTIS function called SQLCA, rather than as elements of an SQLCA data structure.
- ◆ The effects of COMMIT and ROLLBACK in Rdb/VMS differ slightly from COMMIT and RESET in MANTIS. In Rdb/VMS, COMMIT commits only SQL, and only for the specified SQL session. In MANTIS, COMMIT and RESET commit everything (including SQL).
- ◆ Error messages can come from different sources: MANTIS SQL Support, the MANTIS nucleus, and the database system.

Each of these topics is discussed in detail in this chapter.

Running an EXEC_SQL-END block

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

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

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

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

The scope of cursors, statements and SQLDAs

Cursors, statements, and named SQLDAs are identified by names, and are basic to dynamic SQL programs. The scope of a cursor name, statement name, or SQLDA name is limited to the program or external subprogram context in which the name is declared. When a cursor, statement or SQLDA name is redeclared in an external subprogram, it refers to a different structure.

MANTIS maps statement and cursor names onto RDB statement and cursor names. The scope of the RDB statement and cursor names is the MANTIS main program, or the entire connect time to the Rdb/VMS database. This mapping requires further explanation, since certain error conditions relating to open cursors can cause real cursor names to be displayed in error messages.

As each MANTIS cursor is opened, another global real cursor name of the form 'Cnnn' (where 'nnn' is a 3-digit number allocated by MANTIS) is allocated and mapped to the MANTIS cursor name. Therefore, each MANTIS external subprogram maps a range of real cursor names. Upon subprogram EXIT, 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 next invocation of the same subprogram.

The limited scope of a MANTIS cursor name is illustrated by this mapping. MANTIS cursor C1 in main program MAIN might be mapped to real cursor C001, and MANTIS cursor C1 in external subprogram SUB could be mapped to real cursor C006.

Connection to the Rdb/VMS database

The MANTIS RDI interface performs automatic attachment to the Rdb/VMS database identified by the logical name SQL\$DATABASE. This method of attachment is the same as that used by third generation language programs (such as FORTRAN or COBOL) which are processed by the RDB precompilers.

MANTIS does not support explicit database connection by any embedded SQL statement (such as the SUPRA CONNECT statement). MANTIS does support the setting of the logical name SQL\$DATABASE as follows:

- ◆ SQLCA("DBNAME")="RDB" (see MANTIS SQLCA statement)
- ◆ EXEC_SQL: | SET DBNAME *dbname-spec*
- ◆ The MANTIS SET \$LOGICAL *statement*
- ◆ EXEC_SQL("RDB")

The new value of SQL\$DATABASE will be set in the PROCESS logical name table in USER mode.

Disconnection from the Rdb/VMS database

MANTIS detaches from the Rdb/VMS database by executing the SQL FINISH statement. This can be performed explicitly by a MANTIS program, or implicitly by the MANTIS nucleus.

Possible causes of disconnection are:

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

MANTIS main program context cleanup occurs in the following circumstances:

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

The MANTIS EXEC_SQL statement

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

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

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

```
END
```

exp1

Description Specify the database subsystem type (DBTYPE).

Format A text expression equal to "SUPRA," "RDB," or "ORACLE."

exp2

Description *Optional.* Specify connect session number when *exp1* is used to specify SUPRA or ORACLE as the DBTYPE.

Format A numeric expression equal to a session number in the range of one through eight.

General considerations

- ◆ Multiple session support refers to the ability to connect to different SQL databases concurrently, for example, to SUPRA and Rdb/VMS databases. It also refers to the ability to have multiple connections to some databases (e.g., SUPRA), but you can only be connected to one Rdb/VMS database at a time.
- ◆ Another element of the EXEC_SQL statement is the current DBTYPE, defined as the default used when one is not specified in an EXEC_SQL statement. The current DBTYPE can be SUPRA, RDB, or ORACLE and, once set, remains in place until either:
 - Explicitly changed by another EXEC_SQL or SQLCA statement
 - Implicitly changed by signing on to another MANTIS user

The Master User can specify your default DBTYPE in your MANTIS user profile. If this function is not performed by the Master User, MANTIS uses SUPRA as the default.



Not all DBTYPEs are supported on all platforms. For example, Rdb/VMS support is restricted to the OpenVMS environment.

The SQL **WHENEVER** statement

The *WHENEVER* statement in MANTIS SQL Support differs from that of SQL in COBOL in four ways:

- ◆ It is interpretive, not compiled.
- ◆ GOTO is replaced by DO.
- ◆ The default for *SQLERROR* is *FAULT*, not *CONTINUE*.
- ◆ *FAULT* is an extra *WHENEVER* action that allows program termination upon the specified condition.

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

WHENEVER condition action

condition

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

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

SQLERROR

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

Default action FAULT

SQLWARNING

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

Default action CONTINUE

NOT FOUND

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

Default action CONTINUE

action

Description *Required.* Specify the action to be taken when the named condition is met.

Options Valid actions are DO *entry-name*[(*parms*)], FAULT, and CONTINUE.

DO *entry-name*[(*parms*)]

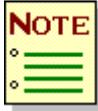
Description *Optional.* Indicates a standard MANTIS DO (internal or external) and corresponds to the WHENEVER-GOTO SQL statement in SQL in COBOL. WHENEVER-DO transfers control to the specified internal routine or external program whenever the named condition is encountered.

Considerations

- ◆ WHENEVER-DO can transfer control to an internal routine or external program, which in turn can contain any MANTIS logic, including CHAIN, EXIT, or STOP statements. The current values of any DO arguments at the time of the EXEC_SQL that caused the DO to occur are passed to the named subroutine. The subroutine EXIT returns control to the next statement following the EXEC_SQL that caused the DO to occur.
- ◆ The WHENEVER-DO action resembles the existing functionality of the SET TRAP statement in MANTIS. If the DO portion of a WHENEVER-DO contains an error, MANTIS returns a MANTIS error message associated with the DO statement, not an SQL WHENEVER-type error. MANTIS displays the line in error in the subroutine. The WHENEVER statement may be outside of the current execution path. Remember that DO is executed as a result of an SQL statement raising the condition with which the DO action is associated.

FAULT

Description *Optional.* Terminates execution of the program and displays the generated database system message in the form of a MANTIS fault (error) message.



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

CONTINUE

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

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

Condition	Default action
SQLERROR	FAULT
SQLWARNING	CONTINUE
NOT FOUND	CONTINUE

Example

```

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

Declarative versus interpretive WHENEVER statements

When SQL is embedded in COBOL, WHENEVER is a declarative statement. It is processed when the program is precompiled, not at execution time. Thus, in a COBOL program the current WHENEVER setting is determined by sequential position. By setting we mean the combination of condition and action that the WHENEVER statement specifies. For example, in the statement:

```
WHENEVER SQLWARNING CONTINUE
```

SQLWARNING is the condition and CONTINUE is the action. Together, they make up the setting. (Remember that a WHENEVER setting is actually an accumulation of three settings: one each for the conditions SQLERROR, SQLWARNING and NOT FOUND.)

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

SQL in 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	C2 FIRST, THEN C2*
50 WHENEVER C2	C2	50 WHENEVER C2	C1 or C2*
"	C2	"	C1 or C2*
70 ENDWHILE	C2	70 ENDWHILE	C1 or C2*
80 EXEC_SQL	C2	80 EXEC_SQL	C1 or C2*

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

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

* However, if the WHILE condition is not true the first time line 40 is executed, C1 remains in effect through line 80 because line 50 was not executed.

Scope of the WHENEVER statement

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

The SQL COMMIT/ROLLBACK statement

**{ COMMIT
ROLLBACK }** **[WORK][RELEASE]**

The RELEASE parameter is supported for compatibility with MANTIS support for SUPRA SQL, and is a request to disconnect from the database (FINISH) upon successful completion of the COMMIT or ROLLBACK.

The SQL SET DBNAME statement

This statement provides an alternate way to change the setting of SQLCA("DBNAME"). It allows you to specify the Rdb/VMS database names to use in all subsequent connects. Refer to the section on SQLCA elements for more information.

The format for using the SET DBNAME statement is:

SET DBNAME **{ ' *database – name* ' }**
{ : *parameter* }

The SQLCA in MANTIS SQL support

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

SQLCA syntax

The syntax of the built-in SQLCA function (read) is shown first; the SQLCA statement (write), second, below.

***sqlca-element* = SQLCA(*element_name*)**

SQLCA(*element_name*) = *sqlca-element-value*

sqlca-element

- Description** *Required.* Specify a MANTIS variable or array element to receive the value of your SQLCA element.
- Format** Valid MANTIS variable reference: a scalar variable, a subscripted array, or a substring reference.
- Consideration** The data type of *sqlca-element* must be compatible with the data type of the SQLCA element referenced in the SQLCA function.

element_name

Description *Required.* Is or contains the name of the element to be returned or read.

Format Valid SQLCA element, as listed in “SQLCA elements” on page 51

Considerations

- ◆ Quotation marks (“”) are required when *element_name* is specified as a text literal. For example:

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

- ◆ A text variable containing the *element_name* is also valid. For example:

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



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

sqlca-element-value

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

Format Valid MANTIS expression or text literal of the appropriate type (dependent upon the element type)

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

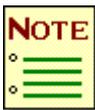
SQLCA elements

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

Element	MANTIS variable*	Usage notes
DBTYPE	TEXT(6)	Multiple session support
DBNAME	TEXT(64)	Multiple session support
SQLCAID	TEXT(8)	Read only
SQLCABC	NUMERIC	Read only
SQLCODE	NUMERIC	
SQLERRML	NUMERIC	Read only
SQLERRMC	TEXT(70)	
SQLERRP	TEXT(8)	
SQLERRD n	NUMERIC	n ranges from 1–6
SQLWARN n	TEXT(1)	n ranges from 0–7
SQLEXT	TEXT(84)	Read only

* If a data value is moved from an SQLCA element to a MANTIS variable of shorter length (e.g., an eight-character SQLCA element to a six-character MANTIS variable), the right-most characters are truncated.

Remember that SQLCAID, SQLCABC, SQLERRML, and SQLEXT are read-only. Although values can be written into the other elements, doing so does not pass any information to the database. In addition, since these elements are written to by the database, their contents may be destroyed at each EXEC_SQL statement execution. See the following discussion of SQLERRMC for more information.



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

Additional elements added to MANTIS SQL Support are described in further detail here:

DBTYPE The SQLCA("DBTYPE") statement allows you to specify the database with which MANTIS SQL Support will communicate. Specifies a one- to six-character text value for the current DBTYPE, as explained in the section on the EXEC_SQL statement. Allowed values for DBTYPE are "SUPRA," "RDB," and "ORACLE." All EXEC_SQL statements executed will now access the relational database implied by the DBTYPE, by default.

DBNAME Specifies a 1–64 character text value which becomes the database name used in all subsequent connects. If you do not specify a database name, MANTIS obtains it from the translation of the logical name SQL\$DATABASE.

SQLERRMC The SQLERRMC element and associated length parameter returns the error message text for the current SQLCODE. For example:

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

SQLCA ("SQLCODE") is reset to zero at an implicit COMMIT. Any solicited input while MANTIS has COMMIT ON causes an implicit COMMIT. MANTIS will always issue an implicit COMMIT at any terminal read operation, unless the program or user explicitly turns the automatic COMMIT logic off with the COMMIT OFF statement.

COMMIT and ROLLBACK and MANTIS SQL support's COMMIT and RESET

In MANTIS SQL Support, SQL's COMMIT and ROLLBACK statements have exactly the same effect on the database as the standard MANTIS COMMIT and RESET statements. The use of an SQL COMMIT or ROLLBACK does not imply a MANTIS COMMIT or RESET, but the use of a MANTIS COMMIT or RESET does imply an SQL COMMIT or ROLLBACK. Execution of an SQL COMMIT statement commits only SQL, and only for the specified SQL session. MANTIS automatically performs a COMMIT at terminal input.

For more information on cursor positioning, consult the *VAX SQL Reference Manual*, AA-JM32B-TE.

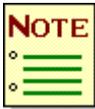


Rdb/VMS is very sensitive to the COMMIT. Care should be taken when using COMMIT and RESET in embedded SQL applications, and the following considerations should be kept in mind.

Considerations

- ◆ MANTIS COMMIT and RESET functions also COMMIT/ROLLBACK the current SQL transaction. Embedded SQL COMMIT and ROLLBACK statements only affect the SQL database.
- ◆ MANTIS COMMIT is performed in the following circumstances:
 - When MANTIS COMMIT statement is encountered.
 - When any terminal input function (CONVERSE, OBTAIN, WAIT, or MORE prompt) is encountered by any MANTIS program (including Program Design when reading command lines), unless COMMIT OFF has been specified.
 - When MANTIS runs the main program cleanup, as outlined above, the COMMIT is performed prior to database disconnection.

- ◆ MANTIS RESET is performed in the following circumstances:
 - When MANTIS RESET statement is encountered.
 - When a MANTIS FAULT occurs, except for breakpoint faults.
- ◆ Beware of the effect of the terminal input request by Program Design at this point. Your MANTIS program may go into a resource wait state, trying to update a table that another user may be reading (via SELECT).
- ◆ You must give careful consideration to the use of COMMIT and RESET in your embedded SQL applications, as well.
- ◆ Some database updates may not be validated until they are committed (for INTEGRITY). For example, when you attempt to insert a NULL value into a column with the NOT NULL attribute, your SQL COMMIT statement may fail, and will continue to fail until the transaction containing the illegal operation is rolled back.
- ◆ COMMIT closes all open cursors. If your cursors have to remain open across any terminal input, you must include a MANTIS COMMIT OFF statement in your main program.



If you are in Program Design, and have set a breakpoint, then a RESET will not occur when the breakpoint is hit (the breakpoint is a FAULT condition). However, unless you have done a COMMIT OFF, an automatic COMMIT will occur when Program Design prompts you for the next input line. Be aware that this can occur, as it could result in the program behaving differently under breakpoints.



When a MANTIS COMMIT fails to COMMIT Rdb/VMS, MANTIS automatically attempts to ROLLBACK only the Rdb/VMS database.

Error messages

This section discusses the types of messages you may receive and how they are displayed in MANTIS Program Design mode and at execution time. You can receive messages from three sources: the MANTIS nucleus, MANTIS SQL Support, and the database system.

When MANTIS encounters an error, it displays the fault message first, then the statement where the error occurred and the text of that line. Messages from the database system are prefaced with the three-character code 750. A message from the database contains the SQLCODE value and its associated text message. The format is:

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

where *nnnn* is the three- or four-digit SQLCODE value and “###...” is the message returned from the database.

For example:

```
750 SQLERROR:-1: %SQL-F-SYNTAX_ERR, syntax error.
```

would be returned from Rdb/VMS if an SQLCODE indicating invalid string constants was returned to the SQLCA and the WHENEVER SQLERROR condition was set to FAULT.

For explanations and actions for all messages generated by MANTIS, refer to [AD/Advantage MANTIS Messages and Codes OpenVMS/UNIX, P39-1330](#).

5

Dynamic SQL in MANTIS SQL support

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

An overview of dynamic SQL

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

Virtually any statement in a static application can also be executed dynamically. The principal statements that enable you to execute SQL statements dynamically are PREPARE, DESCRIBE, EXECUTE, and EXECUTE IMMEDIATE. Alternate forms of DECLARE, OPEN, and FETCH statements are used in dynamic SQL. Communication to and from the database is done using these statements and an SQLDA data structure. This structure consists of header elements and repeating elements (each repeating element group is sometimes called SQLVAR). The SQLDA contains metadata (e.g., data length and data type) about the data going between your program and the database. The SQLDA can be thought of as a representation and repository of the data being transferred.

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

Not all Rdb statements can be executed dynamically (such as the SELECT statement). This causes some problems for MANTIS SQL support, which executes all embedded SQL statements dynamically. MANTIS overcomes the limitation on dynamic execution of SELECT by implicitly performing DECLARE and OPEN statements for each static SELECT statement.

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

The following SQL statements may not be executed dynamically (by way of the dynamic PREPARE and EXECUTE statements).

BEGIN DECLARE SECTION*	FINISH
CLOSE	IMPORT
CREATE VIEW	INCLUDE*
DECLARE CURSOR	INTEGRATE**
DECLARE STATEMENT**	OPEN
DECLARE TABLE**	PREPARE
DESCRIBE	SELECT [INTO]
EDIT	QUIT
END DECLARE SECTION*	RELEASE
EXECUTE	SET**
EXPORT	SHOW**
FETCH	WHENEVER

* May be embedded in a MANTIS program, but are ignored when executed.

** Not supported in MANTIS programs, and will cause errors when executed.

Executing a statement dynamically

In general, to be executed dynamically, most SQL statements must be prepared with a PREPARE statement and then executed with an EXECUTE statement. If data is being retrieved, inserted, or updated, the program must manipulate the SQLDA between preparation and execution. This manipulation can include allocating and expanding an SQLDA, retrieving metadata from SQL with the DESCRIBE statement, and causing data transfer between SQL and MANTIS variables. The code sequence examples included in this chapter illustrate how dynamically executed SQL statements and the SQLDA work together in dynamic routines.

Code sequence for dynamic SQL

The following code provides a sample SELECT and FETCH sequence for dynamic SQL. The syntax for the PREPARE, DESCRIBE, and EXECUTE statements used here appears in the appropriate Rdb/VMS documentation. Sample code for the SQLDA built-in statement and function (described in “[The SQLDA structure](#)” on page 61) is also included in this example.

SELECT statement with input parameters

```

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

```

FETCH statement with output parameters

```

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

```

Other statements with input parameters

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

The first SQLDA statement allocates the SQLDA. Next, the PREPARE statement dynamically compiles the SQL statement. Then, the DESCRIBE statement returns metadata about the results of the SQL statement in the SQLDA. You may want to include the next SQLDA statement to supply values for input host variables. The EXECUTE statement tells the processor to execute the named statement. Finally, you may include the SQLDA built-in function to transfer data from a host variable.

Complete example code for dynamic insert, update, delete, and select routines are in “[Sample MANTIS SQL programs](#)” on page 77, along with their static equivalents.

The SQLDA structure

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

The following figure represents the structure of an SQLDA. The first four elements are header elements; they occur once per SQLDA. The next six elements repeat once per data item. A data item is either one column of an SQL table (output from SQL to your program) or the value of a host variable (input to SQL from your program). The maximum number of entries is 300. “SQLDA header elements” on page 68 and “SQLDA repeating elements” on page 75 summarize header and repeating elements.

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

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

In other programming languages, you must explicitly declare each SQLDA element as a data area in your program and then access SQLDA elements through programming statements. In MANTIS SQL Support, when you declare an SQLDA, an SQLDA with all the elements shown in the preceding illustration is built for you. The SQLDA contains the default number of repeating elements set by your Master User as one of your MANTIS Options. This value can be modified by your program.

The MANTIS SQLDA built-in statement and function allow your MANTIS program to create and maintain SQLDA data structures which are in turn used with programmed dynamic SQL statements. Your MANTIS program can use the SQLDA statement to create a named SQLDA structure and to set input host variable information. The *sqlda-name* parameter must include a text expression or literal which contains the name of a valid MANTIS variable. Your program can use the SQLDA function to retrieve information (output host variable information) about SQL table columns. The SQLDA statement and function are described and illustrated in the sections that follow. Their uses are outlined below:

- ◆ To allocate or deallocate an SQLDA:

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

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

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

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

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

In this syntax, *sqlda-name*, *header-element*, *repeating-element*, and *index* refer to standard MANTIS variables, literals, or expressions. Header elements are listed in “SQLDA header elements” on page 68 and “SQLDA repeating elements” on page 75. In the examples above, *index* refers to the sequential occurrence of the repeating element group in the SQLDA.

Allocate an SQLDA

Use the following SQLDA statement to allocate a new SQLDA.

SQLDA(*sqlda-name*) = NEW

sqlda-name

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

Format 1–18 character text expression

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

General considerations

- ◆ This statement allocates a new, empty SQLDA structure with the default number of repeating elements. The default is set at installation as one of your MANTIS Options by your Master User. Within your program you can also modify an SQLDA's size by resetting the value of SQLMAX (see the discussion of SQLMAX that follows).
- ◆ If you declare an SQLDA of the same name as one that already exists, the second SQLDA statement is ignored.
- ◆ The scope of an SQLDA is the current DO level. For example, you can have two SQLDAs of the same name on different DO levels. Within a DO level, however, you can only access SQLDAs defined for that DO level.

Example

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

Deallocate an SQLDA

Use the following SQLDA statement to deallocate an SQLDA.

SQLDA(*sqlda-name*) = QUIT

sqlda-name

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

Format Must be a valid variable name of 1–18 characters

Consideration The *sqlda-name* may be any valid MANTIS text expression or text literal.

General consideration

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

Example

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

Move data from your program into an SQLDA header element

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

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

sqlda-name

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

header-element

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

Options Only three header elements may be set: SQLN, SLQMAX, and SQLD.

SQLN SQLN and SQLMAX are the same in Rdb support. See SQLMAX below.

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

SQLMAX SQLMAX is the actual number of repeating groups in the physical SQLDA structure. The value can range from 1–300. Setting this number in your program causes the SQLDA to expand or contract by the specified number of repetitions. Once physically expanded, the space occupied by the SQLDA will never physically contract. For example, if an SQLDA 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

SQLD is the number of SELECT LIST columns, or the number of host variable MARKERS currently described in the SQLDA repeating elements. An SQLDA cannot describe both SELECT LIST columns and MARKERS at the same time. When processing a dynamic SELECT statement containing input host variable parameters, the statement must be DESCRIBED twice; first, to obtain the number and type of input parameters which must be supplied in an OPEN statement, and second, to obtain the number and type of output column results from a FETCH statement.



MANTIS will perform the DESCRIBE SELECT LIST on your behalf, using the SQLDA named in the USING clause of the FETCH statement. You are responsible for describing MARKERS into the SQLDA named in the OPEN statement's USING clause.

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

expression

Description *Required.* Supply the SQLDA variable count.

Format Standard MANTIS variable, literal, or expression

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

General considerations

- ◆ In a third-generation language like FORTRAN and COBOL, when a DESCRIBE statement is executed, if the SQLDA is too small (SQLN is less than the number of items that will be returned as a result of the DESCRIBE), SQL sets SQLD to the required number and terminates. The program must then expand the SQLDA accordingly. By contrast, MANTIS SQL Support automatically expands the SQLDA to the required size if the SQLDA is too small to accept the results of a DESCRIBE. You can check the number of occurrences after the DESCRIBE by examining the SQLD value.
- ◆ There are two other read-only header elements illustrated in “[The SQLDA structure](#)” on page 61: SQLDAID and SQLDABC. If you attempt to use a read-only element in this SQLDA statement, you will generate a fault.

Example

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

SQLDA header elements

Element	Function	How set/when used	Results	Updateable?
SQLDAID	Eyecatcher	Set by MANTIS when the SQLDA is created	"SQLDA"	No
SQLABC	Size of the SQLDA in bytes	Set by MANTIS when the SQLDA is created, or when SQLN is changed by the SQLDA statement	16+(44*SQLN)	No
SQLN / SQLMAX	Total number of host variables in SQLDA	Set using value from MANTIS options when SQLDA is allocated, can be changed by the SQLDA statement	Number of repeating groups allocated	Yes
SQLD	Current number of SELECT LIST columns or host variable MARKERS described in the SQLDA	Set as a result of a PREPARE INTO and DESCRIBE statements; can be set by the SQLDA statement, but is not necessary because MANTIS always does implicit DESCRIBE MARKERS where necessary	Number of input or output variables described in SQLDA	Yes

Move data from your program into an SQLDA repeating element

Use the following SQLDA statement to supply values for input host variables, setting the value of repeating elements.

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

sqlda-name

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

repeating-element

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

Options Three repeating elements may be set: SQLNAME, SQLIND, and SQLDATA.

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

SQLIND Contains the indicator value. The indicator value indicates whether the host variable contains a real value, or is NULL or MISSING (see “Indicator variables” on page 31). Possible indicator values and their meanings include:

Value	Meaning
< 0	The host variable data is NULL or MISSING.
> = 0	The host variable contains real values.

SQLDATA In a third-generation program, this element holds a four-byte binary address. This address is used to access the data item being transferred between the program and the database. A third-generation program must acquire space for the data item and place the space's address in this element. In MANTIS SQL Support, this element is used to automatically perform the following actions when you are transferring data into the SQLDA:

- ◆ Allocate a data area for the data item, or expand the data area if necessary.
- ◆ Set the value of SQLDATA to the address of the data area. This address is used internally by MANTIS SQL Support; your program does not need to manipulate this value.
- ◆ Move data from the MANTIS host variable into the SQLDA data area.
- ◆ 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 72. SQLLEN is set to the length of the MANTIS variable.

If you are transferring data out of the SQLDA, this element simply performs the transfer.



Note as well that the SQLDATA element may have a type of numeric or string. When you do not know the type of data you want to retrieve from the database in advance, use the SQLTYPE to determine the data type.

index

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

expression

Description *Required.* Supply an SQLVAR element value.

Format Standard MANTIS variable, literal, or expression

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

General considerations

- ◆ SQLLEN and SQLTYPE will be set to the length and data type of the MANTIS expression.
- ◆ SQLTYPE is always set to the MANTIS equivalent data type. 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 Rdb element names. Use the Rdb names as follows:

VAX_SQL	SUPRA_SQL
SQLNAME	SQLCOLNAME
SQLIND	SQLHOSTIND
SQLDATA	SQLHOSTVAR
SQLLEN	SQLCOLLENGTH
SQLTYPE	SQLCOLTYPE/SQLHOSTVARTY

- ◆ Rdb uses the SQLTYPE element to describe the data type in the Rdb/VMS database, and to receive the host variable data type from MANTIS. MANTIS will set SQLLEN and SQLTYPE to the MANTIS data type (whenever you set SQLDATA with the SQLDA statement). Previous contents of the elements (which may have been set by a previous DESCRIBE statement) may be lost.

Example

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

MANTIS SQL support data type conversion

Rdb TYPE		Description	SQLTYPE set by MANTIS	MANTIS type
VARCHAR	449	Varying string to 64K	449	TEXT / MIXED / KANJI
CHAR	453	Blank padded string	449	TEXT / MIXED / KANJI
FLOAT	481	Floating point	481	SMALL / BIG
INTEGER	497	4-byte integer	481	BIG*
SMALLINT	501	2-byte integer	481	BIG*
DECIMAL	485	Packed-decimal	485	TEXT
DATE	503	64-bit integer	449	TEXT / MIXED / KANJI
QUADWORD	505	8-byte integer	485	DECIMAL**

* MANTIS uses BIG to read integer data from the database because there may be an implied decimal point, for example, INTEGER(2).

** MANTIS uses DECIMAL to read QUADWORD column data from the database in case they contain more significant digits than a BIG variable can hold.

The SQLDA statement is used to assign either string or numeric MANTIS data to an SQLDA repeating element. MANTIS will set the SQLTYPE to 449, 481 or 485 accordingly. If the SQLDA statement is not used in this way, MANTIS will assign SQLTYPES as shown in the table above, replacing the existing values of SQLTYPE.

Move data from an SQLDA header element into your program

Use the following SQLDA function to read header elements.

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

mantis-variable

Description *Required.* Supply the name into which the SQLDA header element is to be placed.

Format Standard MANTIS variable

sqlda-name

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

header-element

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

General considerations

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

Example

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

Move data from an SQLDA repeating element into your program

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

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

mantis-variable

- Description** *Required.* Specify the destination into which the SQLDA repeating element is to be placed.
- Format** Standard MANTIS variable or array element reference

sqlda-name

- Description** *Required.* Specify the name of a previously allocated SQLDA.

repeating-element

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

index

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

General considerations

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

Example

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

SQLDA repeating elements

Element	Function	How set/when used	Results	Updateable?
SQLNAME / SQLCOLNAMES	QL column name	Set by Rdb/VMS as the result of a PREPARE INTO or DESCRIBE statement	Column name	Yes
SQLTYPE / SQLCOLTYPE	Data type in the database or the host variable data type	Set by Rdb/VMS as the result of a PREPARE INTO or DESCRIBE statement; set by MANTIS prior to EXECUTE, OPEN, FETCH statements and by the SQLDA statement	See "MANTIS SQL support data type conversion" on page 72	No
SQLLEN / SQLCOLLENGTH	Maximum number of bytes for a column in the database, or actual number of host variable bytes	Set by Rdb/VMS as the result of a PREPARE INTO or DESCRIBE statement; set by MANTIS prior to EXECUTE, OPEN, FETCH statements and by the SQLDA statement	4 or 8 for numeric columns, or maximum or actual string length	No
SQLFRAC / SQLCOLRFAC	Number of decimal places for INTEGER, SMALLINT, or QUADWORD data types	Set by Rdb/VMS as the result of a PREPARE INTO or DESCRIBE statement	Number of places specified in the CREATE TABLE statement	No
SQLIND / SQLHOSTIND	The value of the NULL or MISSING indicator variable	Set by Rdb/VMS as the result of a FETCH statement; set by the MANTIS SQLDA statement	< = -1, NULL value; otherwise, the value is defined and addressed by SQLDATA	Yes
SQLDATA / SQLHOSTVAR	Address host variable data	Set by MANTIS prior to EXECUTE, OPEN, FETCH statements, and by the MANTIS SQLDA statement	Address of MANTIS variable in the MANTIS data work area, or address of MANTIS data after being copied or converted into work buffer	Yes

A

Sample MANTIS SQL programs

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

Use the following HELP commands:

- ◆ HELP EXEC_SQL
- ◆ HELP SQL (displays a list of available Rdb/VMS help prompters)
- ◆ HELP RDB DECLARE

Each of the Rdb/VMS specific HELP prompters includes an example program that is reproduced in the EXAMPLES program library. You can access these programs by running EXAMPLES:APPLICATIONS and selecting SQL EXAMPLES from the menu.

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

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), 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
```

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, EDUCATION_LEVEL
80 .TEXT EMPLOYEE_NUMBER(6),FIRST_NAME(20), 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
```

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
```

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), 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 .SQL_TEXT=SQL_TEXT+"FIRSTNME = ?, MIDINIT = ?, LASTNAME = ?,"
210 .SQL_TEXT=SQL_TEXT+"BRTHDATE = ?, HIREDATE = ?"
220 .SQL_TEXT=SQL_TEXT+"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,"SQLMAX")=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
```

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, EDUCATION_LEVEL
80 .TEXT EMPLOYEE_NUMBER(6),FIRST_NAME(20), 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
```

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, 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
```

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
```

A dynamic delete routine

Note that using an SQLDA is not required 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
```

A dynamic query-like function

This program enables you to interactively execute SQL statements and display the column data resulting from the execution of those statements at your terminal.

```
10 ENTRY Rdb/VMS QUERY
20 .|
30 .| This example illustrates the use of dynamic SQL to
40 .| perform a QUERY-like function.
50 .|
60 .TEXT STMT(255),TEXT PARM(80), DA MARKERS, DA SELECT
70 .PROGRAM Rdb/VMS SHOW TABLE ("EXAMPLES:Rdb/VMS SHOW TABLE","CASINO")
80 .DA MARKERS="DA MARKERS"
90 .DA SELECT="DA SELECT"
100 .SQLDA(DA MARKERS)=NEW
110 .SQLDA(DA SELECT)=NEW
120 | Define SQL Error Handler - QHANDLER
130 .EXEC SQL:| WHENEVER SQLERROR DO QHANDLER
140 .END
150 | Prompt SQL statements and execute them
160 .WHILE NOT(FINISHED):
170 ..STMT="":ERROR=FALSE
180 ..SHOW "SQL>":OBTAIN STMT
190 ..IF KEY<>"ENTER"OR STMT=""
200 ...FINISHED=TRUE
210 ..ELSE
220 ...EXEC SQL:| PREPARE S1 SELECT LIST INTO DA SELECT FROM :STMT
230 ...END
240 ...IF NOT(ERROR)
250 ....EXEC SQL:| DESCRIBE S1 MARKERS INTO DA MARKERS
260 ....END
```

```
270 ....IF SQLCA("SQLCODE")=0
280 .....IF SQLDA(DA SELECT,"SQLD")>:0 | If it's a SELECT STATEMENT
290 .....DO Rdb/VMS SHOW TABLE(STMT)
300 .....ELSE
310 .....IF SQLDA(DA MARKERS,"SQLD")>0
320 .....DO SQLDA INPUT(DA MARKERS)
330 .....EXEC SQL:| EXECUTE S1 USING DESCRIPTOR DA MARKERS
340 .....END
350 .....ELSE:| No host variable parameters
360 .....EXEC SQL:| EXECUTE IMMEDIATE :STMT
370 .....END
380 .....END
390 ....END
400 ...END
410 ..END
420 .END
430 .END
440 EXIT
450 |
460 | Allow continuation after failure to execute SQL statement
470 | 480 ENTRY QHANDLER
490 .SHOW "*** SQL ERROR CODE =";SQLCA("SQLCODE")
500 .SHOW "*** SQL MESSAGE =";SQLCA("SQLERRMC")
510 .ERROR=TRUE
520 .SHOW "*** press RETURN to continue";WAIT
530 EXIT
540 |
550 ENTRY SQLDA INPUT(DA)
560 .NINPUT=SQLDA(DA,"SQLD")
570 .BIG NVAL:TEXT TYPE(15),TVAL(50)
580 .I=1
```

```
590 .WHILE I<=NINPUT
600 ..DO SQLTYPE(DA,I,TYPE)
610 ..SHOW "Input";I;"(TYPE(2))";
620 ..IF TYPE(1,1)="N"
630 ...OBTAIN NVAL:SQLDA(DA,"sqldata",I)=NVAL
640 ...SQLDA(DA,"SQLDATA",I)=NVAL
650 ..ELSE
660 ...OBTAIN TVAL:SQLDA(DA,"sqldata",I)=TVAL
670 ...SQLDA(DA,"SQLDATA",I)=TVAL
680 ..END
690 ..I=I+1
700 .END
710 EXIT
720 |
730 ENTRY SQLTYPE(DA,VARX,TYPE)
740 .TC=SQLDA(DA,"SQLTYPE",I)
750 .WHEN TC=449
760 ..TYPE="TVARCHAR"
770 .WHEN TC=453
780 ..TYPE="TCHAR"
790 .WHEN TC=481
800 ..TYPE="NFLOAT"
810 .WHEN TC=485
820 ..TYPE="NDECIMAL"
830 .WHEN TC=497
840 ..TYPE="NINTEGER"
850 .WHEN TC=501
860 ..TYPE="NSMALLINT"
870 .WHEN TC=503
880 ..TYPE="TDATE"
890 .WHEN TC=505
900 ..TYPE="NQUADWORD"
910 .END WHEN
920 EXIT
```

A dynamic column select

This program retrieves the column name, type, and length, and the first row of the column from a table specified by the user. The program uses dynamically executed statements.

```

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), "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","SQLLENGTH",COUNTER)
280 ..'AT(55),SQLDA("SQLDA1","SQLDATA",COUNTER)
290 ..COUNTER=COUNTER+1
300 .END
310 .WAIT
320 EXIT

```


B

Features not supported

The following features of SQL are not supported for Rdb/VMS:

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

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

VX is invalidly used as a host variable.

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

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

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

```

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

```

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

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

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

- ◆ MANTIS programs containing dynamic SQL statements are not portable between MANTIS SQL Support for the IBM mainframe and MANTIS SQL Support for OpenVMS and UNIX. You can, however, port programs containing static embedded SQL between the two systems.

C

Differences: MANTIS SQL support versus SQL in COBOL; MANTIS versus SQL

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

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

SQL in MANTIS SQL support versus SQL in COBOL

- ◆ You embed SQL statements in a MANTIS application program as standard MANTIS comments and delimit each SQL statement with an EXEC_SQL-END block. No MANTIS comments are permitted within the EXEC_SQL-END block. All comments within the block are considered SQL statement text.

- ◆ In the SQL WHENEVER statement:
 - The GOTO clause is replaced by a standard MANTIS DO statement, and STOP is replaced by FAULT. See “[The SQL WHENEVER statement](#)” on page 43.
 - The default for the condition SQLERROR is FAULT; in SQL in COBOL, the default is CONTINUE. See “[The SQL WHENEVER statement](#)” on page 43”.
 - WHENEVER settings may have different ranges of applicability than they would in SQL in COBOL. See “[Scope of the WHENEVER statement](#)” on page 48.
- ◆ SQLCA elements are accessed through the SQLCA statement/function rather than as items of data. See “[The SQLCA in MANTIS SQL support](#)” on page 49.
- ◆ Elements in SQLDAs are accessed through the SQLDA statement/function, rather than as items of data. See “[The scope of cursors, statements and SQLDAs](#)” on page 38.
- ◆ In a MANTIS SQL Support application, you may receive messages from three sources: the MANTIS nucleus, MANTIS SQL Support, and the database system. For detailed explanations and actions, refer to [AD/Advantage MANTIS Messages and Codes OpenVMS/UNIX](#), P39-1330.
- ◆ MANTIS SQL Support does not support an SQL INCLUDE statement, as INCLUDE denotes a preprocessor action. The SQLCA and SQLDA functions eliminate the need to INCLUDE these structures. For more information on SQLDA, see “[The scope of cursors, statements and SQLDAs](#)” on page 38; for more information on SQLCA, see “[The SQLCA in MANTIS SQL support](#)” on page 49.
- ◆ DECLARE statements are unnecessary for tables and views.

MANTIS versus SQL

- ◆ In MANTIS, quotation marks (") delimit character-string constants. In SQL, apostrophes (') delimit character-string constants.
- ◆ Permissible data type conversions between SQL and MANTIS are listed in "Data conversion between MANTIS SQL support and the Rdb/VMS database" on page 32.
- ◆ Only data type codes for MANTIS-compatible data types are returned in the SQLCOLTYPE element in the SQLDA. Valid data types are thus limited to those listed in "SQLCA elements" on page 51.

Index

I

|

description 18
in an EXEC_SQL-END block
26

A

accessing
SQLCA elements 19
SQLDAs 19
allocate, an SQLDA 63
array, using host variables in 30
auto-cursor FETCH 58

B

BIG variable 29
binding 37
blanks, using in an EXEC_SQL-
END block 27
breakpoint, and use of RESET 54

C

COBOL SQL, differences from
MANTIS SQL support 19
code sequence, for dynamic SQL
59
colons
in an EXEC_SQL-END block
27
using with host variables 30
columns, indicator variables for
31
comment character 18
comments
in an EXEC_SQL-END block
28
in SQL statements 26
COMMIT 48
failure of MANTIS COMMIT 54

using in embedded SQL
applications 53–54
connection 23. See *also* sign-on
to Rdb/VMS 39
CONTINUE 46
conversion See data conversion
cursor names See *also* real
cursor
dynamic SQL support 34
MANTIS mapping of 38
cursors, keeping open 54

D

data type conversion
error conditions 33
MANTIS SQLDA vs. VAX SQL
72
data type Conversion
MANTIS vs. Rdb/VMS 32
data types
MANTIS vs. SQL 30
VAX SQL 32
data, moving
from an SQLDA header
element into your program
73
from an SQLDA repeating
element into your program
74
from your program into an
SQLDA repeating element
69
from your program into and
SQLDA header element 65
DBNAME 51, 52
DBTYPE 22, 51, 52
deallocate, an SQLDA 64
defaults
DBTYPE 22
SQLERROR 19
WHENEVER conditons 46
disconnection
from Rdb/VMS 40
possible causes of 40
DO 45
dynamic SQL
code sequence for 59
cursor names 34
defined 57
description 20

- E**
- elements, SQLCA 51–52
 - embedding SQL statements
 - described 17
 - rules for 26–28
 - END 18
 - environment variables 20
 - error conditions, during data conversion 33
 - error messages 55
 - EXEC_SQL
 - syntax for 41–42
 - using for DBTYPE 22
 - EXEC_SQL-END block
 - examples 26–28
 - executing 37
 - rules for using 26–28
 - storage of in a MANTIS SQL support program 36
 - using colons in 27
 - using multiple blanks in 27
 - using multiple lines in 27
 - using text in 26
 - with a MANTIS statement 28
 - explicit sign-on 23
- F**
- Facility Selection Menu 21
 - failure of COMMIT 54
 - FAULT 43, 46
 - FETCH
 - auto-cursor 58
 - example 59
- G**
- GOTO, MANTIS SQL support
 - equivalent 43
- H**
- header elements
 - moving data from 73
 - moving data into 65–68
 - SQLD 66
 - SQLDA 68
 - SQLMAX 65
 - host variables
 - description of 18
 - identification of 27
 - number of 66
 - using colons in 30
 - using in a MANTIS array 30
 - using in an EXEC_SQL-END block 30
- I**
- implicit sign-on, to Rdb/VMS 23
 - indentation level 18
 - indicator variables
 - example 31
 - using in SQL statements 31
 - input host variable, description of 18
- L**
- logical names 20
 - required for Rdb/VMS sign-on 23
- M**
- main program cleanup 40
 - MANTIS RDI interface
 - description 39
 - MANTIS RDI Interface
 - description 23
 - restrictions for VAX SQL 34
 - MANTIS SQL Options 22
 - MANTIS SQL support
 - data type conversion 32, 72
 - differences from SQL in COBOL 19
 - function of 17
 - interface to VAX SQL 34
 - messages 19
 - processing of SQL statements 17
 - security 20
 - software requirements for 19
 - MANTIS SQL support programs
 - considerations for writing 36
 - rules for embedding SQL statements 26–28
 - running with nonSQL MANTIS programs 17
 - MANTIS statements
 - in SQL statements 26

- with and EXEC_SQL-END block 28
- MANTIS variables 18. *See also* host variables
- MANTIS_RDI_IF 23, 34
- mapping, of cursor names and statements 38
- Master User 21
- messages, source of in a MANTIS SQL support program 19
- multiple lines, using in an EXEC_SQL-END block 27

N

- nonSQL MANTIS programs, using with MANTIS SQL programs 17

O

- Oracle, interpretation of indicator variables 31
- output host variable 18

P

- PREPARE 34
- processing, SQL statements 17
- program cleanup *See* main program cleanup

R

- Rdb/VMS
 - connection to 39
 - data conversion 32
 - disconnection from 40
 - interpretation of indicator variables 31
 - MANTIS interface to 34
 - signing on to 23
- RDI interface *See* MANTIS RDI Interface
- real cursor 34
- repeating elements
 - moving data from 74
 - moving data into 69–72
 - SQLCOLNAME 69
 - SQLDATA 70

- SQLHOSTIND 69
- SQLHOSTVAR 70
- SQLNAME 69
- requirements
 - for MANTIS SQL support 19
 - logicals for Rdb/VMS sign-on 23
- RESET, using in embedded SQL applications 53
- ROLLBACK 48
 - using in embedded SQL statements 53–54
- rules
 - for COMMIT and RESET in embedded SQL applications 53–54
 - for embedding SQL statements in a MANTIS program 26–28
- RUN 37

S

- security, in MANTIS SQL support 20
- SELECT, example 59
- SET DBNAME 48
- sign-off *See* disconnection
- sign-on *See also* connection to Rdb/VMS 23
- software requirements, for MANTIS SQL support 19
- spaces, using in an EXEC_SQL-END block 27
- SQL statement name 38
- SQL statements
 - COMMIT 48
 - dynamic execution of 57
 - embedding in MANTIS programs 17
 - END 18
 - EXEC_SQL 18, 41
 - invalid for dynamic execution 58
 - invalid uses of 26
 - MANTIS processing of 17
 - ROLLBACK 48
 - SET DBNAME 48
 - syntax, general 18
 - using indicator variables in 31
 - WHENEVER 43–46
 - declarative vs. interpretive 47

- differences between MANTIS
 - SQL and COBOL SQL 19
- SQL Statements
 - embedding rules 26–28
- SQL\$DATABASE, supported
 - setting of 39
- SQLABC 68
- SQLCA
 - elements 51–52
 - syntax for 49–50
 - using for DBTYPE 22
- SQLCABC 51
- SQLCAID 51
- SQLCODE 51
- SQLCOLLENGTH 75
- SQLCOLNAME 69
- SQLCOLNAMES 75
- SQLCOLRFAC 75
- SQLCOLTYPE 75
- SQLD 66, 68
- SQLDA
 - accessing 19
 - allocating 63
 - deallocate 64
 - header elements
 - list of 68
 - moving data from 73
 - moving data into 65
 - SQLD 66
 - SQLMAX 65
 - name of 38
 - repeating elements
 - list of 75
 - moving data from 74
 - moving data into 69–72
 - SQLCOLNAME 69
 - SQLDATA 70
 - SQLHOSTIND 69
 - SQLHOSTVAR 70
 - SQLIND 69
 - SQLNAME 69
 - size of 67
 - structure of 61
 - valid data types 32
- SQLDAID 68
- SQLDATA 70, 75
- SQLERRDn 51
- SQLERRMC 51, 52
- SQLERRML 51
- SQLERROR 44
- SQLERP 51
- SQLEXT 51

- SQLFRAC 75
- SQLHOSTIND 69, 75
- SQLHOSTVAR 70, 75
- SQLIND 69, 75
- SQLLEN 75
- SQLMAX 65, 68
- SQLN 68
- SQLNAME 69, 75
- SQLTYPE 75
- SQLVARINC 22
- SQLWARNING 44
- SQLWARNn 51
- static SQL, description 20
- Super User *See* Master User
- SUPRA, interpretation of
 - indicator variables 31
- syntax
 - COMMIT 48
 - EXEC_SQL-END 41–42
 - ROLLBACK 48
 - SET DBNAME 48
 - SQLCA 49–50
 - WHENEVER 43–46

T

- text, rules for embedding in an
 - EXEC_SQL-END block 26

U

- updating, MANTIS user profile 22
- user Pprofile
 - enhancements to 22
- user profile
 - updating 22

V

- values, for indicator variables 31
- variables *See* MANTIS variables
 - or host variables
- VAX SQL
 - data types 32
 - MANTIS interface to 34
- vertical bar
 - description 18
 - in an EXEC_SQL-END block
 - 26

W

WHENEVER

actions

CONTINUE 46

DO 45

FAULT 46

conditions

SQLERROR 44

SQLWARNING 44

declarative vs. interpretive 47

defaults 46

syntax for 43–46

