

Oracle Rdb™

SQL Reference Manual Volume 5

Release 7.3.3.0 for HP OpenVMS Industry Standard 64 for Integrity Servers and
OpenVMS Alpha operating systems

November 2018

ORACLE®

SQL Reference Manual, Volume 5

Release 7.3.3.0 for HP OpenVMS Industry Standard 64 for Integrity Servers and OpenVMS Alpha operating systems

Copyright © 1987, 2018 Oracle Corporation. All rights reserved.

Primary Author: Rdb Engineering and Documentation group

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, the following notice is applicable:

U.S. GOVERNMENT RIGHTS Programs, software, databases, and related documentation and technical data delivered to U.S. Government customers are "commercial computer software" or "commercial technical data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, duplication, disclosure, modification, and adaptation shall be subject to the restrictions and license terms set forth in the applicable Government contract, and, to the extent applicable by the terms of the Government contract, the additional rights set forth in FAR 52.227-19, Commercial Computer Software License (December 2007). Oracle America, Inc., 500 Oracle Parkway, Redwood City, CA 94065.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle, Java, Oracle Rdb, Hot Standby, LogMiner for Rdb, Oracle SQL/Services, Oracle CODASYL DBMS, Oracle RMU, Oracle CDD/Repository, Oracle Trace, and Rdb7 are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

This software or hardware and documentation may provide access to or information on content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services.

Contents

Send Us Your Comments	xi
Preface	xiii
A Error Messages	
A.1 Types of Error Messages and Their Format	A-1
A.2 Error Message Documentation	A-3
A.3 Errors Generated When You Use SQL Statements	A-4
A.4 Identifying Precompiler and Module Language Errors	A-7
B SQL Standards	
B.1 ANSI/ISO/IEC SQL 1999 Standard	B-1
B.2 SQL:1999 Features in Rdb	B-6
B.3 Establishing SQL:1999 Semantics	B-8
C The SQL Communications Area (SQLCA) and the Message Vector	
C.1 The SQLCA	C-2
C.2 The Message Vector	C-22
C.3 Declarations of the SQLCA and the Message Vector	C-23
C.4 Using SQLCA Include Files	C-30
C.5 SQLSTATE	C-30
C.5.1 Definition of the SQLSTATE Status Parameter	C-31
C.5.2 Use of the SQLSTATE Status Parameter	C-36

D The SQL Dynamic Descriptor Areas (SQLDA and SQLDA2)

D.1	Purpose of the SQLDA	D-1
D.2	How SQL and Programs Use the SQLDA	D-3
D.3	Declaring the SQLDA	D-5
D.4	Description of Fields in the SQLDA	D-9
D.5	Parameters Associated with the SQLDA: SQLSIZE and SQLDAPTR . . .	D-14
D.6	Purpose of the SQLDA2	D-15
D.6.1	Declaring the SQLDA2	D-16
D.6.2	Description of Fields in the SQLDA2	D-19

E Logical Names Used by SQL

F Obsolete SQL Syntax

F.1	Incompatible Syntax	F-1
F.1.1	Incompatible Syntax Containing the SCHEMA Keyword	F-2
F.1.1.1	CREATE SCHEMA Meaning Incompatible	F-2
F.1.1.2	SHOW SCHEMA Meaning Incompatible	F-2
F.1.1.3	DROP SCHEMA Meaning Incompatible	F-2
F.1.2	DROP TABLE Now Restricts by Default	F-3
F.1.3	Database Handle Names Restricted to 25 Characters	F-3
F.1.4	Deprecated Default Semantics of the ORDER BY Clause	F-3
F.1.5	Change to EXTERNAL NAMES IS Clause	F-4
F.2	Deprecated Syntax	F-4
F.2.1	Command Line Qualifiers	F-6
F.2.2	Deprecated Interactive SQL Statements	F-6
F.2.3	Constraint Conformance to the ANSI/ISO SQL Standard	F-7
F.2.4	Obsolete Keywords	F-7
F.2.5	Obsolete Built-in Functions	F-8
F.3	Deprecated Logical Names	F-9
F.3.1	RDB\$CHARACTER_SET Logical Name	F-9
F.4	Reserved Words Deprecated as Identifiers	F-10
F.4.1	ANSI/ISO 1989 SQL Standard Reserved Words	F-10
F.4.2	ANSI/ISO 1992 SQL Standard Reserved Words	F-11
F.4.3	ANSI/ISO 1999 SQL Standard Reserved Words	F-13
F.4.4	Words From ANSI/ISO SQL3 Draft Standard No Longer Reserved	F-14
F.5	Punctuation Changes	F-14
F.5.1	Single Quotation Marks Required for String Literals	F-15
F.5.2	Double Quotation Marks Required for ANSI/ISO SQL Delimited Identifiers	F-15

F.5.3	Colons Required Before Host Language Variables in SQL Module Language	F-15
F.6	Suppressing Diagnostic Messages	F-15

G Oracle Database Compatibility

G.1	Oracle Database Functions	G-1
G.1.1	Optional Oracle SQL Functions	G-1
G.2	Oracle Style Outer Join	G-15
G.2.1	Outer Join Examples	G-16

H System Tables

H.1	Using Data Dictionary	H-1
H.2	Modifying System Tables	H-1
H.3	Updating Metadata	H-2
H.4	LIST OF BYTE VARYING Metadata	H-2
H.5	Read Only Access	H-3
H.6	All System Tables	H-7
H.6.1	RDB\$CATALOG_SCHEMA	H-8
H.6.2	RDB\$COLLATIONS	H-9
H.6.3	RDB\$CONSTRAINTS	H-10
H.6.3.1	RDB\$FLAGS	H-11
H.6.4	RDB\$CONSTRAINT_RELATIONS	H-12
H.6.4.1	RDB\$FLAGS	H-12
H.6.5	RDB\$DATABASE	H-12
H.6.5.1	RDB\$FLAGS	H-16
H.6.6	RDB\$FIELD_VERSIONS	H-17
H.6.6.1	RDB\$FLAGS	H-19
H.6.7	RDB\$PARAMETER_SUB_TYPE	H-19
H.6.8	RDB\$FIELD_SUB_TYPE	H-20
H.6.9	RDB\$FIELDS	H-21
H.6.9.1	RDB\$FLAGS	H-25
H.6.10	RDB\$GRANTED_PROFILES	H-25
H.6.11	RDB\$INDEX_SEGMENTS	H-26
H.6.11.0.1	RDB\$FLAGS	H-27
H.6.12	RDB\$INDICES	H-27
H.6.12.1	RDB\$FLAGS	H-29
H.6.13	RDB\$INTERRELATIONS	H-30
H.6.13.1	RDB\$USAGE	H-31
H.6.13.2	RDB\$FLAGS	H-33
H.6.14	RDB\$MODULES	H-34

H.6.15	RDB\$OBJECT_SYNONYMS	H-35
H.6.15.1	RDB\$FLAGS	H-35
H.6.16	RDB\$PARAMETERS	H-36
H.6.16.1	RDB\$FLAGS	H-36
H.6.17	RDB\$PRIVILEGES	H-37
H.6.17.1	RDB\$FLAGS	H-38
H.6.18	RDB\$PROFILES	H-39
H.6.18.1	RDB\$FLAGS	H-40
H.6.19	RDB\$QUERY_OUTLINES	H-40
H.6.19.1	RDB\$FLAGS	H-41
H.6.20	RDB\$RELATION_CONSTRAINTS	H-41
H.6.20.1	RDB\$FLAGS	H-42
H.6.20.2	RDB\$CONSTRAINT_TYPE	H-42
H.6.21	RDB\$RELATION_CONSTRAINT_FLDS	H-43
H.6.22	RDB\$RELATION_FIELDS	H-44
H.6.23	RDB\$RELATIONS	H-46
H.6.23.1	RDB\$FLAGS	H-49
H.6.24	RDB\$ROUTINES	H-50
H.6.24.1	RDB\$FLAGS	H-51
H.6.24.2	RDB\$SOURCE_LANGUAGE	H-52
H.6.25	RDB\$SEQUENCES	H-53
H.6.25.1	RDB\$FLAGS	H-54
H.6.26	RDB\$STORAGE_MAPS	H-54
H.6.26.1	RDB\$FLAGS	H-55
H.6.27	RDB\$STORAGE_MAP_AREAS	H-55
H.6.27.1	RDB\$FLAGS	H-56
H.6.28	RDB\$SYNONYMS	H-57
H.6.29	RDB\$TRIGGERS	H-59
H.6.29.1	RDB\$FLAGS	H-61
H.6.29.2	RDB\$TRIGGER_TYPE Values	H-61
H.6.30	RDB\$VIEW_RELATIONS	H-61
H.6.31	RDB\$TRIGGER_ACTIONS	H-62
H.6.31.1	RDB\$FLAGS	H-63
H.6.32	RDB\$WORKLOAD	H-63

I Information Tables

I.1	Introduction to Information Tables	I-1
I.1.1	Restrictions for Information Tables	I-5
I.2	All Information Tables	I-5

I.2.1	All_Information_Tables	I-5
I.2.1.1	RDB\$STORAGE_AREAS	I-5
I.2.1.1.1	RDB\$FLAGS	I-7
I.2.1.2	RDB\$JOURNALS	I-8
I.2.1.2.1	RDB\$FLAGS	I-9
I.2.1.3	RDB\$CACHES	I-9
I.2.1.3.1	RDB\$FLAGS	I-10
I.2.1.4	RDB\$DATABASE_ROOT	I-10
I.2.1.4.1	RDB\$FLAGS	I-13
I.2.1.5	RDB\$DATABASE_JOURNAL	I-14
I.2.1.5.1	RDB\$FLAGS	I-15
I.2.1.6	RDB\$DATABASE_USERS	I-16
I.2.1.6.1	RDB\$FLAGS	I-16
I.2.1.7	RDB\$LOGICAL_AREAS	I-17
I.2.1.7.1	RDB\$FLAGS	I-17
I.2.1.7.2	RDB\$RECORD_TYPE	I-18
I.2.1.8	RDB\$CHARACTER_SETS	I-18
I.2.1.8.1	RDB\$REPertoire	I-19
I.2.1.8.2	RDB\$FORM_OF_USE	I-20
I.2.1.8.3	RDB\$FLAGS	I-20
I.2.1.9	RDB\$NLS_CHARACTER_SETS	I-21
I.2.1.10	RDB\$SESSION_PRIVILEGES	I-21

Index

Examples

C-1	Fields in the SQLCA	C-3
C-2	Including Error Literals in a COBOL Program	C-10
C-3	Values in SQLCA after PREPARE Statement	C-14
C-4	Ada SQLCA and Message Vector Declaration	C-24
C-5	BASIC SQLCA and Message Vector Declaration	C-24
C-6	C SQLCA and Message Vector Declaration	C-26
C-7	COBOL SQLCA and Message Vector Declaration	C-27
C-8	FORTRAN SQLCA and Message Vector Declaration	C-27
C-9	Pascal SQLCA and Message Vector Declaration	C-28
C-10	PL/I SQLCA and Message Vector Declaration	C-29
C-11	Declaring SQLSTATE in a C Program	C-37
D-1	Declaration of the SQLDA in Ada	D-7
D-2	Declaration of the SQLDA in BASIC	D-7

D-3	Declaration of the SQLDA in C	D-8
D-4	Declaration of the SQLDA in PL/I	D-9
D-5	Declaration of the SQLDA2 in Ada	D-16
D-6	Declaration of the SQLDA2 in BASIC	D-17
D-7	Declaration of the SQLDA2 in C	D-18

Figures

C-1	Fields of the Message Vector	C-23
-----	------------------------------------	------

Tables

A-1	Explanation of Error Message Severity Codes	A-2
A-2	SQL Errors Generated at Run Time	A-5
B-1	Fully Supported Core SQL:1999 Features	B-3
B-2	Partially Supported Core SQL:1999 Features	B-4
B-3	Unsupported Core SQL:1999 Features	B-6
C-1	Values Returned to the SQLCODE Field	C-4
C-2	Including the Error Literals File in Programs	C-8
C-3	SQLERRD array setting by statement	C-12
C-4	SQLCA SQLERRD[0] Values	C-15
C-5	SQLSTATE Status Parameter Values—Sorted by SQLSTATE Class and Subclass	C-31
C-6	Include Files for SQLSTATE	C-37
D-1	Fields in the SQLDA	D-9
D-2	Codes for SQLTYPE Field of SQLDA and SQLDA2	D-13
D-3	Fields in the SQLDA2	D-20
D-4	Codes for Interval Qualifiers in the SQLDA2	D-26
D-5	Codes for Date-Time Data Types in the SQLDA2	D-26
D-6	Values for the SQLCHAR_SET_NAME Field	D-27
E-1	Summary of SQL Logical Names	E-1
E-2	Valid Equivalence Names for RDB\$CHARACTER_SET Logical Name	E-3
F-1	Deprecated Syntax for SQL	F-4
F-2	Obsolete SQL Keywords	F-7
G-1	Optional Oracle SQL Functions	G-2
I-1	Supported Information Tables	I-2

Send Us Your Comments

Oracle Rdb for OpenVMS

Oracle SQL Reference Manual, Release 7.3.3.0

Oracle Corporation welcomes your comments and suggestions on the quality and usefulness of this publication. Your input is an important part of the information used for revision.

- Did you find any errors?
- Is the information clearly presented?
- Do you need more information? If so, where?
- Are the examples correct? Do you need more examples?
- What features did you like most?

If you find any errors or have any other suggestions for improvement, please indicate the document title, chapter, section, and page number (if available). You can send comments to us in the following ways:

- Electronic mail: InfoRdb_US@oracle.com
- FAX — 603-897-3825 Attn: Oracle Rdb
- Postal service:
Oracle Corporation
Oracle Rdb Documentation
One Oracle Drive
Nashua, NH 03062-2804
USA

If you would like a reply, please give your name, address, telephone number, and (optionally) electronic mail address.

If you have problems with the software, please contact your local Oracle Support Services.

Preface

This manual describes the syntax and semantics of the statements and language elements for the SQL (structured query language) interface to the Oracle Rdb database software.

Intended Audience

To get the most out of this manual, you should be familiar with data processing procedures, basic database management concepts and terminology, and the OpenVMS operating system.

Operating System Information

You can find information about the versions of the operating system and optional software that are compatible with this version of Oracle Rdb in the *Oracle Rdb Installation and Configuration Guide*.

For information on the compatibility of other software products with this version of Oracle Rdb, refer to the *Oracle Rdb Release Notes*.

Contact your Oracle representative if you have questions about the compatibility of other software products with this version of Oracle Rdb.

Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit <http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info> or visit <http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs> if you are hearing impaired.

Structure

This manual is divided into five volumes. Volume 1 contains Chapter 1 through Chapter 5 and an index. Volume 2 contains Chapter 6 and an index. Volume 3 contains Chapter 7 and an index. Volume 4 contains Chapter 8 and an index. Volume 5 contains the appendixes and an index.

The index for each volume contains entries for the respective volume only and does not contain index entries from the other volumes in the set.

The following table shows the contents of the chapters and appendixes in Volumes 1, 2, 3, 4, and 5 of the *Oracle Rdb SQL Reference Manual*:

Chapter 1	Introduces SQL (structured query language) and briefly describes SQL functions. This chapter also describes conformance to the ANSI standard, how to read syntax diagrams, executable and nonexecutable statements, keywords and line terminators, and support for Multivendor Integration Architecture.
Chapter 2	Describes the language and syntax elements common to many SQL statements.
Chapter 3	Describes the syntax for the SQL module language and the SQL module processor command line.
Chapter 4	Describes the syntax of the SQL precompiler command line.
Chapter 5	Describes SQL routines.
Chapter 6	Describe in detail the syntax and semantics of the SQL statements. These chapters include descriptions of data definition statements, data manipulation statements, and interactive control commands.
Chapter 7	
Chapter 8	
Appendix A	Describes the different types of errors encountered in SQL and where they are documented.
Appendix B	Describes the SQL standards to which Oracle Rdb conforms.
Appendix C	Describes the SQL Communications Area, the message vector, and the SQLSTATE error handling mechanism.
Appendix D	Describes the SQL Descriptor Areas and how they are used in dynamic SQL programs.

Appendix E	Summarizes the logical names that SQL recognizes for special purposes.
Appendix F	Summarizes the obsolete SQL features of the current Oracle Rdb version.
Appendix G	Summarizes the SQL functions that have been added to the Oracle Rdb SQL interface for compatibility with Oracle Database SQL. This appendix also describes the SQL syntax for performing an outer join between tables.
Appendix H	Describes the Oracle Rdb system tables.
Appendix I	Describes information tables that can be used with Oracle Rdb.
Index	Index for each volume.

Related Manuals

For more information on Oracle Rdb, see the other manuals in this documentation set, especially the following:

- *Oracle Rdb Guide to Database Design and Definition*
- *Oracle Rdb7 Guide to Database Performance and Tuning*
- *Oracle Rdb Introduction to SQL*
- *Oracle Rdb Guide to SQL Programming*

Conventions

In examples, an implied carriage return occurs at the end of each line, unless otherwise noted. You must press the Return key at the end of a line of input.

Often in examples the prompts are not shown. Generally, they are shown where it is important to depict an interactive sequence exactly; otherwise, they are omitted.

The following conventions are also used in this manual:

- Vertical ellipsis points in an example mean that information not directly related to the example has been omitted.
-
-
-
- ... Horizontal ellipsis points in statements or commands mean that parts of the statement or command not directly related to the example have been omitted.

e, f, t	Index entries in the printed manual may have a lowercase e, f, or t following the page number; the e, f, or t is a reference to the example, figure, or table, respectively, on that page.
boldface text	Boldface type in text indicates a new term.
< >	Angle brackets enclose user-supplied names in syntax diagrams.
[]	Brackets enclose optional clauses from which you can choose one or none.
\$	The dollar sign represents the command language prompt. This symbol indicates that the command language interpreter is ready for input.

References to Products

The Oracle Rdb documentation set to which this manual belongs often refers to the following Oracle Corporation products by their abbreviated names:

- In this manual, Oracle Rdb refers to Oracle Rdb for OpenVMS. Version 7.3 of Oracle Rdb software is often referred to as V7.3.
- Oracle CDD/Repository software is referred to as the dictionary, the data dictionary, or the repository.
- Oracle ODBC Driver for Rdb software is referred to as the ODBC driver.
- OpenVMS I64 refers to HP OpenVMS Industry Standard 64 for Integrity Servers.
- OpenVMS means the OpenVMS I64 and OpenVMS Alpha operating systems.

A

Error Messages

This appendix describes:

- The types and format of error messages you can encounter when using SQL
- How to find and use the documentation for error messages

A.1 Types of Error Messages and Their Format

You can receive messages not only from SQL, but also from underlying software.

Messages you encounter while using SQL come from the following levels:

- The SQL interface itself. Messages generated by SQL are preceded by a facility code of SQL. For example:

```
%SQL-E-CURALROPE, Cursor K was already open
```

In programs, you can use the message vector structure in the `SQL_SIGNAL`, `SQL_GET_ERROR_TEXT`, `SQL_GET_MESSAGE_VECTOR`, and `SQL$GET_ERROR_TEXT` routines, described in Section C.2, to signal errors and return the corresponding message text.

- Common Operating System Interface (COSI) facility error messages. For example:

```
%COSI-F-NOQUAL, qualifiers not allowed - supply only verb and parameters
```

- The underlying database product. The facility code for messages generated by the underlying database depends on the database product with which you are using SQL.

Oracle Rdb messages are preceded by a facility code of RDMS. For example:

```
%RDMS-F-INVDB_NAME, invalid database name
```

Refer to the appropriate documentation for other products.

- The repository. Messages generated by the repository are preceded by a facility code of CDD. For example:

`%CDD-E-ERRSHOW, error displaying object`

Whatever the source of an error message, the format is the same. All error messages contain the following elements:

- The facility name preceded by a percent sign (%) or a hyphen (-)
- The severity code followed by a hyphen (-)

Table A-1 lists the severity codes in order of increasing severity.

- The diagnostic error message name followed by a comma (,)

This name identifies the message. In the documentation for error messages, the messages are alphabetized within each facility by diagnostic error message name.

- The diagnostic error message text

The text is a brief description of the problem. Error messages may contain string substitutions that are specific to a user's error. In the documentation for error messages, these string substitutions are delimited by angle brackets (< >) within a message. For example:

`%SQL-F-CURNOTOPE, Cursor <str> is not opened`

If you receive this message, SQL substitutes the actual string (in this case, a cursor name) for <str>.

You can suppress the display of any or all elements of an error message with the SET MESSAGE command in DCL.

Table A-1 Explanation of Error Message Severity Codes

Code	Severity	Explanation
S	Success	Indicates that your command executed successfully.
I	Information	Reports on actions taken by the software.

(continued on next page)

Table A–1 (Cont.) Explanation of Error Message Severity Codes

Code	Severity	Explanation
W	Warning	Indicates error conditions for which you may not need to take corrective action.
E	Error	Indicates conditions that are not fatal, but that must be handled or corrected.
F	Fatal	Indicates conditions that are fatal and must be handled or corrected.

A.2 Error Message Documentation

Because error messages are updated frequently, documentation is provided in the following text files:

- SQL messages:
In `SYSS$HELP:SQL$MSGnn.DOC`
where *nn* is the current version number for Oracle Rdb.
This file contains the same text as the Help Errors help topic in interactive SQL.
- RDB messages:
In `SYSS$HELP:RDB_MSGnn.DOC`
where *nn* is the current version number for Oracle Rdb.
- RDMS messages:
In `SYSS$HELP:RDMS_MSG.DOC`
- COSI messages:
In `SYSS$HELP:COSI_MSG.DOC`
- SQL/Services messages:
In `SYSS$HELP:SQLSRV$MSG.DOC`
- Repository messages:
In `SYSS$HELP:CDD_MSG.DOC`

The message documentation for all the facilities follows the same format, with messages alphabetized by message name. After the message name and text, the documentation includes an explanation and suggested user action.

The online message documentation files may be updated even if you do not install a new version of SQL. In particular, any installation of Oracle Rdb databases may replace the RDB_MSG.DOC file with one that is more up-to-date.

You can print the online message documentation files for reference. In addition, you can search the files for the message information you need.

A.3 Errors Generated When You Use SQL Statements

When you write application programs that use SQL, you must use one of the following methods to return the error messages:

- The SQLCODE parameter, which stores a value that represents the execution status of SQL statements.
- The SQLSTATE status parameter, a string of five characters, provides error handling that complies with the ANSI/ISO SQL standard. See Appendix C for more information on the SQLSTATE status parameter.
- The longword array RDB\$MESSAGE_VECTOR, which stores information about the execution status of SQL statements.
- The calls `sql_signal`, `sql_get_error_text`, and `SQL$GET_ERROR_TEXT`, which use error information returned through the RDB\$MESSAGE_VECTOR array.
- The call `sql_get_message_vector`, which retrieves information from the message vector about the status of the last SQL statement.
- The SQL statement `WHENEVER`, which provides error handling for all SQL statements that follow the `WHENEVER` statement. (However, you cannot use this statement in programs that call procedures in an SQL module.)

For more information about handling errors using SQL options, see the *Oracle Rdb Guide to SQL Programming*.

Table A-2 lists SQL statements and errors they commonly generate at run time. This is not an exhaustive list. The second column lists the error message status code and the third column lists the corresponding value of the SQLCODE field in the SQLCA. See Appendix C for more information about SQLCODE values.

Table A–2 SQL Errors Generated at Run Time

SQL Statement	Error Status Code ²	SQLCODE Value
ALTER DOMAIN	SQL\$_BAD_LENGTH	–1029
	SQL\$_BAD_SCALE	–1030
	SQL\$_NO_SUCH_FIELD	–1027
ALTER TABLE	RDB\$_DEADLOCK	–913
	RDB\$_INTEG_FAIL	–1001
	RDB\$_LOCK_CONFLICT	–1003
	RDB\$_NO_PRIV	–1028
	RDB\$_READ_ONLY_REL	–1031
	RDB\$_READ_ONLY_TRANS	–817
	RDB\$_READ_ONLY_VIEW	–1031
	RDB\$_REQ_NO_TRANS	Not available ¹
	SQL\$_BAD_LENGTH	–1029
	SQL\$_BAD_SCALE	–1030
	SQL\$_COEXISTS	–1023
	SQL\$_FLDNOTDEF	–1024
	SQL\$_FLDNOTINREL	–1024
	SQL\$_NO_SUCH_FIELD	–1027
	RDB\$_REQ_WRONG_DB	–1020
ATTACH	SQL\$_CURNOTOPE	–501
COMMIT	RDB\$_DEADLOCK	–913
	RDB\$_INTEG_FAIL	–1001
	RDB\$_LOCK_CONFLICT	–1003
	SQL\$_NO_TXNOUT	–1005
CREATE DOMAIN	SQL\$_FIELD_EXISTS	–1026
CREATE VIEW	RDB\$_DEADLOCK	–913
	RDB\$_LOCK_CONFLICT	–1003
	SQL\$_NO_SUCH_FIELD	–1027

¹No specific numeric value. Check the SQLCODE for negative values.

²–1 is a general error SQLCODE value that does not correspond to any specific error. Use sql_signal or sql_get_error_text to return a meaningful error.

(continued on next page)

Table A–2 (Cont.) SQL Errors Generated at Run Time

SQL Statement	Error Status Code ²	SQLCODE Value
DELETE	SQL\$_REL_EXISTS	–1025
	RDB\$_DEADLOCK	–913
	RDB\$_INTEG_FAIL	–1001
	RDB\$_LOCK_CONFLICT	–1003
DELETE . . . WHERE	RDB\$_DEADLOCK	–913
CURRENT OF . . .	RDB\$_INTEG_FAIL	–1001
FETCH	SQL\$_CURNOTOPE	–501 / –507
	SQL\$_FETNOTDON	–508
	RDB\$_DEADLOCK	–913
	RDB\$_LOCK_CONFLICT	–1003
INSERT	RDB\$_STREAM_EOF	100
	SQL\$_CURNOTOPE	–501
	SQL\$_NULLNOIND	–305
	RDB\$_ARITH_EXCEPT	–304
	RDB\$_DEADLOCK	–913
	RDB\$_INTEG_FAIL	–1001
	RDB\$_LOCK_CONFLICT	–1003
	RDB\$_NO_DUP	–803
	RDB\$_NO_PRIV	–1028
	RDB\$_NOT_VALID	–1002
	RDB\$_OBSOLETE_METADATA	–1032
	RDB\$_READ_ONLY_REL	–1031
	RDB\$_READ_ONLY_TRANS	–817
	RDB\$_READ_ONLY_VIEW	–1031
	RDB\$_REQ_NO_TRANS	Not available ¹
	RDB\$_REQ_WRONG_DB	–1020
	RDB\$_UNRES_REL	–1033

¹No specific numeric value. Check the SQLCODE for negative values.

²–1 is a general error SQLCODE value that does not correspond to any specific error. Use sql_signal or sql_get_error_text to return a meaningful error.

(continued on next page)

Table A–2 (Cont.) SQL Errors Generated at Run Time

SQL Statement	Error Status Code ²	SQLCODE Value
OPEN	RDB\$_DEADLOCK	–913
	RDB\$_LOCK_CONFLICT	–1003
ROLLBACK	SQL\$ _NO_TXNOUT	–1005
SET TRANSACTION	RDB\$_DEADLOCK	–913
	RDB\$_LOCK_CONFLICT	–1003
	SQL\$ _BAD_TXN_STATE	–1004
singleton SELECT	RDB\$_DEADLOCK	–913
	RDB\$_LOCK_CONFLICT	–1003
	SQL\$ _NULLNOIND	–305
	SQL\$ _SELMORVAL	–811
UPDATE	RDB\$_DEADLOCK	–913
	RDB\$ _INTEG_FAIL	–1001
	RDB\$ _LOCK_CONFLICT	–1003
	RDB\$ _NO_DUP	–803
	RDB\$ _NOT_VALID	–1002
	RDB\$ _READ_ONLY_REL	–1031
UPDATE . . . WHERE	RDB\$_DEADLOCK	–913
CURRENT OF . . .	RDB\$ _INTEG_FAIL	–1001
	RDB\$ _LOCK_CONFLICT	–1003
	RDB\$ _NO_DUP	–803
	RDB\$ _NOT_VALID	–1002
	SQL\$ _CURNOTOPE	–501
	SQL\$ _FETNOTDON	–508

²–1 is a general error SQLCODE value that does not correspond to any specific error. Use sql_signal or sql_get_error_text to return a meaningful error.

A.4 Identifying Precompiler and Module Language Errors

The SQL precompiler and the SQL module language processor let you identify (flag) syntax that is not ANSI/ISO SQL standard. See Chapter 3 and Chapter 4 for more information.

Error messages for SQL precompilers and SQL module language are flagged in the following way:

```
EXEC SQL SELECT SUM(DISTINCT QTY), AVG(DISTINCT QTY)    /* multiple distincts*/
%SQL-I-NONSTADIS, (1) The standard only permits one DISTINCT in a select expression
      INTO :int1, :int2 FROM D.SP;                      /*      in a query      */
```

B

SQL Standards

This appendix describes the SQL standards to which Oracle Rdb conforms.

B.1 ANSI/ISO/IEC SQL 1999 Standard

- The SQL interface to Oracle Rdb is referred to as SQL. This interface is the Oracle Rdb implementation of the SQL standard commonly referred to as the ANSI/ISO SQL standard or SQL99.
- The new SQL standard adopted in 1999 consists of the following five parts:
 - ANSI/ISO/IEC 9075-1:1999, Information technology - Database language - SQL - Part 1: Framework (SQL/Framework)
 - ANSI/ISO/IEC 9075-2:1999, Information technology - Database language - SQL - Part 2: Foundation (SQL/Foundation)
 - ANSI/ISO/IEC 9075-3:1999, Information technology - Database language - SQL - Part 3: Call-Level Interface (SQL/CLI)
 - ANSI/ISO/IEC 9075-4:1999, Information technology - Database language - SQL - Part 4: Persistent Stored Modules (SQL/PSM)
 - ANSI/ISO/IEC 9075-5:1999, Information technology - Database language - SQL - Part 5: Host Language Bindings (SQL/Bindings)

In general, the Oracle Rdb documentation refers to this standard as SQL:1999. SQL:1999 supersedes the SQL92 standard.

The minimal conformance level for SQL:1999 is known as Core. Core SQL:1999 is a superset of the SQL92 Entry Level specification. Oracle Rdb is broadly compatible with the SQL:1999 Core specification. However, a small number of SQL:1999 Core features are not currently implemented in Oracle Rdb or differ from the Oracle Rdb implementation. Oracle Corporation is committed to fully supporting SQL:1999 Core functionality in a future release, while providing upward compatibility for existing applications.

Additionally, Oracle Rdb also complies to most of the ANSI/ISO/IEC 9075-4:1999 (Persistent Stored Modules) portion of the standard.

The following functionality described by SQL:1999 CORE is not currently available in Oracle Rdb:

- **SQL99 flagger**
The flagger would alert the programmer to extensions to the SQL:1999 SQL database language.
- **Basic Information Schema, and Documentation Schema**
A set of tables and views that describe the database definitions, similar in content to the Rdb system tables.
- **TIME and TIMESTAMP precision up to 6 fractional seconds**
Oracle Rdb currently supports a maximum fractional second precision of 2.
- **CREATE TYPE**
The CREATE TYPE statement in the SQL:1999 CORE allows a user to define a typed name, similar to a domain, but with strong typing rules.
- **REVOKE . . . { RESTRICT | CASCADE }**
These variations to REVOKE requires that a check be performed during protection updates so that privilege changes do not effect the correct execution of existing procedures and functions.

You can obtain a copy of ANSI standards from the following address:

American National Standards Institute
11 West 42nd Street
New York, NY 10036
USA
Telephone: 212.642.4900
FAX: 212.398.0023

Or from their web site:

<http://webstore.ansi.org/ansidocstore/default.asp>

A subset of ANSI standards, including the SQL standard, are X3 or NCITS standards. You can obtain these from the National Committee for Information Technology Standards (NCITS) at:

<http://www.cssinfo.com/ncitsquate.html>

The Core SQL:1999 features that Oracle Rdb fully supports are listed in Table B–1.

Table B–1 Fully Supported Core SQL:1999 Features

Feature ID	Feature
E011	Numeric data types
E021	Character data types
E031	Identifiers
E051	Basic query specification
E061	Basic predicates and search conditions
E071	Basic query expressions
E081	Basic privileges
E091	Set functions
E101	Basic data manipulation
E111	Single row SELECT statement
E121	Basic cursor support
E131	Null value support (nulls in lieu of values)
E141	Basic integrity constraints
E151	Basic transaction support
E152	Basic SET TRANSACTION statement
E153	Updatable queries with subqueries
E161	SQL comments using leading double minus
E171	SQLSTATE support
E182	Module language
F041	Basic joined table
F081	UNION and EXCEPT in views
F131	Grouped operations
F181	Multiple module support
F201	CAST function
F221	Explicit defaults
F261	CASE expression

(continued on next page)

Table B–1 (Cont.) Fully Supported Core SQL:1999 Features

Feature ID	Feature
F311	Schema definition statement
F471	Scalar subquery values
F481	Expanded NULL predicate

Core SQL:1999 features that Oracle Rdb partially supports are listed in Table B–2.

Table B–2 Partially Supported Core SQL:1999 Features

Feature ID	Feature	Partial Support
F031	Basic schema	<p>Oracle Rdb fully supports the following manipulation subfeatures:</p> <ul style="list-style-type: none"> • F031-01, Clause 11, "Schema definition and manipulation": Selected facilities as indicated by the subfeatures of this Feature • F031-02, CREATE VIEW statement • F031-03, GRANT statement • F031-04, ALTER TABLE statement: ADD COLUMN clause • F031-13, DROP TABLE statement: RESTRICT clause • F031-16, DROP VIEW statement: RESTRICT clause <p>Oracle Rdb does not support the following subfeature:</p> <ul style="list-style-type: none"> • F031-19, REVOKE statement: RESTRICT clause

(continued on next page)

Table B–2 (Cont.) Partially Supported Core SQL:1999 Features

Feature ID	Feature	Partial Support
F051	Basic date and time	<p>Oracle Rdb fully supports the following subfeatures:</p> <ul style="list-style-type: none">• F051-01, DATE data type (including support of DATE literal)• F051-02, TIME data type (including support of TIME literal) with fractional seconds precision of at least 0.• F051-03, TIMESTAMP data type (including support of TIMESTAMP literal) with the maximum fractional seconds precision of 2• F051-04, comparison predicate on DATE, TIME, and TIMESTAMP data types• F051-05, explicit CAST between datetime types and character types• F051-06, CURRENT_DATE• F051-07, LOCALTIME• F051-08, LOCALTIMESTAMP <p>Oracle Rdb does not support the following subfeature:</p> <ul style="list-style-type: none">• F051-03, fractional seconds precision greater than 2

(continued on next page)

Table B–2 (Cont.) Partially Supported Core SQL:1999 Features

Feature ID	Feature	Partial Support
T321	Basic SQL-invoked routines	<p>Oracle Rdb fully supports the following subfeatures:</p> <ul style="list-style-type: none"> • T321-01, user-defined functions with no overloading • T321-02, user-defined stored procedures with no overloading • T321-03, function invocation • T321-04, CALL statement • T321-05, RETURN statement <p>Oracle Rdb does not support the following subfeatures:</p> <ul style="list-style-type: none"> • T321-06, ROUTINES view • T321-07, PARAMETERS view

The Core SQL:1999 features that Oracle Rdb does not support are listed in Table B–3.

Table B–3 Unsupported Core SQL:1999 Features

Feature ID	Feature
F021	Basic information schema; you can get this information from the Oracle Rdb system tables
F501	Features and conformance views
F812	Basic flagging; Oracle Rdb's SQL flagger only shows up through SQL92
S011	Distinct data types

B.2 SQL:1999 Features in Rdb

Oracle Rdb includes the following SQL:1999 features to SQL:

- AND CHAIN clause for COMMIT and ROLLBACK
- LOCALTIME, LOCALTIMESTAMP, ABS functions
- START TRANSACTION statement
- ITERATE loop control statement

- WHILE looping statement using revised SQL:1999 syntax
- REPEAT looping statement
- Searched CASE statement
- DETERMINISTIC, and NOT DETERMINISTIC attributes
These clauses replace NOT VARIANT and VARIANT attributes, respectively.
- RETURNS NULL ON NULL INPUT and CALLED ON NULL INPUT clauses for functions
- Support for module global variables which can be accessed by all routines in a module.
- DEFAULT VALUES clause for INSERT
- DEFAULT keyword for INSERT and UPDATE
- Full SIGNAL statement syntax
- BETWEEN SYMMETRIC predicate support
- USER and ROLE support including the GRANT/REVOKE enhancements
- INITIALLY IMMEDIATE and INITIALLY DEFERRED clauses for constraints
- UNIQUE predicate
- TABLE query specification
This is a shorthand for SELECT * FROM
- DISTINCT keyword for UNION
- FOREIGN KEY reference semantics
The columns listed by the REFERENCES clause can be in a different order to that of the matching PRIMARY KEY or UNIQUE constraint. Requires SQL99 dialect.
- ALTER MODULE, ALTER PROCEDURE and ALTER FUNCTION statements
- EXCEPT DISTINCT operator
- INTERSECT DISTINCT operator
- CORRESPONDING clause for UNION, EXCEPT and INTERSECT operators

- VAR_POP, VAR_SAMP, STDDEV_POP, STDDEV_SAMP statistical operators
- FILTER modifier for statistical functions

B.3 Establishing SQL:1999 Semantics

The following commands can be used to establish the SQL:1999 database language standard semantics:

- SET DIALECT
- SET QUOTING RULES
- SET KEYWORD RULES
- SET DEFAULT DATE FORMAT

For example:

```
SQL> SET DIALECT 'SQL99';
```

In most cases, the semantics of the SQL99 dialect are the same as SQL92. As new features are added, these may have different semantics in these two dialects.

The following command displays the current settings for this connection:

```
SQL> SHOW CONNECTION <connectionname>
```

For example:

```
SQL> show connection rdb$default_connection
Connection: RDB$DEFAULT_CONNECTION
Default alias is RDB$DBHANDLE
Default catalog name is RDB$CATALOG
Default schema name is SMITHI
Dialect: SQL99
Default character unit: CHARACTERS
Keyword Rules: SQL99
View Rules: ANSI/ISO
Default DATE type: DATE ANSI
Quoting Rules: ANSI/ISO
Optimization Level: DEFAULT
Hold Cursors default: WITH HOLD PRESERVE NONE
Quiet commit mode: ON
Compound transactions mode: EXTERNAL
Default character set is DEC_MCS
National character set is DEC_MCS
Identifier character set is DEC_MCS
Literal character set is DEC_MCS
Display character set is UNSPECIFIED
```

The session variables DIALECT, DATE_FORMAT, QUOTING_RULES, and KEYWORD_RULES can also return the string 'SQL99'.

For example:

```
SQL> declare :a, :b, :c, :d char(10);
SQL> get environment (session)
cont>      :a = DIALECT,
cont>      :b = DATE_FORMAT,
cont>      :c = QUOTING_RULES,
cont>      :d = KEYWORD_RULES;
SQL> print :a, :b, :c, :d;
  A      B      C      D
SQL99    SQL99    SQL99    SQL99
```

The SQL Communications Area (SQLCA) and the Message Vector

The SQLCA and message vector are two separate host structures that SQL declares when it precompiles an INCLUDE SQLCA statement.

Both the SQLCA and the message vector provide ways of handling errors:

- The SQLCA is a collection of parameters that SQL uses to provide information about the execution of SQL statements to application programs. The SQLCODE parameter in the SQLCA shows if a statement was successful and, for some errors, the particular error when a statement is not successful.

To illustrate how the SQLCA works in applications, interactive SQL displays its contents when you issue the SHOW SQLCA statement.

- The message vector is also a collection of parameters that SQL updates after it executes a statement. It lets programs check if a statement was successful, but provides more detail than the SQLCA about the type of error if a statement is not successful. The message vector, for example, provides a way to access any follow-on messages in addition to those containing the facility code RDB or SQL.

You can use the following steps to examine the message vector:

- Assign any value to the logical name SQL\$KEEP_PREP_FILES.
- Precompile any program that contains the line “EXEC SQL INCLUDE SQLCA”. (You can use the programs in the sample directory.)
- Examine the generated host language program.

SQL updates the contents of the SQLCA and the message vector after completion of every executable SQL statement (nonexecutable statements are the DECLARE, WHENEVER, and INCLUDE statements).

You do not have to use the `INCLUDE SQLCA` statement in programs. However, if you do not, you must explicitly declare the `SQLCODE` parameter to receive values from SQL. `SQLCODE` is explicitly declared as an unscaled, signed longword integer.

`SQLCODE` is a deprecated feature of the ANSI/ISO SQL standard and is replaced by `SQLSTATE`. To comply with the ANSI/ISO SQL standard, you should explicitly declare either `SQLCODE` or, preferably, `SQLSTATE` instead of using the `INCLUDE SQLCA` statement. `SQLCA` (and the `INCLUDE SQLCA` statement) is not part of the ANSI/ISO SQL standard. If you declare `SQLCODE` or `SQLSTATE` but use the `INCLUDE SQLCA` statement, SQL uses the `SQLCA`.

Programs that do not use the `INCLUDE SQLCA` statement will not have the message vector declared by the precompiler. Such programs must explicitly declare the message vector if they:

- Use the `RDB$LU_STATUS` field of the message vector in their error checking
- Use system calls such as `SYSPUTMSG`

The message vector is not part of the ANSI/ISO SQL standard.

When the `SQLCA` structure is explicitly declared by a program, SQL does not update the `SQLERRD` fields. If you want the `SQLERRD` fields updated, include the `SQLCA` definitions in the program using the `EXEC SQL INCLUDE SQLCA` statement.

Section C.1 and Section C.2 describe the `SQLCA` and the message vector in more detail. Section C.3 shows the declarations SQL makes for them in different host language programs.

C.1 The SQLCA

The only fields of interest in the `SQLCA` are the `SQLCODE` field and the second through sixth elements of the `SQLERRD` array.

Example C-1 shows the interactive SQL display for the `SQLCA` after the “attempt to fetch past end of stream” error.

Example C–1 Fields in the SQLCA

```
SQL> SHOW SQLCA
SQLCA:
      SQLCAID:      SQLCA      SQLCABC:      128
      SQLCODE:      100
      SQLEERRD:      [0]: 0
                      [1]: 0
                      [2]: 0
                      [3]: 0
                      [4]: 0
                      [5]: 0
      SQLWARN0:      0          SQLWARN1:      0          SQLWARN2:      0
      SQLWARN3:      0          SQLWARN4:      0          SQLWARN5:      0
      SQLWARN6:      0          SQLWARN7:      0
      SQLSTATE:      02000
```

SQLSTATE is not part of the SQLCA, although it appears in the display.

The remainder of this section describes the fields of the SQLCA.

Fields of the SQLCA

SQLCAID

An 8-character field whose value is always the character string SQLCA. The FORTRAN SQLCA does not include this field.

SQLCABC

An integer field whose value is always the length, in bytes, of the SQLCA. The value is always 128. The FORTRAN SQLCA does not include this field.

SQLCODE

An integer field whose value indicates the error status returned by the most recently executed SQL statement. A positive value other than 100 indicates a warning, a negative value indicates an error, and a zero indicates successful execution.

Table C–1 shows the possible numeric and literal values that SQL returns to the SQLCODE field and explains the meaning of the values.

Table C–1 Values Returned to the SQLCODE Field

Numeric Value	Literal Value	Meaning
Success Status Code		
0	SQLCODE_SUCCESS	Statement completed successfully.
Warning Status Codes		
100	SQLCODE_EOS	SELECT statement or cursor came to the end of stream.
1003	SQLCODE_ELIM_NULL ¹	Null value was eliminated in a set function.
1004	SQLCODE_TRUN_RTRV ¹	String truncated during assignment. This occurs only during data retrieval.
Error Status Codes		
–1	SQLCODE_RDBERR	Oracle Rdb returned an error. The value of –1 is a general error SQLCODE value returned by any error not corresponding to the other values in this table. Use sql_signal or sql_get_error_text to return a meaningful error.
–304	SQLCODE_OUTOFRAN	Value is out of range for a host variable.
–305	SQLCODE_NULLNOIND	Tried to store a null value into a host language variable with no indicator variable.
–306	SQLCODE_STR_DAT_TRUNC ¹	String data, right truncation.
–307	SQLCODE_INV_DATETIME	Date-time format is invalid.
–501	SQLCODE_CURNOTOPE	Cursor is not open.
–502	SQLCODE_CURALROPE	Cursor is already open.
–507	SQLCODE_UDCURNOPE	Cursor in an UPDATE or DELETE operation is not opened.
–508	SQLCODE_UDCURNPOS	Cursor in an UPDATE or DELETE operation is not positioned on a row.
–509	SQLCODE_UDCURDEL	Cursor in an UPDATE or DELETE operation is positioned on a deleted row.

¹Only the SQL92 and SQL99 dialects return this value.

(continued on next page)

Table C–1 (Cont.) Values Returned to the SQLCODE Field

Numeric Value	Literal Value	Meaning
Error Status Codes		
–803	SQLCODE_NO_DUP	Updating would cause duplication on a unique index.
–811	SQLCODE_SELMORVAL	The result of a singleton select returned more than one value.
–817	SQLCODE_ROTXXN	Attempt to update from a read-only transaction.
–880	SQLCODE_SVPTINVSPEC	Savepoint is invalid.
–881	SQLCODE_SVPTEXIST	Savepoint already exists.
–882	SQLCODE_SVPTNOEXIST	Savepoint does not exist.
–913	SQLCODE_DEADLOCK	Request failed due to resource deadlock.
–1001	SQLCODE_INTEG_FAIL	Constraint failed.
–1002	SQLCODE_NOT_VALID	Valid-if failed.
–1003	SQLCODE_LOCK_CONFLICT	NO WAIT request failed because resource was locked.
–1004	SQLCODE_BAD_TXN_STATE	Invalid transaction state—the transaction already started.
–1005	SQLCODE_NO_TXN	No transaction active.
–1006	SQLCODE_BAD_VERSION	Version of the underlying system does not support a feature that this query uses.
–1007	SQLCODE_TRIG_ERROR	Trigger forced an error.
–1008	SQLCODE_NOIMPTXN	No implicit distributed transaction outstanding.
–1009	SQLCODE_DISTIDERR	Distributed transaction ID error.
–1010	SQLCODE_BAD_CTX_VER	Version field in the context structure is defined incorrectly.
–1011	SQLCODE_BAD_CTX_TYPE	Type field in the context structure is defined incorrectly.
–1012	SQLCODE_BAD_CTX_LEN	Length field in the context structure is defined incorrectly.
–1013	SQLCODE_BASROWDEL	Row that contains the list has been deleted.

(continued on next page)

Table C–1 (Cont.) Values Returned to the SQLCODE Field

Numeric Value	Literal Value	Meaning
Error Status Codes		
–1014	SQLCODE_DIFFDEFINV	Invoker of the module is not the same as the definer (the user who compiled the module).
–1015	SQLCODE_STMTNOTPRE	Dynamic statement is not prepared.
–1016	SQLCODE_NOSUCHCONN	Connection does not exist.
–1017	SQLCODE_CONNAMEXI	Connection name already exists.
–1018	SQLCODE_DBENVSYNERR	Database environment specification contains a syntax error.
–1019	SQLCODE_DBSPECSYNERR	Database specification contains a syntax error.
–1020	SQLCODE_ATTACHERR	Error attaching to the database.
–1021	SQLCODE_NOSUCHALIAS	Alias is not known.
–1022	SQLCODE_ALIASINUSE	Alias is already declared.
–1023	SQLCODE_COEXISTS	Column already exists in the table.
–1024	SQLCODE_COLNOTDEF	Column not defined in the table.
–1025	SQLCODE_TBLEXISTS	Table already exists in the database or schema.
–1026	SQLCODE_DOMEXISTS	Domain already exists in the database or schema.
–1027	SQLCODE_DOMNOTDEF	Domain is not defined in the database or schema.
–1028	SQLCODE_NO_PRIV	No privilege for attempted operation.
–1029	SQLCODE_BAD_LENGTH	Negative length specified for a column.
–1030	SQLCODE_BAD_SCALE	Negative scale specified for a column.
–1031	SQLCODE_RO_TABLE	Attempt to update a read-only table.
–1032	SQLCODE_OBSMETADATA	Metadata no longer exists.
–1033	SQLCODE_UNRES_REL	Table is not reserved in the transaction.
–1034	SQLCODE_CASENOTFND	Case not found; WHEN not specified.
–1035	SQLCODE_CHKOPT_VIOL	Integer failure with check option.

(continued on next page)

Table C–1 (Cont.) Values Returned to the SQLCODE Field

Numeric Value	Literal Value	Meaning
Error Status Codes		
–1036	SQLCODE_UNTERM_C_STR	Unterminated C string.
–1037	SQLCODE_INDIC_OVERFLOW	Indicator overflow.
–1038	SQLCODE_INV_PARAM_VAL	Invalid parameter value.
–1039	SQLCODE_NULL_ELIMIN	Null eliminated in the set function.
–1040	SQLCODE_INV_ESC_SEQ	Invalid escape sequence.
–1041	SQLCODE_RELNOTDEF	Table not defined in the database or schema.
–1045	SQLCODE_INV_INTERVAL	Invalid interval format.
–1046	SQLCODE_INV_FRACSEC	Time, Timestamp or interval has too many fractional digits.
–1047	SQLCODE_INV_INTLEAD	Interval leading field is too large.
–1048	SQLCODE_INC_CSET	Incompatible character set.
–1049	SQLCODE_DATA_CVT_ERROR	Data conversion error.
–20111	SQLCODE_SVPTBADLOC	Cannot use SAVEPOINT from function or trigger.

Programs can use the literal values to check for success, the end of record stream warnings, or specific errors. Your program can check for particular error codes and execute different sets of error-handling statements depending upon the error code returned. However, because the values in Table C–1 do not reflect all the possible errors or warnings, your program should check for *any* negative value.

SQL inserts the RDB message vector (see Section C.2) along with the SQLCA structure when it executes an SQL statement.

Also, string truncation conditions are only reported when the dialect is set to SQL92 or SQL99 prior to a database attach in interactive SQL or when your application is compiled. For example:

```

SQL> SET DIALECT 'SQL99';
SQL> ATTACH 'FILENAME mf_personnel';
SQL> DECLARE :ln CHAR(10);
SQL> SELECT last_name INTO :ln FROM employees WHERE employee_id = '00164';
%RDB-I-TRUN_RTRV, string truncated during assignment to a variable or parameter
SQL> SHOW SQLCA
SQLCA:
      SQLCAID:      SQLCA      SQLCABC:      128
      SQLCODE:      1004
      SQLERRD:      [0]: 0
                   [1]: 0
                   [2]: 1
                   [3]: 0
                   [4]: 0
                   [5]: 0
      SQLWARN0:      0      SQLWARN1:      0      SQLWARN2:      0
      SQLWARN3:      0      SQLWARN4:      0      SQLWARN5:      0
      SQLWARN6:      0      SQLWARN7:      0
      SQLSTATE:      01004
%RDB-I-TRUN_RTRV, string truncated during assignment to a variable or parameter

```

For each language, SQL provides a file that contains the declarations of all the error literals shown in Table C–1. You can include this file in precompiled SQL and module language programs.

Table C–2 shows how to include this file in your program.

Table C–2 Including the Error Literals File in Programs

Precompiled or Module Language	Declaration
Ada	with SQL_SQLCODE; with SQL_SQLDA; with SQL_SQLDA2; ¹
BASIC	%INCLUDE "sys\$library:sql_literals.bas"
C	#include "sys\$library:sql_literals.h"
COBOL	COPY 'SYSS\$LIBRARY:SQL_LITERALS'
FORTRAN	INCLUDE 'SYSS\$LIBRARY:SQL_LITERALS.FOR'
Pascal	%include 'sys\$library:sql_literals.pas'
PL/I	%INCLUDE 'sys\$library:sql_literals.pli';
¹ You must compile the Ada package, SYSS\$LIBRARY:SQL_LITERALS.ADA, before you use it in a program. Only declare SQL_SQLDA and SQL_SQLDA2 when you use dynamic SQL.	

In addition to the error literals, the file contains declarations for the SQLTYPE field in the SQLDA. See Appendix D for information about the SQLTYPE field.

Example C-2 shows how to include the error literals file in a COBOL program.

Example C-2 Including Error Literals in a COBOL Program

```
IDENTIFICATION DIVISION.
PROGRAM-ID. LITERAL-TESTS.
*
* This program tests the use of symbolic literals for SQLCODE and
* SQLDA_DATATYPE. All the literal definitions are part of a file that
* is used with the COPY command.
*
DATA DIVISION.
WORKING-STORAGE SECTION.
COPY SQL_LITERALS.
EXEC SQL INCLUDE SQLCA END-EXEC.
01 CDE          PIC X(5).
01 DISP_SQLCODE PIC S9(9) DISPLAY SIGN LEADING SEPARATE.
01 GETERRVARS.
    02 error-buffer-len      PIC S9(9) COMP VALUE 132.
    02 error-msg-len        PIC S9(9) COMP.
    02 error-buffer         PIC X(132).

exec sql whenever sqlerror continue end-exec.

PROCEDURE DIVISION.
*
* test for sqlcode -501 SQLCODE_CURNOTOPE
*
    exec sql declare A cursor for
        select college_code from colleges
        where college_name like 'D%' order by 1
    end-exec.
    exec sql fetch A into :CDE end-exec.
    if sqlcode = SQLCODE_CURNOTOPE
    then
        MOVE sqlcode to DISP_SQLCODE
        DISPLAY "SQLCODE after attempt to fetch is ", DISP_SQLCODE
        CALL "sql_get_error_text" USING BY REFERENCE error-buffer,
                                         BY VALUE error-buffer-len,
                                         BY REFERENCE error-msg-len.
        DISPLAY BUFFER(1:error-msg-len)
    end-if.
    exec sql close A end-exec.
*
```

(continued on next page)

Example C–2 (Cont.) Including Error Literals in a COBOL Program

```
* test for SQLCODE 0 SQLCODE_SUCCESS
*
      exec sql
      insert into employees (employee_id, last_name, sex)
        values ('00999','Jones','M')
      end-exec.
      if sqlcode = SQLCODE_SUCCESS
      then
        MOVE sqlcode to DISP_SQLCODE
        DISPLAY "SQLCODE after insert is ", DISP_SQLCODE
        CALL "sql_get_error_text" USING BY REFERENCE error-buffer,
                                      BY VALUE error-buffer-len,
                                      BY REFERENCE error-msg-len.
        DISPLAY BUFFER(1:error-msg-len)
      end-if.

      EXEC SQL ROLLBACK END-EXEC.
      STOP RUN.
```

SQLERRM

The SQLERRM is a structure containing two fields: a word field called SQLERRML and a 70-character field called SQLERRMC.

SQLERRD[x]

The SQLERRD is an array of 6 integer values. When displayed by the SHOW SQLCA statement, these elements are numbered 0 through 5 and this same convention is followed in this section. The contents of this array is dependent on the successful execution of a SQL statement. These statements and its affect on the SQLERRD are described in Table C–3. The values in the SQLERRD are undefined for all other statements and after an error.

The SQLCA field SQLERRD[0] is updated with the statement type by the PREPARE and EXECUTE IMMEDIATE statements for all dialects. Numeric codes which describe the statement type are listed in the table Table C–4.

Table C–3 SQLERRD array setting by statement

SQL Statement	Description
DELETE	The DELETE statement updates the third element (SQLERRD[2]) with the number of rows deleted by the statement.
DESCRIBE	The DESCRIBE statement updates the second element (SQLERRD[1]) with the following values: 0: the statement is any SQL statement except a SELECT statement or CALL statement. 1: the statement is a SELECT statement. 2: the statement is a CALL statement.
FETCH (list cursor)	The FETCH statement updates the second element (SQLERRD[1]) with the segment size in octets. The third element (SQLERRD[2]) with the number of the segment on which the cursor is currently positioned. If you fetch a list element that is longer than the target you specify, the fetched element will be truncated and the sixth element (SQLERRD[5]) of the SQLERRD array will be set to the length of the untruncated segment. If no truncation occurs, the sixth element is set to zero.
FETCH (table cursor)	The FETCH statement updates the third element (SQLERRD[2]) with the number of the row on which the cursor is currently positioned.
INSERT	The INSERT statement updates the third element (SQLERRD[2]) with the number of rows stored by the statement.
OPEN (list cursor)	The OPEN statement for a list cursor updates the second element (SQLERRD[1]) with the length of the longest actual segment for this column value. The fourth element (SQLERRD[3]) with number of segments for this column value. The fifth and sixth elements (SQLERRD[4,5]) are treated as a single BIGINT (quadword) containing the total octets in all segments for this column value.
OPEN (table cursor)	The OPEN statement for a table cursor updates the third element (SQLERRD[2]) with the estimated result table cardinality. The fourth element (SQLERRD[3]) with the estimated I/O operations.

(continued on next page)

Table C–3 (Cont.) SQLERRD array setting by statement

SQL Statement	Description
PREPARE	The PREPARE statement updates the first element (SQLERRD[0]) with the statement type. The numeric codes, along with the corresponding symbolic names and SQL statements are listed in Table C–4. The third ¹ element (SQLERRD[2]) will be the count of output parameters, and the fourth ¹ element (SQLERRD[3]) will be the count of input parameters. The values may be zero if there are no output or input parameters in the statement, and CALL parameters of INOUT mode will appear in both the input and output count.
REPLACE	The REPLACE statement updates the third element (SQLERRD[2]) with the number of rows stored by the statement.
UPDATE	The UPDATE statement updates the third element (SQLERRD[2]) with the number of rows modified by the statement.
SELECT	The SELECT statement updates the third element (SQLERRD[2]) with the number of rows in the result table formed by the SELECT statement. Note: The SQLERRD[2] field is not updated for dynamic SELECT statements.

¹ The SQLCA was not updated by PREPARE prior to Rdb Release 7.1.3 so Oracle recommends that the SQLERRD[2] and SQLERRD[3] values be set to a known value that can never normally be seen (such as -1) prior to the PREPARE call. If after the PREPARE these values remain -1 then the application must estimate the counts by examining the SQLDA.

SQLWARNx

A series of 1-character fields, numbered from 0 through 7.

If the statement being prepared is a SELECT statement containing an INTO clause, then the SQLCA field SQLWARN6 will contain the character "I". Such Singleton SELECT statements can be executed without using an cursor.

If the statement being prepared is a compound statement (BEGIN ... END), then the SQLCA field SQLWARN6 will contain the character "B".

If the statement being dynamically executed was a compound statement, or a CALL statement then it is possible that a COMMIT or ROLLBACK was executed as part of the procedure body. In such cases, the SQLCA field SQLWARN6 will contain the character "Y". Further if a transaction was activated prior to the statement, then SQLWARN7 will contain the character "Y". This will indicate that any cursors not declared as being preserved across

COMMIT or ROLLBACK using the ON HOLD clause or statement will have been closed.

The following simple program shows the effect of the PREPARE statement on SQLCA. This program prepares a compound statement which requires both input and output parameters.

Example C-3 Values in SQLCA after PREPARE Statement

```
#include <stdio.h>
#include <sql_rdb_headers.h>

exec sql
    declare alias filename 'db$:mf_personnel';

exec sql
    include SQLCA;

char * s1 = "begin insert into work_status values (?, ?, ?);\n
            select count(*) into ? from work_status; end";

void main ()
{
    int i;
    SQLCA.SQLERRD[2] = SQLCA.SQLERRD[3] = -1;
    exec sql
        prepare stmt from :s1;
    if (SQLCA.SQLCODE != 0) sql_signal ();
    printf( "SQLCA:\n  SQLCODE:    %9d\n", SQLCA.SQLCODE);
    for (i = 0; i < 6; i++)
        printf( "  SQLERRD[%d]: %9d\n", i, SQLCA.SQLERRD[i]);
    printf( "  SQLWARN0: %s\n", SQLCA.SQLWARN.SQLWARN0);
    printf( "  SQLWARN1: %s\n", SQLCA.SQLWARN.SQLWARN1);
    printf( "  SQLWARN2: %s\n", SQLCA.SQLWARN.SQLWARN2);
    printf( "  SQLWARN3: %s\n", SQLCA.SQLWARN.SQLWARN3);
    printf( "  SQLWARN4: %s\n", SQLCA.SQLWARN.SQLWARN4);
    printf( "  SQLWARN5: %s\n", SQLCA.SQLWARN.SQLWARN5);
    printf( "  SQLWARN6: %s\n", SQLCA.SQLWARN.SQLWARN6);
}
```

The results displayed by the program show that PREPARE has reported 3 input arguments (SQLERRD[3]) and 1 output argument (SQLERRD[2]) in the statement. The field SQLWARN6 also indicates that this is a compound statement.

(continued on next page)

Example C–3 (Cont.) Values in SQLCA after PREPARE Statement

```
SQLCA:
  SQLCODE:          0
  SQLERRD[0]:       0
  SQLERRD[1]:       0
  SQLERRD[2]:       1
  SQLERRD[3]:       3
  SQLERRD[4]:       0
  SQLERRD[5]:       0
  SQLWARN0:
  SQLWARN1:
  SQLWARN2:
  SQLWARN3:
  SQLWARN4:
  SQLWARN5:
  SQLWARN6: B
```

Table C–4 SQLCA SQLERRD[0] Values

Symbolic Name†	Value	SQL Statement
	0	Statement is unknown
SQL_K_OCTRDB_CONNECT	-1	Rdb Connect
SQL_K_OCTRDB_ATTACH	-2	Rdb Attach
SQL_K_OCTRDB_DISCONNECT	-3	Rdb Disconnect
SQL_K_OCTRDB_CREATE_MODULE	-4	Rdb Create Module
SQL_K_OCTRDB_ALTER_MODULE	-5	Rdb Alter Module
SQL_K_OCTRDB_DROP_MODULE	-6	Rdb Drop Module
SQL_K_OCTRDB_CREATE_DOMAIN	-7	Rdb Create Domain
SQL_K_OCTRDB_ALTER_DOMAIN	-8	Rdb Alter Domain

†The positive values are defined for compatibility with Oracle 10g. Not all statements are supported by Oracle Rdb therefore not all values will appear in the SQLCA. Negative values are Oracle Rdb specific values.

(continued on next page)

Table C–4 (Cont.) SQLCA SQLERRD[0] Values

Symbolic Name†	Value	SQL Statement
SQL_K_OCTRDB_DROP_DOMAIN	-9	Rdb Drop Domain
SQL_K_OCTRDB_CREATE_CATALOG	-10	Rdb Create Catalog
SQL_K_OCTRDB_ALTER_CATALOG	-11	Rdb Alter Catalog
SQL_K_OCTRDB_DROP_CATALOG	-12	Rdb Drop Catalog
SQL_K_OCTRDB_ALTER_SCHEMA	-13	Rdb Alter Schema
SQL_K_OCTRDB_DROP_SCHEMA	-14	Rdb Drop Schema
SQL_K_OCTRDB_SET_SESSION	-15	Rdb Set Session Authorization
SQL_K_OCTCTB	1	create table
SQL_K_OCTINS	2	insert
SQL_K_OCTSEL	3	select
SQL_K_OCTCCL	4	create cluster
SQL_K_OCTACL	5	alter cluster
SQL_K_OCTUPD	6	update
SQL_K_OCTDEL	7	delete
SQL_K_OCTDCL	8	drop cluster
SQL_K_OCTCIX	9	create index
SQL_K_OCTDIX	10	drop index
SQL_K_OCTAIX	11	alter index
SQL_K_OCTDTB	12	drop table
SQL_K_OCTCSQ	13	create sequence
SQL_K_OCTASQ	14	alter sequence
SQL_K_OCTATB	15	alter table
SQL_K_OCTDSQ	16	drop sequence

†The positive values are defined for compatibility with Oracle 10g. Not all statements are supported by Oracle Rdb therefore not all values will appear in the SQLCA. Negative values are Oracle Rdb specific values.

(continued on next page)

Table C–4 (Cont.) SQLCA SQLERRD[0] Values

Symbolic Name†	Value	SQL Statement
SQL_K_OCTGRA	17	grant
SQL_K_OCTREV	18	revoke
SQL_K_OCTCSY	19	create synonym
SQL_K_OCTDSY	20	drop synonym
SQL_K_OCTCVW	21	create view
SQL_K_OCTDVW	22	drop view
SQL_K_OCTVIX	23	validate index
SQL_K_OCTCPR	24	create procedure
SQL_K_OCTAPR	25	alter procedure
SQL_K_OCTLTB	26	lock table
SQL_K_OCTNOP	27	no operation
SQL_K_OCTRNM	28	rename
SQL_K_OCTCMT	29	comment
SQL_K_OCTAUD	30	audit
SQL_K_OCTNOA	31	noaudit
SQL_K_OCTCED	32	create database link
SQL_K_OCTDED	33	drop database link
SQL_K_OCTCDB	34	create database
SQL_K_OCTADB	35	alter database
SQL_K_OCTCRS	36	create rollback segment
SQL_K_OCTARS	37	alter rollback segment
SQL_K_OCTDRS	38	drop rollback segment
SQL_K_OCTCTS	39	create tablespace

†The positive values are defined for compatibility with Oracle 10g. Not all statements are supported by Oracle Rdb therefore not all values will appear in the SQLCA. Negative values are Oracle Rdb specific values.

(continued on next page)

Table C–4 (Cont.) SQLCA SQLERRD[0] Values

Symbolic Name†	Value	SQL Statement
SQL_K_OCTATS	40	alter tablespace
SQL_K_OCTDTS	41	drop tablespace
SQL_K_OCTASE	42	alter session
SQL_K_OCTAUR	43	alter user
SQL_K_OCTCWK	44	commit
SQL_K_OCTROL	45	rollback
SQL_K_OCTSPT	46	savepoint
SQL_K_OCTPLS	47	pl/sql execute
SQL_K_OCTSET	48	set transaction
SQL_K_OCTASY	49	alter system switch log
SQL_K_OCTXPL	50	explain
SQL_K_OCTCUS	51	create user
SQL_K_OCTCRO	52	create role
SQL_K_OCTDUS	53	drop user
SQL_K_OCTDRO	54	drop role
SQL_K_OCTSER	55	set role
SQL_K_OCTCSC	56	create schema
SQL_K_OCTCCF	57	create control file
SQL_K_OCTATR	58	Alter tracing
SQL_K_OCTCTG	59	create trigger
SQL_K_OCTATG	60	alter trigger
SQL_K_OCTDTG	61	drop trigger
SQL_K_OCTANT	62	analyze table
SQL_K_OCTANI	63	analyze index
SQL_K_OCTANC	64	analyze cluster
SQL_K_OCTCPF	65	create profile

†The positive values are defined for compatibility with Oracle 10g. Not all statements are supported by Oracle Rdb therefore not all values will appear in the SQLCA. Negative values are Oracle Rdb specific values.

(continued on next page)

Table C–4 (Cont.) SQLCA SQLERRD[0] Values

Symbolic Name†	Value	SQL Statement
SQL_K_OCTDPF	66	drop profile
SQL_K_OCTAPF	67	alter profile
SQL_K_OCTDPR	68	drop procedure
SQL_K_OCTARC	70	alter resource cost
SQL_K_OCTCSL	71	create snapshot log
SQL_K_OCTASL	72	alter snapshot log
SQL_K_OCTDSL	73	drop snapshot log
SQL_K_OCTCSN	74	create snapshot
SQL_K_OCTASN	75	alter snapshot
SQL_K_OCTDSN	76	drop snapshot
SQL_K_OCTCTY	77	create type
SQL_K_OCTDTY	78	drop type
SQL_K_OCTARO	79	alter role
SQL_K_OCTATY	80	alter type
SQL_K_OCTCYB	81	create type body
SQL_K_OCTAYB	82	alter type body
SQL_K_OCTDYB	83	drop type body
SQL_K_OCTDLB	84	drop library
SQL_K_OCTTTB	85	truncate table
SQL_K_OCTTCL	86	truncate cluster
SQL_K_OCTCBM	87	create bitmapfile
SQL_K_OCTAVW	88	alter view
SQL_K_OCTDBM	89	drop bitmapfile

†The positive values are defined for compatibility with Oracle 10g. Not all statements are supported by Oracle Rdb therefore not all values will appear in the SQLCA. Negative values are Oracle Rdb specific values.

(continued on next page)

Table C–4 (Cont.) SQLCA SQLERRD[0] Values

Symbolic Name†	Value	SQL Statement
SQL_K_OCTSCO	90	set constraints
SQL_K_OCTCFN	91	create function
SQL_K_OCTAFN	92	alter function
SQL_K_OCTDFN	93	drop function
SQL_K_OCTCPK	94	create package
SQL_K_OCTAPK	95	alter package
SQL_K_OCTDPK	96	drop package
SQL_K_OCTCPB	97	create package body
SQL_K_OCTAPB	98	alter package body
SQL_K_OCTDPB	99	drop package body
SQL_K_OCTCDR	157	create directory
SQL_K_OCTDDR	158	drop directory
SQL_K_OCTCLB	159	create library
SQL_K_OCTCJV	160	create java
SQL_K_OCTAJV	161	alter java
SQL_K_OCTDJV	162	drop java
SQL_K_OCTCOP	163	create operator
SQL_K_OCTCIT	164	create indextype
SQL_K_OCTDIT	165	drop indextype
SQL_K_OCTAIT	166	reserver for alter indextype
SQL_K_OCTDOP	167	drop operator
SQL_K_OCTAST	168	associate statistics
SQL_K_OCTDST	169	disassociate statistics

†The positive values are defined for compatibility with Oracle 10g. Not all statements are supported by Oracle Rdb therefore not all values will appear in the SQLCA. Negative values are Oracle Rdb specific values.

(continued on next page)

Table C–4 (Cont.) SQLCA SQLERRD[0] Values

Symbolic Name†	Value	SQL Statement
SQL_K_OCTCAL	170	call method
SQL_K_OCTCSM	171	create summary
SQL_K_OCTASM	172	alter summary
SQL_K_OCTDSM	173	drop summary
SQL_K_OCTCDM	174	create dimension
SQL_K_OCTADM	175	alter dimension
SQL_K_OCTDDM	176	drop dimension
SQL_K_OCTCCT	177	create context
SQL_K_OCTDCT	178	drop context
SQL_K_OCTASO	179	alter outline
SQL_K_OCTCSO	180	create outline
SQL_K_OCTDSO	181	drop outline
SQL_K_OCTAOP	183	alter operator
SQL_K_OCTCEP	184	create encryption profile
SQL_K_OCTAEP	185	alter encryption profile
SQL_K_OCTDEP	186	drop encryption profile
SQL_K_OCTCSP	187	create spfile from pfile
SQL_K_OCTCPS	188	create pfile from spfile
SQL_K_OCTUPS	189	merge
SQL_K_OCTCPW	190	change password
SQL_K_OCTUJI	191	update join index

†The positive values are defined for compatibility with Oracle 10g. Not all statements are supported by Oracle Rdb therefore not all values will appear in the SQLCA. Negative values are Oracle Rdb specific values.

(continued on next page)

Table C–4 (Cont.) SQLCA SQLERRD[0] Values

Symbolic Name†	Value	SQL Statement
SQL_K_OCTASYN	192	alter synonym
SQL_K_OCTADG	193	alter disk group
SQL_K_OCTCDG	194	create disk group
SQL_K_OCTDDG	195	drop disk group
SQL_K_OCTALB	196	alter library
SQL_K_OCTPRB	197	purge user recyclebin
SQL_K_OCTPDB	198	purge dba recyclebin
SQL_K_OCTPTS	199	purge tablespace
SQL_K_OCTPTB	200	purge table
SQL_K_OCTPIX	201	purge index
SQL_K_OCTUDP	202	undrop object
SQL_K_OCTDDB	203	drop database
SQL_K_OCTFBD	204	flashback database
SQL_K_OCTFBT	205	flashback table

†The positive values are defined for compatibility with Oracle 10g. Not all statements are supported by Oracle Rdb therefore not all values will appear in the SQLCA. Negative values are Oracle Rdb specific values.

C.2 The Message Vector

When SQL precompiles a program, it declares a host structure for the message vector immediately following the SQLCA. It calls the structure `RDB$MESSAGE_VECTOR`.

Programs most often use the message vector in two ways:

- By checking the message vector field `RDB$LU_STATUS` for the return status value from the last SQL statement. The program can either check the low-order bit of that field (successful if set) or use the entire field to determine the specific return status value.

- By using the message vector in the `sql_signal` and `sql_get_error_text` routines:
 - The `sql_signal` routine uses the message vector to signal the error to the OpenVMS condition handler.
 - The `sql_get_error_text` routine puts the message text corresponding to the return status value in the message vector into a buffer the program specifies.
- For more information about `sql_signal` and `sql_get_error_text`, see Chapter 5.

Figure C–1 summarizes the fields of the message vector.

Figure C–1 Fields of the Message Vector

RDB\$MESSAGE_VECTOR	
RDB\$LU_NUM_ARGUMENTS	Number of arguments in the vector
RDB\$LU_STATUS	Number corresponding to return status for the condition
RDB\$ALU_ARGUMENTS	An array containing information about FAO arguments and follow-on messages related to the primary message, if any
RDB\$LU_ARGUMENTS [1]	Number of FAO arguments to primary message
.	Pointer to FAO arguments, if any
.	Return status for follow-on message, if any
.	Number of FAO arguments, for follow-on message, if any

C.3 Declarations of the SQLCA and the Message Vector

This section shows the SQLCA and message vector declarations for the host languages supported by the SQL precompiler and module processor.

Example C–4 shows the Ada SQLCA and message vector declaration.

Example C-4 Ada SQLCA and Message Vector Declaration

```
Package SQL_ADA_CURSOR is
TYPE SQL_TYPE_1 IS NEW STRING(1..6);
  type SQLERRM_REC is
    record
      SQLERRML : short_integer;
      SQLERRMC : string (1..70);
    end record;

  type SQLERRD_ARRAY is array (1..6) of integer;

type SQLCA is
  record
    SQLCAID : string (1..8) := "SQLCA  ";
    SQLABC : integer := 128;
    SQLCODE : integer;
    SQLERRM : sqlerrm_rec;
    SQLERRD : sqlerrd_array;
    SQLWARN0 : character := ' ';
    SQLWARN1 : character := ' ';
    SQLWARN2 : character := ' ';
    SQLWARN3 : character := ' ';
    SQLWARN4 : character := ' ';
    SQLWARN5 : character := ' ';
    SQLWARN6 : character := ' ';
    SQLWARN7 : character := ' ';
    SQLEXT : string (1..8) := "      ";
  end record;

RDB_MESSAGE_VECTOR : SYSTEM.UNSIGNED_LONGWORD_ARRAY(1..20);
pragma PSECT_OBJECT(RDB_MESSAGE_VECTOR, "RDB$MESSAGE_VECTOR");
```

Example C-5 shows the BASIC SQLCA and message vector declaration.

Example C-5 BASIC SQLCA and Message Vector Declaration

```
RECORD SQLCA_REC
  string SQLCAID = 8
  long SQLCABC
  long SQLCODE
  GROUP SQLERRM
    word SQLERRML
    string SQLERRMC = 70
  END GROUP SQLERRM
  long SQLERRD(5)
  string SQLWARN0 = 1
  string SQLWARN1 = 1
```

(continued on next page)

Example C-5 (Cont.) BASIC SQLCA and Message Vector Declaration

```
      string SQLWARN2 = 1
      string SQLWARN3 = 1
      string SQLWARN4 = 1
      string SQLWARN5 = 1
      string SQLWARN6 = 1
      string SQLWARN7 = 1
      string SQLEXT = 8
END RECORD SQLCA_REC

DECLARE SQLCA_REC SQLCA

RECORD RDB$MESSAGE_VECTOR_REC
      long RDB$LU_NUM_ARGUMENTS
      long RDB$LU_STATUS
      GROUP RDB$ALU_ARGUMENTS(17) ! Arrays in BASIC are always relative
          long RDB$LU_ARGUMENT ! to 0. There are 18 array elements.
      END GROUP RDB$ALU_ARGUMENTS
END RECORD RDB$MESSAGE_VECTOR_REC

COMMON (RDB$MESSAGE_VECTOR) &
      RDB$MESSAGE_VECTOR_REC RDB$MESSAGE_VECTOR
```

Example C-6 shows the C SQLCA and message vector declaration.

Example C-6 C SQLCA and Message Vector Declaration

```
struct
{
    char SQLCAID[8];
    int SQLCABC;
    int SQLCODE;
    struct {
        short SQLERRML;
        char SQLERRMC[70];
    } SQLERRM;
    int SQLERRD[6];
    struct {
        char SQLWARN0[1];
        char SQLWARN1[1];
        char SQLWARN2[1];
        char SQLWARN3[1];
        char SQLWARN4[1];
        char SQLWARN5[1];
        char SQLWARN6[1];
        char SQLWARN7[1];
    } SQLWARN;
    char SQLEXT[8];
} SQLCA = {
    {'S','Q','L','C','A',' ',' ',' '},
    128, 0,
    {0, ""},
    {0,0,0,0,0,0},
    {"", "", "", "", "", "", "", ""},
    "" };

extern
struct Rdb$MESSAGE_VECTOR_str
RDB$MESSAGE_VECTOR;
```

Example C-7 shows the COBOL SQLCA and message vector declaration.

Example C-7 COBOL SQLCA and Message Vector Declaration

```

01      SQLCA      GLOBAL.
          02      SQLCAID PIC X(8) VALUE IS "SQLCA  ".
          02      SQLCABC PIC S9(9) COMP VALUE IS 128.
          02      SQLCODE PIC S9(9) COMP.
          02      SQLERRM.
              03      SQLERRML PIC S9(4) COMP VALUE IS 0.
              03      SQLERRMC PIC X(70).
          02      SQLERRD PIC S9(9) COMP OCCURS 6 TIMES.
          02      SQLWARN.
              03      SQLWARN0 PIC X.
              03      SQLWARN1 PIC X.
              03      SQLWARN2 PIC X.
              03      SQLWARN3 PIC X.
              03      SQLWARN4 PIC X.
              03      SQLWARN5 PIC X.
              03      SQLWARN6 PIC X.
              03      SQLWARN7 PIC X.
          02      SQLEXT PIC X(8).

01 Rdb$MESSAGE_VECTOR EXTERNAL GLOBAL.
          03 Rdb$LU_NUM_ARGUMENTS      PIC S9(9) COMP.
          03 Rdb$LU_STATUS              PIC S9(9) COMP.
          03 Rdb$ALU_ARGUMENTS          OCCURS 18 TIMES.
              05 Rdb$LU_ARGUMENTS      PIC S9(9) COMP.

```

Example C-8 shows the FORTRAN SQLCA and message vector declaration.

Example C-8 FORTRAN SQLCA and Message Vector Declaration

```

CHARACTER*1 SQLCA (128)
INTEGER*4  SQLCOD
EQUIVALENCE (SQLCOD, SQLCA(13))
INTEGER*2  SQLTXL
EQUIVALENCE (SQLTXL, SQLCA(17))
CHARACTER*70 SQLTXT
EQUIVALENCE (SQLTXT, SQLCA(19))
INTEGER*4  SQLERR(1:6)
EQUIVALENCE (SQLERR, SQLCA(89))
CHARACTER*1 SQLWRN(0:7)
EQUIVALENCE (SQLWRN, SQLCA(113))

```

(continued on next page)

Example C-8 (Cont.) FORTRAN SQLCA and Message Vector Declaration

```
INTEGER*4 Rdb$MESSAGE_VECTOR(20), Rdb$LU_NUM_ARGUMENTS
INTEGER*4 Rdb$LU_STATUS, Rdb$ALU_ARGUMENTS(18)
COMMON /Rdb$MESSAGE_VECTOR/ Rdb$MESSAGE_VECTOR
EQUIVALENCE ( Rdb$MESSAGE_VECTOR(1), Rdb$LU_NUM_ARGUMENTS)
EQUIVALENCE ( Rdb$MESSAGE_VECTOR(2), Rdb$LU_STATUS)
EQUIVALENCE ( Rdb$MESSAGE_VECTOR(3), Rdb$ALU_ARGUMENTS )
```

Example C-9 shows the Pascal SQLCA and message vector declaration.

Example C-9 Pascal SQLCA and Message Vector Declaration

```
TYPE
  RDB$LU_ARGUMENTS = [HIDDEN] INTEGER;
  RDB$ALU_ARGUMENTS_ARRAY = [HIDDEN] ARRAY [1..18] OF RDB$LU_ARGUMENTS;
  RDB$MESSAGE_VECTOR_REC = [HIDDEN] RECORD
    RDB$LU_NUM_ARGUMENTS      : INTEGER;
    RDB$LU_STATUS             : INTEGER;
    RDB$ALU_ARGUMENTS         : RDB$ALU_ARGUMENTS_ARRAY;
  END;
VAR
  RDB$MESSAGE_VECTOR : [HIDDEN, common(rdb$message_vector) ]
  RDB$MESSAGE_VECTOR_REC;
TYPE
  SQL$SQLCA_REC = [HIDDEN] RECORD
    SQLCAID : PACKED ARRAY [1..8] OF CHAR;
    SQLCABC : INTEGER;
    SQLCODE : INTEGER;
    SQLERRM : RECORD
      SQLERRML : SQL$SMALLINT;
      SQLERRMC : PACKED ARRAY [1..70] OF CHAR;
    END;
  END;
```

(continued on next page)

Example C-9 (Cont.) Pascal SQLCA and Message Vector Declaration

```
SQLERRD : ARRAY [1..6] OF INTEGER;
SQLWARN : RECORD
    SQLWARN0 : CHAR;
    SQLWARN1 : CHAR;
    SQLWARN2 : CHAR;
    SQLWARN3 : CHAR;
    SQLWARN4 : CHAR;
    SQLWARN5 : CHAR;
    SQLWARN6 : CHAR;
    SQLWARN7 : CHAR;
END;
SQLEXT : PACKED ARRAY [1..8] OF CHAR;
END;
VAR
    RDB$DBHANDLE : [HIDDEN] INTEGER;
    SQLCA : [HIDDEN] SQL$SQLCA_REC;
```

Example C-10 shows the PL/I SQLCA and message vector declaration.

Example C-10 PL/I SQLCA and Message Vector Declaration

```
DCL 1 SQLCA STATIC ,
    2 SQLCAID character(8) INITIAL('SQLCA '),
    2 SQLCABC fixed binary(31) INITIAL(128),
    2 SQLCODE fixed binary(31),
    2 SQLERRM ,
    3 SQLERRML fixed binary(15) INITIAL(0),
    3 SQLERRMC character(70),
    2 SQLERRD (1:6) fixed binary(31),
    2 SQLWARN ,
    3 SQLWARN0 character(1),
    3 SQLWARN1 character(1),
    3 SQLWARN2 character(1),
    3 SQLWARN3 character(1),
    3 SQLWARN4 character(1),
    3 SQLWARN5 character(1),
    3 SQLWARN6 character(1),
    3 SQLWARN7 character(1),
    2 SQLEXT character(8);
```

(continued on next page)

Example C–10 (Cont.) PL/I SQLCA and Message Vector Declaration

```
DCL 1 Rdb$MESSAGE_VECTOR EXTERNAL,  
    2 Rdb$LU_NUM_ARGUMENTS FIXED BINARY(31),  
    2 Rdb$LU_STATUS FIXED BINARY(31),  
    2 Rdb$ALU_ARGUMENTS (18),  
    3 Rdb$LU_ARGUMENTS FIXED BINARY (31);
```

C.4 Using SQLCA Include Files

Use of the SQLCA include files such as the SQL_SQLCA.H file for C, are intended for use with the host language files only. That is, only *.C should be included in that file. Precompiled files (*.SC files) should use the EXEC SQL INCLUDE SQLCA embedded SQL command in the declaration section of the module. In this way the precompiler can properly define the structure to be used by the related SQL generated code.

Remember that the SQLCA is always scoped at the module level, unlike the SQLCODE or SQLSTATE variables which may be routine specific.

C.5 SQLSTATE

SQL defines a set of status parameters that can be part of the parameter list for a procedure definition in a nonstored module. They are SQLSTATE, SQLCODE, and SQLCA. An SQL procedure is required to contain at least one of these status parameters in its parameter list. All status parameters are implicitly output parameters.

The purpose of these status parameters is to return the status of each SQL statement that is executed. Each status parameter gives information that allows you to determine whether the statement completed execution or an exception has occurred. These status parameters differ in the amount of diagnostic information they supply, when an exception occurs as follows:

- **SQLCODE**—This is the original SQL error handling mechanism. It is an integer value. SQLCODE differentiates among errors (negative numbers), warnings (positive numbers), successful completion (0), and a special code of 100, which means no data. SQLCODE is a deprecated feature of the ANSI/ISO SQL standard.
- **SQLCA**—This is an extension of the SQLCODE error handling mechanism. It contains other context information that supplements the SQLCODE value. SQLCA is not part of the ANSI/ISO SQL standard. However, many databases such as DB2 and Oracle Database have defined proprietary semantics and syntax to implement it.

- **SQLSTATE**—This is the error handling mechanism for the ANSI/ISO SQL standard. The SQLSTATE value is a character string that is associated with diagnostic information.

This section covers the following SQLSTATE topics:

- Definition of the SQLSTATE status parameter
- Use of the SQLSTATE status parameter

C.5.1 Definition of the SQLSTATE Status Parameter

The value returned in an SQLSTATE status parameter is a string of five characters. It comprises a two-character class value followed by a three-character subclass value. Each class value corresponds to an execution condition such as success, connection exception, or data exception. Each subclass corresponds to a subset of its execution condition. For example, connection exceptions are differentiated by “connection name in use”, “connection not open”, and “connection failure” categories. A subclass of 000 means there is no subcondition.

Table C–5 shows the SQLSTATE values that SQL has defined with its corresponding execution condition. The SQLSTATE classes beginning with either the characters *R* or *S* are Oracle Rdb-specific SQLSTATE values.

Table C–5 SQLSTATE Status Parameter Values—Sorted by SQLSTATE Class and Subclass

Class /Subclass	Condition	Subcondition
00000	Successful completion	<i>No subcondition</i>
01000	Warning	<i>No subcondition</i>
01003		Null value eliminated in aggregate function
01004		String data, right truncation
02000	No data	<i>No subcondition</i>
08002	Connection exception	Connection name in use

(continued on next page)

Table C–5 (Cont.) SQLSTATE Status Parameter Values—Sorted by SQLSTATE Class and Subclass

Class /Subclass	Condition	Subcondition
08003		Connection does not exist
08006		Connection failure
09000	Trigger action exception	<i>No subcondition</i>
20000	Case not found for case statement	<i>No subcondition</i>
21000	Singleton select returned more than one value	<i>No subcondition</i>
22001	Data exception	String data, right truncation
22002		Null value, no indicator parameter
22003		Numeric value out of range
22004		Null value not allowed
22005		Error in assignment
22006		Invalid fetch orientation
22007		Invalid date-time format
22008		Datetime field overflow
22009		Invalid time displacement value
22010		Invalid indicator parameter value
22011		Substring error
22012		Division by zero
22015		Datetime field overflow

(continued on next page)

Table C–5 (Cont.) SQLSTATE Status Parameter Values—Sorted by SQLSTATE Class and Subclass

Class /Subclass	Condition	Subcondition
22018		Invalid character value for cast
22019		Invalid escape character
22020		Invalid limit value
22021		Character not in repertoire
22022		Indicator overflow
22023		Invalid parameter value
22024		C string not terminated
22025		Invalid escape sequence
22027		Trim error
2201B		Invalid regular expression
2200F		Zero length character string
23000	Integrity constraint violation	<i>No subcondition</i>
24000	Invalid cursor state	<i>No subcondition</i>
25000	Invalid transaction state	<i>No subcondition</i>
25001		Active SQL transaction
25006		Read-only SQL transaction
26000	Invalid SQL statement identifier	<i>No subcondition</i>
2F000	SQL routine exception	<i>No subcondition</i>
2F005		Function did not execute return statement

(continued on next page)

Table C–5 (Cont.) SQLSTATE Status Parameter Values—Sorted by SQLSTATE Class and Subclass

Class /Subclass	Condition	Subcondition
30000	Invalid SQL statement	<i>No subcondition</i>
31000	Invalid target specification value	<i>No subcondition</i>
32000	Invalid constraint mode state	<i>No subcondition</i>
33000	Invalid SQL descriptor name	<i>No subcondition</i>
34000	Invalid cursor name	<i>No subcondition</i>
35000	Invalid condition number	<i>No subcondition</i>
37000	Database specification syntax error	<i>No subcondition</i>
38000	External procedure exception	<i>No subcondition</i>
39000	External procedure call exception	<i>No subcondition</i>
39001		Invalid SQLSTATE returned
3B000	Savepoint exception	<i>No subcondition</i>
3B001		Savepoint exception, Invalid specification
3B002		Savepoint exception, Too many savepoints
3B503		Savepoint exception, Cannot use from function or trigger
3C000	Ambiguous cursor name	<i>No subcondition</i>
3E000	Invalid catalog name	<i>No subcondition</i>
3F000	Invalid schema name	<i>No subcondition</i>
42000	Syntax error or access rule violation	<i>No subcondition</i>
44000	With check option violation	<i>No subcondition</i>
R1001 ¹	Lock error exception	Deadlock encountered

¹Oracle Rdb specific SQLSTATE code

(continued on next page)

Table C–5 (Cont.) SQLSTATE Status Parameter Values—Sorted by SQLSTATE Class and Subclass

Class /Subclass	Condition	Subcondition
R1002 ¹		Lock conflict
R2000 ¹	Duplicate value not allowed in index	<i>No subcondition</i>
R3000 ²	Trigger forced an ERROR statement	<i>No subcondition</i>
R4000 ¹	Distributed transaction identification error	<i>No subcondition</i>
R5000 ¹	Attempted to update a read-only table	<i>No subcondition</i>
R6000 ¹	Metadata no longer available	<i>No subcondition</i>
R7000 ¹	Table in request not reserved in transaction	<i>No subcondition</i>
RR000 ¹	Oracle Rdb returned an error	<i>No subcondition</i>
S0000 ¹	No implicit transaction	<i>No subcondition</i>
S1001 ¹	Context exception	Bad version in context structure
S1002 ¹		Bad type in context structure
S1003 ¹		Bad length in context structure
S2000 ¹	Row containing list deleted	<i>No subcondition</i>
S3000 ¹	Invoker was not the definer	<i>No subcondition</i>
S4001 ¹	Alias exception	Alias unknown
S4002 ¹		Alias already declared
S7000 ¹	Base system does not support feature being used	<i>No subcondition</i>
S6000 ³	Case not found; WHEN or ELSE not specified	<i>No subcondition</i>
S7000 ¹	Bad SQL version	<i>No subcondition</i>
S5001 ¹	Negative length and scale for column	Negative length specified for column

¹Oracle Rdb specific SQLSTATE code

²Obsolete. Use SQLSTATE 09000 instead

³Obsolete. Use SQLSTATE 20000 instead

(continued on next page)

Table C–5 (Cont.) SQLSTATE Status Parameter Values—Sorted by SQLSTATE Class and Subclass

Class /Subclass	Condition	Subcondition
S5002 ¹		Negative scale specified for column
¹ Oracle Rdb specific SQLSTATE code		

C.5.2 Use of the SQLSTATE Status Parameter

Table C–5 shows the SQLSTATE classes 00, 01, and 02 as completion conditions of success, warning, and no data respectively. All other classes define exception conditions.

When using embedded SQL, the embedded exception declaration defines the following categories of exceptions:

- NOT FOUND: SQLSTATE class = 02
- SQLWARNING: SQLSTATE class = 01
- SQLEXCEPTION: SQLSTATE class > 02
- SQLERROR: SQLEXCEPTION or SQLWARNING

Example C–11 shows how to declare SQLSTATE as a parameter in a C program and how to evaluate the SQLSTATE value using the string compare function. When you declare SQLSTATE in a C program, you must type SQLSTATE in all uppercase characters.

Example C–11 Declaring SQLSTATE in a C Program

```
char SQLSTATE[6];
long SQLCODE;

main()
{
    EXEC SQL SELECT T_INT INTO :c1 FROM FOUR_TYPES
           WHERE T_DECIMAL = 4.1;
    printf ("SQLCODE should be < 0; its value is %ld\n", SQLCODE);
    printf ("SQLSTATE should be '22002'; its value is %s\n", SQLSTATE);
    if (SQLCODE >= 0 || strcmp (SQLSTATE, "22002", 5) != 0)
        flag = 0;
}
```

You can use the GET DIAGNOSTICS statement to return the SQLSTATE information to your program. For more information, see the GET DIAGNOSTICS Statement.

Note that Oracle Rdb provides a set of include file for the value of SQLSTATE. These file are located in SYS\$LIBRARY with the following names:

Table C–6 Include Files for SQLSTATE

File Name	Description
SQLSTATE.BAS	BASIC include file
SQLSTATE.FOR	Fortran include file
SQLSTATE.H	C or C++ header file
SQLSTATE.LIB	COBOL include file
SQLSTATE.PAS	Pascal include file
SQLSTATE.SQL	SQL declare file

In addition a special script (SQLSTATE_TABLE.SQL) is provided to create a table (SQLSTATE_TABLE) in a database and populate it with the values and symbolic names.

Oracle Corporation will periodically add to these definition files as new SQLSTATE values are used by Oracle Rdb, or as required by the ANSI and ISO SQL database standard.

D

The SQL Dynamic Descriptor Areas (SQLDA and SQLDA2)

An SQL Descriptor Area (SQLDA) is a collection of parameters used only in dynamic SQL programs. SQL provides two descriptor areas: SQLDA and SQLDA2. Sections D.6 through D.6.2 include information specific to the SQLDA2.

Dynamic SQL lets programs accept or generate SQL statements at run time, in contrast to SQL statements that are part of the source code for precompiled programs or SQL module language procedures. Unlike precompiled SQL or SQL module language statements, such dynamically executed SQL statements are not necessarily part of a program's source code, but can be generated while the program is running. Dynamic SQL is useful when you cannot predict the type of SQL statement your program will need to process.

To use an SQLDA, host languages must support pointer variables that provide indirect access to storage by storing the address of data instead of directly storing data in the variable. The languages supported by the SQL precompiler that also support pointer variables are PL/I, C, BASIC, and Ada. Any other language that supports pointer variables can use an SQLDA, but must call SQL module procedures containing SQL statements instead of embedding the SQL statements directly in source code.

D.1 Purpose of the SQLDA

The SQLDA provides information about dynamic SQL statements to the program and information about memory allocated by the program to SQL. Specifically, SQL and host language programs use the SQLDA for the following purposes:

- SQL uses the SQLDA as a place to *write* information about parameter markers and select list items in a prepared statement. SQL writes information about the number and data types of input and output parameter markers and select list items to the SQLDA when it processes `PREPARE . . . SELECT LIST INTO` statements or `DESCRIBE` statements.

Parameter markers are question marks (?) that denote parameters in the statement string of a PREPARE statement. SQL replaces parameter markers with values in parameters or dynamic memory when it executes a dynamic SQL statement.

The DESCRIBE statement writes information about select list items in a prepared SELECT statement to the SQLDA so the host language program can allocate storage (parameters or dynamic memory) for them. The storage allocated by the program then receives values in rows of the prepared SELECT statement's result table in subsequent FETCH statements.

An SQLDA at any particular time can contain information about either input or output parameter markers or select list items, but not about both:

- SQL writes information about select list items to the SQLDA when it executes DESCRIBE . . . SELECT LIST or PREPARE . . . SELECT LIST statements.
- SQL writes information about parameter markers to the SQLDA when it executes DESCRIBE . . . MARKERS statements. If a prepared statement has no parameter markers, a DESCRIBE . . . MARKERS statement puts values in the SQLDA to indicate that there are no parameter markers.
- The program uses the SQLDA as a place to *read* the information SQL wrote to the SQLDA about any select list items, or input or output parameter markers in the prepared statement:
 - After either a DESCRIBE . . . SELECT LIST or DESCRIBE . . . MARKERS statement, the program reads the number and data type of select list items or parameter markers.

The program uses that information to allocate storage (either by declaring parameters or allocating dynamic memory) for values that correspond to the parameter markers or select list items.
- The program uses the SQLDA as a place to *write* the addresses of the storage it allocated for parameter markers and select list items.
- SQL uses the SQLDA as a place to *read* information about parameter markers or select list items:
 - In OPEN statements, SQL reads the addresses of a prepared SELECT statement's parameter markers to set up a cursor for the program to process.

- In FETCH statements, SQL reads the addresses of a prepared SELECT statement's select list items so it can write the values of the row being fetched to the storage allocated by the program.
- In EXECUTE statements, SQL reads the addresses of parameter markers of any prepared statement other than a SELECT statement.

The OPEN and FETCH statements used to read information from the SQLDA are not themselves dynamic statements used in a PREPARE statement, nor is a DECLARE CURSOR statement that declares the cursor named in the OPEN and FETCH statements. Although these statements *use* prepared statements, they are among the SQL statements that cannot themselves *be* prepared statements. See the PREPARE Statement for a list of statements that cannot be dynamically executed.

- The behavior of dynamic SQL and the SQLDA in particular can be controlled using the SET SQLDA Statement. For instance, parameter markers can be named variables instead of simple question marks (?).

D.2 How SQL and Programs Use the SQLDA

The specific sequence of operations that uses the SQLDA depends on whether a program can accept dynamically generated SELECT statements only, non-SELECT statements only, or both. The following sequence describes in general the steps a program follows in using the SQLDA. For specific examples, see the chapter on using dynamic SQL in the *Oracle Rdb Guide to SQL Programming* and the sample programs created during installation of Oracle Rdb in the Samples directory.

1. The program uses the embedded SQL statement INCLUDE SQLDA to automatically declare an SQLDA. In addition, the program must allocate memory for the SQLDA and set the value of one of its fields, SQLN. The value of SQLN specifies the maximum number of parameter markers or select list items about which information can be stored in the SQLDA.

Programs can use more than one SQLDA but must explicitly declare additional SQLDA structures with names other than SQLDA. Declaring two SQLDAs can be useful for dynamic SQL programs that can accept both SELECT and non-SELECT statements. One SQLDA stores information about parameter markers and another stores information about select list items. (An alternative to declaring multiple SQLDA structures in such programs is to issue additional DESCRIBE . . . SELECT LIST statements after the program finishes with parameter marker information in the SQLDA.)

Declaration and allocation of SQLDAs need to be done only once. The remaining steps repeat as many times as the program has dynamic SQL statements to process.

2. SQL writes the number and data types of any select list items (for a DESCRIBE . . . SELECT LIST statement) or parameter markers (for a DESCRIBE . . . MARKERS statement) of a prepared statement into the SQLDA. SQL puts the number of select list items or parameter markers in the SQLD field of the SQLDA, and stores codes denoting their data types in the SQLTYPE fields.
3. If the program needs to determine if a particular prepared statement is a SELECT statement, it reads the value of the second element of the SQLCA.SQLERRD array after a DESCRIBE . . . SELECT LIST statement. If the value is one, the prepared statement is a SELECT statement and the program needs to allocate storage for rows generated during subsequent FETCH statements.
4. When you use parameter markers in SQL statements, you should not make any assumptions about the data types of the parameters. SQL may convert the parameter to a data type that is more appropriate to a particular operation. For example, when you use a parameter marker as one value expression in a LIKE predicate, SQL returns a data type of VARCHAR for that parameter even though the other value expression has a data type of CHAR. The STARTING WITH predicate and the CONTAINING predicate treat parameter markers in the same way. You can override the VARCHAR data type in such predicates by explicitly setting the SQLTYPE field of the SQLDA to CHAR.
5. The program reads information about the number, data type, and length of any select list items (after a DESCRIBE . . . SELECT LIST statement) or parameter markers (after a DESCRIBE . . . MARKERS statement) from the SQLDA. The program then allocates storage (parameters or dynamic memory) for each of the select list items or parameters, and writes the addresses for that storage to the SQLDA. The program puts the addresses into the SQLDATA fields of the SQLDA.

If SQL uses a data type for the parameter marker or select list item that is not supported by the programming language, the program must convert the SQLTYPE and SQLLEN fields to an appropriate data type and length. The program changes the values of SQLTYPE and SQLLEN that SQL returns from the DESCRIBE statement to a data type and length that both SQL and the host language support.
6. The program supplies values that will be substituted for parameter markers and writes those values to the storage allocated for them.

7. SQL reads information about parameter markers from the SQLDA:
 - If the prepared statement is a prepared SELECT statement, SQL reads the addresses of any parameter markers for that prepared SELECT statement when it executes an OPEN statement that refers to the SQLDA.
 - If the statement is any other prepared statement, SQL reads the addresses of parameter markers for that statement when it executes an EXECUTE statement that refers to the SQLDA.

SQL uses the addresses of parameter markers to retrieve the values in storage (supplied by the program) and to substitute them for parameter markers in the prepared statement.

8. Finally, for prepared SELECT statements only, SQL reads the addresses of select list items when it executes a FETCH statement that refers to the SQLDA. SQL uses the information to write the values from the row of the result table to memory.

D.3 Declaring the SQLDA

Programs can declare the SQLDA in the following ways:

- By using the INCLUDE SQLDA statement embedded in Ada, C, or PL/I programs to be precompiled. The INCLUDE SQLDA statement automatically inserts a declaration of an SQLDA structure, called SQLDA, in the program when it precompiles the program.
- In precompiled Ada programs, by specifying the SQLDA_ACCESS type in the SQL definition package. Specifying SQLDA_ACCESS offers an advantage over an embedded INCLUDE SQLDA statement because you can use it in more than one declaration to declare multiple SQLDA structures.
- In precompiled C programs and C host language programs, you can use the sql_sqlda.h header file. The following example shows how to include the file in a C program:

```
#include <sql_sqlda.h>
```

The sql_sqlda.h header file includes typedef statements for the SQLDA structure defining the SQL_T_SQLDA (or the SQL_T_SQLDA2) data type. In addition, it defines the SQL_T_SQLDA_FULL (or SQL_T_SQLDA2_FULL) data type as a superset to the definition of the SQLDA structure. The SQL_T_SQLDA_FULL data type is identical in layout to the SQL_T_SQLDA data type except that it contains additional unions with additional fields that SQL uses when describing CALL statements.

For additional information on declaring SQLDA structures, see the *Oracle Rdb Guide to SQL Programming*.

- By explicitly declaring the SQLDA in programs written in host languages that support pointer variables. Such host languages can then take advantage of dynamic SQL even though the SQL precompiler does not support them. Instead of embedding SQL statements directly in the host language source code, languages unsupported by the precompiler must call SQL module language procedures that contain SQL statements to use dynamic SQL. See Chapter 3 for more information about the SQL module language.

Programs that explicitly declare SQLDA structures (whether or not they have precompiler support) supply a name for the SQLDA structure, which can be SQLDA or any other valid name. Declaring two SQLDAs can be useful for dynamic SQL programs that can accept both SELECT and non-SELECT statements. One SQLDA stores information about parameter markers and another stores information about select list items.

An SQLDA always includes four fields, and may sometimes include a fifth field. The fifth field, SQLVAR, is a repeating field. For languages other than C, it comprises five parameters that describe individual select list items or parameter markers of a prepared statement. For C, it comprises six parameters.

The following examples show declarations of the SQLDA for different host languages. For PL/I, C, and Ada, the examples show the declaration SQL inserts when it processes a program that contains the INCLUDE SQLDA statement. For BASIC, the example shows the format a program should use when it declares the SQLDA explicitly.

These sample declarations all use the name SQLDA as the name for the SQLDA structure, but programs can use any valid name.

Example D-1 shows the declaration that SQL inserts when it processes a program that contains the INCLUDE SQLDA statement.

Example D-1 Declaration of the SQLDA in Ada

```
type SQLNAME_REC is
  record
    NAME_LEN : standard.short_integer;
    NAME_STR : standard.string (1..30);
  end record;

type SQLVAR_REC is
  record
    SQLTYPE : standard.short_integer;
    SQLLEN : standard.short_integer;
    SQLDATA : system.address;
    SQLIND : system.address;
    SQLNAME : sqlname_rec;
  end record;
type SQLVAR_ARRAY is array (1..255) of sqlvar_rec;

type SQLDA_RECORD;
type SQLDA_ACCESS is access SQLDA_RECORD;
type SQLDA_RECORD is
  record
    SQLDAID : standard.string (1..8) := 'SQLDA  ';
    SQLDABC : standard.integer;
    SQLN : standard.short_integer;
    SQLD : standard.short_integer;
    SQLVAR : sqlvar_array;
  end record;
```

Example D-2 shows the format that BASIC programs should use when they explicitly declare the SQLDA.

Example D-2 Declaration of the SQLDA in BASIC

```
RECORD SQLDA_REC
  string SQLDAID = 8
  long SQLDABC
  word SQLN      ! Program must explicitly
  word SQLD      ! set SQLN equal to the number
  GROUP SQLVAR(100) ! of occurrences of SQLVAR
  word SQLTYPE
  word SQLLEN
  long SQLDATA
  long SQLIND
```

(continued on next page)

Example D-2 (Cont.) Declaration of the SQLDA in BASIC

```
GROUP SQLNAME
  word SQLNAME
  string SQLNAMEC = 30
END GROUP SQLNAME
END GROUP SQLVAR
END RECORD SQLDA_REC

DECLARE SQLDA_REC SQLDA
```

Example D-3 shows the declaration that SQL inserts when it processes a C program that contains the INCLUDE SQLDA statement.

Example D-3 Declaration of the SQLDA in C

```
struct SQLDA_STRUCT {
    char SQLDAID[8];
    int SQLDABC;
    short SQLN;
    short SQLD;
    struct SQLVAR_STRUCT {
        short SQLTYPE;
        short SQLLEN;
        char *SQLDATA;
        short *SQLIND;
        short SQLNAME_LEN;
        char SQLNAME[30];
        } SQLVAR[1];
    } *SQLDA;
```

Example D-4 shows the declaration that SQL inserts when it processes a PL/I program that contains the INCLUDE SQLDA statement.

Example D–4 Declaration of the SQLDA in PL/I

```
/*  
    EXEC SQL INCLUDE SQLDA;  
*/  
DCL 1 SQLDA BASED ( SQLDAPTR ),  
    2 SQLDAID CHAR(8),  
    2 SQLDABC BIN FIXED(31),  
    2 SQLN BIN FIXED(15),  
    2 SQLD BIN FIXED(15),  
    2 SQLVAR (SQLSIZE REFER(SQLN)),  
    3 SQLTYPE BIN FIXED(15),  
    3 SQLLEN BIN FIXED(15),  
    3 SQLDATA PTR,  
    3 SQLIND PTR,  
    3 SQLNAME CHAR(30) VAR;  
DCL SQLSIZE BIN FIXED;  
DCL SQLDAPTR PTR;
```

D.4 Description of Fields in the SQLDA

Table D–1 describes the different fields of the SQLDA and the ways SQL uses the fields. Remember that the SQLDA, at any particular time, can contain information about either select list items or parameter markers, but not both.

Table D–1 Fields in the SQLDA

Field Name	Meaning of the Field	Set by	Used by
SQLDAID	Character string field whose value is always the character string "SQLDA".	Program	SQL to determine if the structure is an SQLDA or an SQLDA2.
SQLDABC	The length in bytes of the SQLDA, which is a function of SQLN ($\text{SQLDABC} = 16 + (44 * \text{SQLN})$).	SQL	Not used.
SQLN	The total number of occurrences of the SQLVAR group field (the value must equal or exceed the value in SQLD, or the DESCRIBE statement). Generates a run-time error.	Program	SQL to determine if a program allocated enough storage for the SQLDA.

(continued on next page)

Table D–1 (Cont.) Fields in the SQLDA

Field Name	Meaning of the Field	Set by	Used by
SQLD	Number of output items (if DESCRIBE . . . OUTPUT) or parameter markers (if DESCRIBE . . . INPUT) in prepared statement (if none, the value is 0).	SQL	Program to determine how many input or output parameters for which to allocate storage.
SQLVAR	A repeating group field, each occurrence of which describes a select list item or parameter marker (not used if the value of SQLD is 0).	No value	See descriptions of subfields in the following entries.

SQLVAR Subfields (Each Occurs Once for Each Select List Item or Parameter Marker)

Field Name	Meaning of the Field	Set by	Used by
SQLTYPE	A subfield of SQLVAR whose value indicates the data type of the select list item or parameter marker (see Table D–2).	SQL	Program to allocate storage with the appropriate data type for the parameter.
SQLLEN	<p>A subfield of SQLVAR whose value indicates the length in bytes of the select list item or parameter marker.</p> <p>For CHAR² and CHARACTER VARYING², indicates the declared length of the data without length field overhead.</p> <p>For fixed-length data types (TINYINT, SMALLINT, INTEGER, BIGINT, and DECIMAL), SQLLEN is split in half.</p> <p>For TINYINT, SMALLINT, INTEGER, and BIGINT, the low-order byte of SQLLEN indicates the length, and the high-order byte indicates the scale (the number of digits to the right of the decimal point).</p>	SQL unless program resets, except DECIMAL or H_FLOAT, which can only be set by user	Program to allocate storage with the appropriate size for the select list item or parameter marker.

²Includes VARCHAR, VARCHAR2, NATIONAL CHARACTER VARYING, RAW, and LONG VARCHAR

(continued on next page)

Table D–1 (Cont.) Fields in the SQLDA

SQLVAR Subfields (Each Occurs Once for Each Select List Item or Parameter Marker)			
Field Name	Meaning of the Field	Set by	Used by
	<p>For DECIMAL, the low-order byte indicates the precision, and the high-order byte indicates the scale. However, the SQLLEN for a DECIMAL data type can be set only by the user; it is not returned by SQL on a DESCRIBE statement.</p> <p>List cursors cannot return data in data types that require a scale factor.</p> <p>For floating-point data types, the SQLLEN shows the length of the field in bytes so that SQLLEN = 4 indicates the REAL data type, SQLLEN = 8 indicates DOUBLE PRECISION, and SQLLEN = 16 indicates the H_FLOAT data type. The floating point representation of the data (VAX versus IEEE) is determined by the /FLOAT qualifier on the SQLSPRE command line.</p>		
SQLDATA	<p>A subfield of SQLVAR whose value is the address of the storage allocated for the select list item or parameter marker.</p> <p>For CHARACTER VARYING ², allocate sufficient memory to allow the length field (that is, SQLLEN plus two octets).</p>	Program	<p>SQL:</p> <ul style="list-style-type: none">• In EXECUTE and OPEN statements, to retrieve a value stored by the program and substitute it for a parameter marker in the prepared statement.• In FETCH statements, to store a value from a result table.

²Includes VARCHAR, VARCHAR2, NATIONAL CHARACTER VARYING, RAW, and LONG VARCHAR

(continued on next page)

Table D–1 (Cont.) Fields in the SQLDA

SQLVAR Subfields (Each Occurs Once for Each Select List Item or Parameter Marker)			
Field Name	Meaning of the Field	Set by	Used by
SQLIND	A subfield of SQLVAR whose value is the address of the indicator variable, a word (16 bits) in size (if program does not set SQLIND, the value is 0).	Program	Program or SQL: <ul style="list-style-type: none">In FETCH statements, by SQL, to store the value for an indicator variable associated with a select list item.After FETCH statements, by the program, to retrieve the value of a select list item's associated indicator variable.In EXECUTE and OPEN statements, by SQL, to retrieve the value of a parameter marker's associated indicator variable.
SQLNAME	<p>A varying character string subfield of SQLVAR whose value is:</p> <p>For output items, the name of the column in the select list of the prepared SELECT statement.</p> <p>For input, the name of the column to which a parameter marker is assigned (in INSERT or UPDATE statements) or compared (in basic predicates).</p> <p>If the select list item, assignment, or comparison involves an arithmetic expression or predicates other than basic predicates; SQL does not assign a value to SQLNAME.</p>	SQL	The program, optionally, to find out the name of the column associated with a select list item or parameter marker.

Table D–2 shows the numeric and literal values for the SQLTYPE subfield of SQLVAR and the meaning of those values.

Table D–2 Codes for SQLTYPE Field of SQLDA and SQLDA2

Numeric Value	Literal Value	Data Type
449	SQLDA_VARCHAR	VARCHAR ¹ , CHARACTER VARYING ¹
453	SQLDA_CHAR	CHAR, CHARACTER
481	SQLDA_FLOAT	FLOAT ⁵ , REAL, DOUBLE PRECISION
485	SQLDA_DECIMAL	DECIMAL
497	SQLDA_INTEGER	INTEGER
501	SQLDA_SMALLINT	SMALLINT
503	SQLDA_DATE	DATE VMS
505	SQLDA_QUADWORD	BIGINT
507	SQLDA_ASCIZ	ASCIZ ²
509	SQLDA_SEGSTRING	LIST OF BYTE VARYING
515	SQLDA_TINYINT	TINYINT
516	SQLDA_VARBYTE	VARBYTE ^{3,4}
519	SQLDA2_DATETIME	Date-time (ANSI)
521	SQLDA2_INTERVAL	INTERVAL
909	SQLDA2_VARBINARY	BINARY VARYING (VARBINARY)
913	SQLDA2_BINARY	BINARY

¹For the SQLDA2 structure, this data type has a longword length prefix.

²The SQLTYPE code for ASCIZ is never returned in the SQLDA by a DESCRIBE statement, but it can be used to override the data type that is returned.

³This data type value is only valid for fetches of list elements.

⁴This data type does not allow null values.

⁵The floating point representation assumed by SQL for the floating point number is determined by the /FLOAT qualifier on the SQL\$MOD or SQL\$PRE command line.

SQL provides a file that contains the declarations of all the SQLTYPE literal values. Table C–2 shows how to include this file in precompiled SQL and module language programs.

There is some confusion over the use of ASCII and ASCIZ in dynamic SQL and C programs. When a CHAR data type is written to the database using INSERT or UPDATE, the string is not padded with blank spaces. It contains a null-terminated character, which makes it difficult to access the data.

SQL does not know what the host language is when using dynamic SQL; it returns the data type of the field as in the DESCRIBE statement, (CHAR(n)), and not the data type of the user's host variable. The interpretation of CHAR(n) being ASCIZ is for host variables and not database variables.

If you change the SQLDA's SQLTYPE from CHAR to ASCIZ and increase SQLLEN by 1, no truncation occurs and the CHAR STRING fields will be padded with blank spaces accordingly (where incrementing SQLLEN by 1 accounts for the null terminator).

Note

SQL sets the value of SQLTYPE during the DESCRIBE statement. However, your application program can change the value of SQLTYPE to that of another data type.

For example, SQL does not support the DECIMAL data type in database columns. This means that SQL will never return the code for the DECIMAL data type in the SQLTYPE field in the SQLDA. However, programs can set the code to that for DECIMAL, and SQL will convert data from databases to DECIMAL, and data from DECIMAL parameters in the program to the data type in the database.

However, SQL assumes that program parameters will correspond to the data type indicated by the SQLTYPE code. If they do not, SQL may generate unpredictable results.

D.5 Parameters Associated with the SQLDA: SQLSIZE and SQLDAPTR

In addition to the declaration of the SQLDA itself, SQL declares two related parameters: SQLSIZE and SQLDAPTR. These parameters can only be used in PL/I programs. The PL/I program uses both parameters when it dynamically allocates storage for the SQLDA before a DESCRIBE or PREPARE . . . SELECT LIST INTO statement. Your program must:

- Assign a value to SQLSIZE and then assign the same value to SQLN. Because the declaration of the SQLDA refers both to SQLSIZE and SQLN, the program uses that value when it allocates memory for the SQLDA.
- Dynamically allocate memory for the SQLDA based on the value assigned to SQLN, and assign the address for memory used by the SQLDA into SQLDAPTR.

The following program fragment shows how a PL/I program uses `SQLSIZE` and `SQLDAPTR` to allocate storage for the `SQLDA`:

```
#include <stdlib.h>
#define SQLVAR_ELEMENTS 20

/* Declare the SQL Descriptor Area: */
exec sql
    include SQLDA;

/* Allocate memory for the SQLDA and
 * set the value of its SQLN field:
 */
SQLDA = malloc (16 + 44 * SQLVAR_ELEMENTS);
SQLDA->SQLN = SQLVAR_ELEMENTS;
```

D.6 Purpose of the `SQLDA2`

SQL provides an extended version of the `SQLDA`, called the `SQLDA2`, which supports additional fields and field sizes.

You can use either the `SQLDA` or `SQLDA2` in any dynamic SQL statement that calls for a descriptor area. SQL assumes default values for `SQLDA2` fields and field sizes if you use an `SQLDA` structure to provide input parameters for an application; however, SQL issues an error message if the application cannot represent resulting values.

Use the `SQLDA2` instead of the `SQLDA` when any of the following applies to the parameter markers or select list items:

- The length of the column name is greater than 30 octets. (An octet is 8 bits.)
- The data type of the column is `DATE`, `DATE VMS`, `DATE ANSI`, `TIME`, `TIMESTAMP`, or any of the interval data types.
- The data type is `CHAR`, `CHAR VARYING`, `CHARACTER`, `CHARACTER VARYING`, `VARCHAR`, `LONG VARCHAR`, or `RAW` and any of the following is true:
 - The character set is not the default 8-bit character set.
 - The maximum length in octets exceeds 32,767.

You can examine the `SQLDA2` after SQL fills in the items on a `PREPARE` statement. Oracle Rdb recommends this rather than setting the fields yourself.

Use one of the following methods to extract the data for your own use:

- The `CAST` function to convert the data to `TEXT` before using it

- The EXTRACT function to extract individual fields so you can format it
- The CAST function to convert to DATE VMS so that you can use OpenVMS system services

The ANSI/ISO SQL standard specifies that the data is always returned to the application program as CHAR data.

D.6.1 Declaring the SQLDA2

Programs can declare the SQLDA2 in the same way as they declare an SQLDA, described in Section D.3.

To indicate to SQL that the structure is an SQLDA2 instead of an SQLDA, your program must set the SQLDAID field to be the character string containing the word SQLDA2 followed by two spaces.

The following examples show declarations of the SQLDA2 for different host languages. For PL/I, C, and Ada, the examples show the declaration SQL inserts when it processes a program that contains the INCLUDE SQLDA statement. For other languages, the examples show the format that programs should use when they explicitly declare the SQLDA.

Example D-5 shows the declaration that SQL inserts when it processes an Ada program that contains the INCLUDE SQLDA2 statement. In this example, *N* stands for the maximum number of occurrences of SQLVAR2.

Example D-5 Declaration of the SQLDA2 in Ada

(continued on next page)

Example D-5 (Cont.) Declaration of the SQLDA2 in Ada

```
type SQLNAME_REC is
  record
    NAME_LEN : standard.short_integer;
    NAME_STR : standard.string (1..128);
  end record;
type SQLVAR_REC is
  record
    SQLTYPE : standard.short_integer;
    SQLLEN : standard.integer;
    SQLDATA : system.address;
    SQLIND : system.address;
    SQLCHRONO_SCALE: standard.integer;
    SQL_CHRONO_PRECISION: standard.integer;
    SQLNAME : sqlname_rec;
    SQLCHAR_SET_NAME : standard.string(1..128);
    SQLCHAR_SET_SCHEMA : standard.string(1..128);
    SQLCHAR_SET_CATALOG : standard.string(1..128);
  end record;
type SQLVAR_ARRAY is array (1..N) of sqlvar_rec;
type SQLDA_RECORD;
type SQLDA_ACCESS is access SQLDA_RECORD;
type SQLDA_RECORD is
  record
    SQLDAID : standard.string (1..8) := 'SQLDA2 ';
    SQLDABC : standard.integer;
    SQLN : standard.short_integer;
    SQLD : standard.short_integer;
    SQLVAR : sqlvar_array;
  end record;
```

Example D-6 shows the format that BASIC programs should use when they explicitly declare the SQLDA2.

Example D-6 Declaration of the SQLDA2 in BASIC

```
RECORD SQLDA_REC
  string SQLDAID = 8      ! Value must be "SQLDA2 ".
  long SQLDABC
```

(continued on next page)

Example D-6 (Cont.) Declaration of the SQLDA2 in BASIC

```
word SQLN          ! Program must explicitly
word SQLD          ! set SQLN equal to the number
GROUP SQLVAR(N)    ! of occurrences of SQLVAR.
  word SQLTYPE
  long SQLLEN
  long SQLDTEXT_LEN
  long SQLDATA
  long SQLIND
  long SQLCHRONO_SCALE
  long SQLCHRONO_PRECISION
  GROUP SQLNAME
    word SQLNAME
    string SQLNAMEC = 128
  END GROUP SQLNAME
  string SQLCHAR_SET_NAME = 128
  string SQLCHAR_SET_SCHEMA = 128
  string SQLCHAR_SET_CATALOG = 128
END GROUP SQLVAR
END RECORD SQLDA_REC
DECLARE SQLDA_REC SQLDA2
```

Example D-7 shows the declaration that SQL inserts when it processes a C program that contains the INCLUDE SQLDA2 statement.

Example D-7 Declaration of the SQLDA2 in C

```
struct SQLDA_STRUCT {
    char SQLDAID[8]; /*Value must be "SQLDA2" */
    int SQLDABC; /* ignored. */
    short SQLN;
    short SQLD;
```

(continued on next page)

Example D–7 (Cont.) Declaration of the SQLDA2 in C

```
struct {
    short SQLTYPE;
    long  SQLLEN;
    long  SQLOCTET_LEN
    char  *SQLDATA;
    long  *SQLIND;
    long  SQLCHRONO_SCALE
    long  SQLCHRONO_PRECISION
    short SQLNAME_LEN;
    char  SQLNAME[128];
    char  SQLCHAR_SET_NAME[128];
    char  SQLCHAR_SET_SCHEMA[128];
    char  SQLCHAR_SET_CATALOG[128];
} SQLVAR[N];          /* N is maximum number of */
} *SQLDA;              /* occurrences of SQLVAR. */
```

D.6.2 Description of Fields in the SQLDA2

The SQLVAR2 field for an SQLDA2 structure comprises the following parameters that describe individual select list items or parameter markers of a prepared statement:

- Length (SQLLEN and SQLOCTET_LEN fields)

Note

There is a major difference between the SQLLEN fields in the SQLDA and the SQLDA2. In the SQLDA, the SQLLEN field contains the length of the field in bytes. In the SQLDA2, the SQLLEN field either contains the length of the field in characters or is a subtype field for certain data types (INTERVAL and LIST OF BYTE VARYING). This is the case when you issue the DESCRIBE statement to return information from SQL to your program.

The SQLOCTET_LEN field in the SQLDA2 is analogous to the SQLLEN field in the SQLDA. Use SQLOCTET_LEN instead of SQLLEN to allocate dynamic memory for the SQLDATA field when using the SQLDA2.

- Data type (SQLTYPE)
- Scale and precision (SQLLEN or SQLCHRONO_SCALE and SQLCHRONO_PRECISION)

- Character set information (SQLCHAR_SET_NAME, SQLCHAR_SET_SCHEMA, SQLCHAR_SET_CATALOG)
- Data value (SQLDATA)
- Null indicator value (SQLIND)
- Name for resulting columns of a cursor specification (SQLNAME)

Table D–3 describes the different fields of the SQLDA2 and the ways in which SQL uses the fields when passing them to dynamic SQL. Remember that the SQLDA2 at any particular time can contain information about either select list items or parameter markers, but not both.

Table D–3 Fields in the SQLDA2

Field Name	Meaning of the Field	Set by	Used by
SQLDAID	Character string field whose value is always the character string "SQLDA2 " (SQLDA2 followed by two spaces).	Program	SQL to determine if the structure is an SQLDA or an SQLDA2.
SQLDABC	The length in bytes of the SQLDA2, which is a function of SQLN (SQLDABC = 16+ (540 * SQLN)).	SQL	Not used.
SQLN	The total number of occurrences of the SQLVAR2 group field (the value must equal or exceed the value in SQLD or the DESCRIBE or PREPARE OUTPUT INTO statement). Generates a run-time error.	Program	SQL to determine if program allocated enough storage for the SQLDA.
SQLD	Number of select list items (if DESCRIBE . . . OUTPUT) or parameter markers (if DESCRIBE . . . INPUT) in prepared statement (if none, the value is 0).	SQL	Program to determine how many input or output parameters for which to allocate storage.
SQLVAR2	A repeating group field, each occurrence of which describes a select list item or parameter marker (not used if the value of SQLD is 0).	No value	See descriptions of subfields in following entries.

SQLVAR2 Subfields (Each Occurs Once for Each Select List Item or Parameter Marker):

Field Name	Meaning of the Field	Set by	Used by
SQLTYPE	A subfield of SQLVAR2 whose value indicates the data type of the select list item or parameter marker (see Table D–2).	SQL	Program to allocate storage with the appropriate data type for the parameter.

(continued on next page)

Table D–3 (Cont.) Fields in the SQLDA2

SQLVAR2 Subfields (Each Occurs Once for Each Select List Item or Parameter Marker):			
Field Name	Meaning of the Field	Set by	Used by
SQLLEN	<p>A subfield of SQLVAR2 whose value indicates the length of the select list item or parameter marker.</p> <p>For character types, CHAR, CHARACTER VARYING types SQLLEN indicates the declared size, not including length overheads. See SQLOCTET_LEN.</p> <p>For fixed-length data types (TINYINT, SMALLINT, INTEGER, BIGINT, NUMERIC, and DECIMAL), SQLLEN is split in half.</p> <p>SQLSIZE—the low-order 16 bits</p> <ul style="list-style-type: none">For TINYINT, SMALLINT, INTEGER, and BIGINT; SQLSIZE and SQLOCTET_LENGTH indicate the length in bytes of the select list item or parameter marker.For DECIMAL; SQLSIZE indicates the precision. However, the SQLLEN for a DECIMAL data type can only be set by the user; it is not returned by SQL on a DESCRIBE statement. <p>SQLSCALE—the high-order 16 bits</p> <ul style="list-style-type: none">SQLSCALE indicates the scale (the number of digits to the right of the decimal point).List cursors cannot return data in data types that require a scale factor. <p>For floating-point data types, SQLLEN and SQLOCTET_LEN are the size in octets of the select list item or parameter marker.</p>	SQL, unless the program resets, except for DECIMAL, which can only be set by the user.	

(continued on next page)

Table D–3 (Cont.) Fields in the SQLDA2

SQLVAR2 Subfields (Each Occurs Once for Each Select List Item or Parameter Marker):			
Field Name	Meaning of the Field	Set by	Used by
SQLOCTET_LEN	For DATE, DATE ANSI, DATE VMS, TIME, or TIMESTAMP, SQLLEN is the length of the date-time data type.	SQL, unless the program resets.	Program to allocate storage with the appropriate size for the select list item or parameter marker.
	For INTERVAL data types, SQLLEN is set to one of the codes specified in Table D–4.		
SQLCHRONO_SCALE	A subfield of SQLVAR2 whose value indicates the length in octets of the select list item or parameter marker.	SQL, unless the program resets.	Program or SQL.
	<p>If SQLTYPE indicates CHAR¹, then SQLOCTET_LEN is the maximum possible length in octets of the character string.</p> <p>If SQLTYPE² indicates CHARACTER VARYING, SQLOCTET_LEN is the maximum possible length in octets required to represent the character string, including the octets required to represent the string length (that is, 4 additional octets.)</p> <p>If SQLTYPE indicates a fixed-scale or floating-point numeric data type, SQLOCTET_LEN is the size in octets of the numeric select list item or parameter marker.</p> <p>If SQLTYPE indicates a date-time or interval data type, then dynamic SQL ignores SQLOCTET_LEN.</p>		
SQLCHRONO_SCALE	A longword subfield of SQLVAR2 whose value indicates the specific date-time data type of the column.	SQL, unless the program resets.	Program.
	<p>When SQLTYPE represents a date-time data type, SQLCHRONO_SCALE contains a code specified in Table D–5.</p> <p>When SQLTYPE represents an interval data type, SQLCHRONO_SCALE contains the implied or specified interval leading field precision.</p>		

¹Includes CHARACTER, NCHAR, NATIONAL CHARACTER

²Includes VARCHAR, VARCHAR2, NATIONAL CHARACTER VARYING, RAW, and LONG VARCHAR

(continued on next page)

Table D–3 (Cont.) Fields in the SQLDA2

SQLVAR2 Subfields (Each Occurs Once for Each Select List Item or Parameter Marker):			
Field Name	Meaning of the Field	Set by	Used by
SQLCHRONO_ PRECISION	When SQLTYPE represents a data type that is neither date-time nor interval, SQLCHRONO_SCALE contains 0.	SQL, unless the program resets.	Program.
	A longword subfield of SQLVAR2 whose value indicates the precision of the column represented by SQLVAR2 when that column has a date-time data type.		
	When SQLTYPE represents a TIME or TIMESTAMP data type, SQLCHRONO_PRECISION contains the time precision or timestamp precision.		
SQLCHAR_SET_ NAME	When SQLTYPE represents an interval data type with a fractional seconds precision, SQLCHRONO_PRECISION is set to that value. Otherwise, SQLCHRONO_PRECISION is set to 0.	SQL, unless the program resets.	The SQLCHAR_SET_NAME field indicates the character set name of a select list item or parameter marker if the select list item or parameter marker has a character data type. Table D–6 shows the possible values for the SQLCHAR_SET_NAME field when the SQLTYPE indicates one of the character data types.
	A 128-byte subfield of SQLVAR2 whose value is the character set name if SQLTYPE is a character string type, and spaces if SQLTYPE is any other data type.		
SQLCHAR_SET_ SCHEMA	A 128-byte subfield of SQLVAR2 whose value is the character set of the schema name if SQLTYPE is a character string type, and spaces if SQLTYPE is any other data type.	Reserved for future use.	Reserved for future use.
SQLCHAR_SET_ CATALOG	A 128-byte subfield of SQLVAR2 whose value is the character set of the catalog name if SQLTYPE is a character string type, and spaces if SQLTYPE is any other data type.	Reserved for future use.	Reserved for future use.

(continued on next page)

Table D–3 (Cont.) Fields in the SQLDA2

SQLVAR2 Subfields (Each Occurs Once for Each Select List Item or Parameter Marker):			
Field Name	Meaning of the Field	Set by	Used by
SQLDATA	A subfield of SQLVAR2 whose value is the address of the storage allocated for the select list item or parameter marker. Use SQLOCTET_LEN to allocate memory for this pointer.	Program.	SQL: <ul style="list-style-type: none"> In EXECUTE and OPEN statements, to retrieve a value stored by the program and substitute it for a parameter marker in the prepared statement. In FETCH statements, to store a value from a result table.
SQLIND	A subfield of SQLVAR2 whose value is the address of a longword indicator variable, a longword (32 bits) in size (if the program does not set an indicator variable, the value is 0).	Program.	Program or SQL: <ul style="list-style-type: none"> In FETCH statements, by SQL to store the value for an indicator variable associated with a select list item. After FETCH statements, by program to retrieve the value of a select list item's associated indicator variable. In EXECUTE and OPEN statements, by SQL to retrieve the value of a parameter marker's associated indicator variable.

(continued on next page)

Table D–3 (Cont.) Fields in the SQLDA2

SQLVAR2 Subfields (Each Occurs Once for Each Select List Item or Parameter Marker):			
Field Name	Meaning of the Field	Set by	Used by
SQLNAME	<p>A varying character string subfield of SQLVAR2 whose value is:</p> <ul style="list-style-type: none"> For select list items, the name of the column in the select list of the prepared SELECT statement. For parameter markers, the name of the column to which a parameter marker is assigned (in INSERT or UPDATE statements) or compared (in basic predicates). <p>If the select list item, assignment, or comparison involves an arithmetic expression or predicates other than basic predicates, SQL does not assign a value to SQLNAME.</p>	SQL.	Program, optionally, to find out the name of the column associated with a select list item or parameter marker.
SQLNAME_LEN	A subfield of SQLVAR2 whose value is the length in octets of the column named by SQLNAME.		

Table D–4 shows the possible values for the SQLLEN field when the SQLTYPE indicates one of the interval data types.

Table D–4 Codes for Interval Qualifiers in the SQLDA2

Code	Interval Qualifier	Interval Subtype
1	YEAR	SQLDA2_DT_YEAR
2	MONTH	SQLDA2_DT_MONTH
3	DAY	SQLDA2_DT_DAY
4	HOURL	SQLDA2_DT_HOURL
5	MINUTE	SQLDA2_DT_MINUTE
6	SECOND	SQLDA2_DT_SECOND
7	YEAR TO MONTH	SQLDA2_DT_YEAR_MONTH
8	DAY TO HOURL	SQLDA2_DT_DAY_HOURL
9	DAY TO MINUTE	SQLDA2_DT_DAY_MINUTE
10	DAY TO SECOND	SQLDA2_DT_DAY_SECOND
11	HOURL TO MINUTE	SQLDA2_DT_HOURL_MINUTE
12	HOURL TO SECOND	SQLDA2_DT_HOURL_SECOND
13	MINUTE TO SECOND	SQLDA2_DT_MINUTE_SECOND

Table D–5 shows the possible values for the SQLCHRONO_SCALE field when SQLTYPE indicates the data type DATE, DATE ANSI, DATE VMS, TIME or TIMESTAMP.

Table D–5 Codes for Date-Time Data Types in the SQLDA2

Code	Date-Time Data Type	Date-Time Subtypes
1	DATE ANSI	SQLDA2_DT_DATE
2	TIME	SQLDA2_DT_TIME
3	TIMESTAMP	SQLDA2_DT_TIMESTAMP
4	TIME WITH TIME ZONE	SQLDA2_DT_TIME_TZ
5	TIMESTAMP WITH TIME ZONE	SQLDA2_DT_TIMESTAMP_TZ

Table D–6 shows the possible values for the SQLCHAR_SET_NAME field when the SQLTYPE indicates one of the character data types.

Table D–6 Values for the SQLCHAR_SET_NAME Field

Character Set Value	Description
DEFAULT	Database default character set
NATIONAL	National character set
UNSPECIFIED	The character set is unspecified. SQL does not check for compatibility of data.
name-of-cset	See Table 2-1 in Volume 1 for a list of supported character set names.

E

Logical Names Used by SQL

Table E-1 lists the logical names that SQL recognizes for special purposes.

Table E-1 Summary of SQL Logical Names

Logical Name	Function
RDB\$CHARACTER_SET	<p>Specifies the database default and national character sets in addition to the session default, identifier, literal, and national character sets. Table E-2 shows the valid equivalence names for the logical name.</p> <p>The logical name is used by the EXPORT and IMPORT statements and by the SQL precompiler and SQL module language to allow compatibility of most recent versions with earlier versions of Oracle Rdb. This logical name sets the attributes for the default connection.</p> <p>This logical name is also deprecated and will not be supported in a future release.</p>
RDB\$ROUTINES	<p>Specifies the location of an external routine image. If you do not specify a location clause in a CREATE FUNCTION, CREATE PROCEDURE, or CREATE MODULE statement, or if you specify the DEFAULT LOCATION clause, SQL uses the RDB\$ROUTINES logical name as the default image location.</p>
RDMS\$BIND_OUTLINE_MODE	<p>When multiple outlines exist for a query, this logical name is defined to select which outline to use.</p>
RDMS\$BIND_QG_CPU_TIMEOUT	<p>Specifies the amount of CPU time used to optimize a query for execution.</p>
RDMS\$BIND_QG_REC_LIMIT	<p>Specifies the number of rows that SQL fetches before the query governor stops output.</p>
RDMS\$BIND_QG_TIMEOUT	<p>Specifies the number of seconds that SQL spends compiling a query before the query governor aborts that query.</p>
RDMS\$BIND_SEGMENTED_STRING_BUFFER	<p>Allows you to reduce the overhead of I/O operations at run time when you are manipulating a segmented string.</p>
RDMS\$DEBUG_FLAGS	<p>Allows you to examine database access strategies and the estimated cost of those strategies when your program runs.</p>

(continued on next page)

Table E–1 (Cont.) Summary of SQL Logical Names

Logical Name	Function
RDMS\$SET_FLAGS	Allows you to examine database access strategies and the estimated cost of those strategies when your program runs. See the SET FLAGS Statement for a list of valid keywords that can be used with this logical name.
RDMS\$DIAG_FLAGS	When defined to 'L', prevents the opening of a scrollable list cursor when the online format of lists is chained.
RDMS\$RTX_SHRMEM_PAGE_CNT	Specifies the size of the shared memory area used to manipulate server site-bound, external routine parameter data and control data.
RDMS\$USE_OLD_CONCURRENCY	Allows applications to use the isolation-level behavior that was in effect for V4.1.
RDMS\$USE_OLD_SEGMENTED_STRING	When defined to YES, the default online format for lists (segmented strings) is chained.
RDMS\$VALIDATE_ROUTINE	Controls the validation of routines.
SQL\$DATABASE	Specifies the database that SQL declares if you do not explicitly declare a database.
SQL\$DISABLE_CONTEXT	Disables the two-phase commit protocol. Useful for turning off distributed transactions when you want to run batch-update transactions.
SQL\$EDIT	Specifies the editor that SQL invokes when you issue the EDIT statement in interactive SQL. See the EDIT Statement for details.
SQLINI	Specifies the command file that SQL executes when you invoke interactive SQL.
SY\$CURRENCY	Specifies the character that SQL substitutes for the dollar sign (\$) symbol in an EDIT STRING clause of a column or domain definition, or the EDIT USING clause of a SELECT statement.
SY\$DIGIT_SEP	Specifies the character that SQL substitutes for the comma symbol (,) in an EDIT STRING clause of a column or domain definition, or the EDIT USING clause of a SELECT statement.
SY\$LANGUAGE	Specifies the language that SQL uses for date and time input and displays, or the EDIT USING clause of a SELECT statement.
SY\$RADIX_POINT	Specifies the character that SQL substitutes for the decimal point symbol (.) in an EDIT STRING clause of a column or domain definition, or the EDIT USING clause of a SELECT statement.

Table E–2 shows the valid equivalence names for the logical name RDB\$CHARACTER_SET.

Table E–2 Valid Equivalence Names for RDB\$CHARACTER_SET Logical Name

Character Set	Name of Character Set	Equivalence Name
MCS	DEC_MCS	Undefined
Korean and ASCII	DEC_KOREAN	DEC_HANGUL
Hanyu and ASCII	DEC_HANYU	DEC_HANYU
Hanzi and ASCII	DEC_HANZI	DEC_HANZI
Kanji and ASCII	DEC_KANJI	DEC_KANJI

For more information on these and other logical names, see the *Oracle Rdb7 Guide to Database Performance and Tuning*.

Obsolete SQL Syntax

This appendix describes:

- Incompatible syntax

Certain SQL statements that were allowed in earlier versions of SQL now have different behavior that is incompatible with earlier versions. *You must modify existing applications.*

- Deprecated syntax

Certain SQL statements that were allowed in earlier versions of SQL will be identified (flagged) with diagnostic messages. SQL refers to such statements as deprecated features. Although these statements will process with expected behavior for this release, SQL may not support them in future versions. You should replace deprecated syntax with the new syntax in applications.

- Reserved words deprecated as identifiers

If any of the listed reserved words is used as an identifier without double quotation marks ("), SQL flags the usage as being noncompliant with the ANSI/ISO standard and issues a deprecated feature message.

- Punctuation changes

This section describes changes to punctuation marks used in SQL.

- Suppressing diagnostic messages

This section describes how to suppress the diagnostic messages about deprecated features.

F.1 Incompatible Syntax

The following sections describe incompatible syntax.

F.1.1 Incompatible Syntax Containing the SCHEMA Keyword

Because one database may contain multiple schemas, the following incompatible changes apply to SQL syntax containing the SCHEMA keyword.

F.1.1.1 CREATE SCHEMA Meaning Incompatible

Use of the CREATE SCHEMA statement to create a database is deprecated. If you use the CREATE SCHEMA statement to specify the physical attributes of a database such as the root file parameters, SQL issues the deprecated feature message and interprets the statement as it did in previous versions of SQL.

```
SQL> CREATE SCHEMA PARTS SNAPSHOT IS ENABLED;  
%SQL-I-DEPR_FEATURE, Deprecated Feature: SCHEMA (meaning DATABASE)  
SQL>
```

However, if you do not specify any physical attributes of a database, you must enable multischema naming to use the CREATE SCHEMA statement.

```
SQL> CREATE SCHEMA PARTS;  
%SQL-F-SCHCATMULTI, Schemas and catalogs may only be referenced with  
multischema enabled
```

When you enable multischema naming, the CREATE SCHEMA statement creates a new schema within the current catalog.

```
SQL> ATTACH 'ALIAS Q4 FILENAME INVENTORY MULTISCHEMA IS ON';  
SQL> CREATE SCHEMA PARTS;  
SQL> SHOW SCHEMAS;  
Schemas in database with alias Q4  
RDB$SCHEMA  
PARTS
```

F.1.1.2 SHOW SCHEMA Meaning Incompatible

If you use a SHOW SCHEMA statement when you are attached to a database with the multischema attribute and have multischema naming enabled, SQL shows all the schemas for the current catalog. To show a database, use the SHOW DATABASE or SHOW ALIAS statement.

If you use a SHOW SCHEMA statement when you do not have multischema enabled, SQL issues an error message.

F.1.1.3 DROP SCHEMA Meaning Incompatible

If you use a DROP SCHEMA statement when you are attached to a database with the multischema attribute and have multischema naming enabled, SQL deletes the named schema from that database.

If you use a DROP SCHEMA statement when you do not have multischema enabled, SQL issues an error message.

If you use a DROP SCHEMA FILENAME statement, SQL interprets this as it would have in V4.0 and prior versions; it deletes the database with the named file name, and issues a deprecated feature error message.

F.1.2 DROP TABLE Now Restricts by Default

In V4.1 and higher, the default behavior of the DROP TABLE statement is a restricted delete, not a cascading delete as in earlier versions. Only the table will be deleted. If other items (views, constraints, indexes, or triggers) refer to the specified table, the delete will fail, as shown in the following example:

```
SQL> DROP TABLE DEGREES;
%RDB-E-NO_META_UPDATE, metadata update failed
-RDMS-F-TRGEXI, relation DEGREES is referenced in trigger
COLLEGE_CODE_CASCADE_UPDATE
-RDMS-F-RELNOTDEL, relation DEGREES has not been deleted
```

If you specify the CASCADE keyword for SQL DROP TABLE statements, SQL deletes all items that refer to the table or view, then deletes the table itself.

The following example shows a cascading delete:

```
SQL> DROP TABLE JOB_HISTORY CASCADE;
View CURRENT_INFO is also being dropped.
View CURRENT_JOB is also being dropped.
Constraint JOB_HISTORY_FOREIGN1 is also being dropped.
Constraint JOB_HISTORY_FOREIGN2 is also being dropped.
Constraint JOB_HISTORY_FOREIGN3 is also being dropped.
Index JH_EMPLOYEE_ID is also being dropped.
Index JOB_HISTORY_HASH is also being dropped.
VIA clause on storage map JOB_HISTORY_MAP is also being dropped.
Trigger EMPLOYEE_ID_CASCADE_DELETE is also being dropped.
```

F.1.3 Database Handle Names Restricted to 25 Characters

The database handle name is called an alias in SQL. When sessions are enabled by the OPTIONS=(CONNECT) qualifier on the SQL precompiler command line or the CONNECT qualifier on the module language command line, the length of an alias can be no more than 25 characters. The database handle was called an authorization identifier in versions of SQL prior to V4.1.

F.1.4 Deprecated Default Semantics of the ORDER BY Clause

In V4.1 and previous versions, SQL had the following default semantics:

- The ANSI/ISO 1989 standard provides a different direction. In future releases, SQL will assign the sort order of ASC to any key not specifically qualified with DESC.

- SQL will issue a deprecated feature warning if any sort keys inherit the DESC qualifier.

Note

If you do not specify ASC or DESC for the second or subsequent sort keys, SQL uses the order you specified for the preceding sort keys. If you do not specify the sorting order with the first sort key, the default order is ascending.

F.1.5 Change to EXTERNAL NAMES IS Clause

The multischema EXTERNAL NAME IS clause has changed to the STORED NAME IS clause to avoid confusion with ANSI/ISO SQL standards.

F.2 Deprecated Syntax

Table F–1 lists SQL statements that have been replaced by new syntax. These statements will be allowed by SQL, but in some cases SQL flags the statement with a deprecated feature message.

Table F–1 Deprecated Syntax for SQL

Deprecated Statement	New Syntax	Deprecated Feature Message?
ALTER CACHE ... LARGE MEMORY IS ENABLED	ALTER CACHE ... SHARED MEMORY IS PROCESS RESIDENT	
ALTER CACHE ... SHARED MEMORY IS SYSTEM	ALTER CACHE ... SHARED MEMORY IS PROCESS RESIDENT	
ALTER CACHE ... WINDOW COUNT IS ...	None	
ALTER SCHEMA	ALTER DATABASE	Yes
CREATE CACHE ... LARGE MEMORY IS ENABLED	CREATE CACHE ... SHARED MEMORY IS PROCESS RESIDENT	
CREATE CACHE ... SHARED MEMORY IS SYSTEM	CREATE CACHE ... SHARED MEMORY IS PROCESS RESIDENT	
CREATE CACHE ... WINDOW COUNT IS ...	None	
CREATE SCHEMA	CREATE DATABASE	Yes ¹

¹See Section F.1 for more information.

(continued on next page)

Table F–1 (Cont.) Deprecated Syntax for SQL

Deprecated Statement	New Syntax	Deprecated Feature Message?
DECLARE SCHEMA — module language and precompiled SQL	DECLARE ALIAS	Yes
DECLARE SCHEMA — dynamic and interactive SQL	ATTACH	In interactive SQL, but not in dynamic SQL
DECLARE and SET TRANSACTION — CONSISTENCY LEVEL 2, 3	ISOLATION LEVEL READ COMMITTED ISOLATION LEVEL REPEATABLE READ ISOLATION LEVEL SERIALIZABLE	Yes
DROP SCHEMA FILENAME	DROP DATABASE FILENAME	Message only in precompiled SQL and SQL module language
DROP SCHEMA PATHNAME	DROP DATABASE PATHNAME	Message only in precompiled SQL and SQL module language
DROP SCHEMA AUTHORIZATION	DROP DATABASE ALIAS	Message only in precompiled SQL and SQL module language
EXPORT SCHEMA FILENAME	EXPORT DATABASE FILENAME	No
EXPORT SCHEMA PATHNAME	EXPORT DATABASE PATHNAME	No
EXPORT SCHEMA AUTHORIZATION	EXPORT DATABASE ALIAS	No
FINISH	DISCONNECT DEFAULT	Yes, if databases are declared with DECLARE SCHEMA; otherwise, error message on nonconforming usage
GRANT on SCHEMA AUTHORIZATION	GRANT ON DATABASE ALIAS	Yes
IMPORT SCHEMA AUTHORIZATION	IMPORT DATABASE FROM filespec WITH ALIAS	Yes
INTEGRATE	INTEGRATE DATABASE	Yes
PREPARE . . . SELECT LIST	DESCRIBE . . . SELECT LIST	Yes
REVOKE	REVOKE ON DATABASE ALIAS	Yes
SET ANSI	SET DEFAULT DATE FORMAT SET KEYWORD RULES SET QUOTING RULES SET VIEW UPDATE RULES	No
ALTER DATABASE . . . JOURNAL IS . . . [NO]CACHE FILENAME . . .	None	Yes. Functionality no longer provides benefit on new hardware

(continued on next page)

Table F–1 (Cont.) Deprecated Syntax for SQL

Deprecated Statement	New Syntax	Deprecated Feature Message?
ALTER DATABASE . . . JOURNAL IS . . . NOTIFY	CREATE or ALTER DATABASE NOTIFY	Yes. Feature no longer part of the Alter image journaling functionality.
WRITE ONCE storage area attribute	None	Yes. Functionality is no longer available in hardware
VARIANT	NOT DETERMINISTIC	No. New syntax conforms to SQL:1999 Language Standard
NOT VARIANT	DETERMINISTIC	No. New syntax conforms to SQL:1999 Language Standard
GENERAL PARAMETER STYLE	PARAMETER STYLE GENERAL	No. New syntax conforms to SQL:1999 Language Standard
WHILE . . . LOOP . . . END LOOP	WHILE . . . DO . . . END WHILE	No. New syntax conforms to SQL:1999 Language Standard

F.2.1 Command Line Qualifiers

Certain qualifiers in the SQL module language and precompiler command lines have been replaced. These are:

- The ANSI_AUTHORIZATION qualifier is replaced by the RIGHTS clause.
- The ANSI_DATE qualifier is replaced by the DEFAULT DATE FORMAT clause.
- The ANSI_IDENTIFIERS qualifier is replaced by the KEYWORD RULES clause.
- The ANSI_PARAMETERS qualifier is replaced by the PARAMETER COLONS clause.
- The ANSI_QUOTING qualifier is replaced by the QUOTING RULES clause.

F.2.2 Deprecated Interactive SQL Statements

If you use the SET ANSI statement, SQL returns a deprecated feature message. This statement has been replaced by:

- The SET ANSI DATE statement is replaced by the SET DEFAULT DATE FORMAT statement. See the SET DEFAULT DATE FORMAT Statement for more information.
- The SET ANSI IDENTIFIERS statement is replaced by the SET KEYWORD RULES statement. See the SET KEYWORD RULES Statement for more information.

- The SET ANSI QUOTING statement is replaced by the SET QUOTING RULES statement. See the SET QUOTING RULES Statement for more information.

F.2.3 Constraint Conformance to the ANSI/ISO SQL Standard

The location of the constraint name in the CREATE TABLE statement has been changed for ANSI/ISO SQL conformance. Constraint names are expected before the constraint rather than after. If you place a constraint name after the constraint, you get the following deprecated feature message:

```
SQL> CREATE TABLE TEMP2
cont> (COL1 REAL NOT NULL CONSTRAINT C7);
%SQL-I-DEPR_FEATURE, Deprecated Feature: Constraint name clause following
constraint definition
%SQL-I-DEPR_FEATURE, Deprecated Feature: Default evaluation for constraints:
DEFERRABLE
```

The default evaluation time of DEFERRABLE for constraints has been deprecated. If your dialect is SQLV40, constraints are still DEFERRABLE by default. However, you will receive the following deprecated feature message if you do not specify an evaluation time:

```
SQL> CREATE TABLE TEMP3
cont> (COL1 REAL CONSTRAINT C6 NOT NULL);
%SQL-I-DEPR_FEATURE, Deprecated Feature: Default evaluation for constraints:
DEFERRABLE
```

If your dialect is SQL92 or SQL99, constraints are NOT DEFERRABLE by default, and you do not receive deprecated feature messages.

F.2.4 Obsolete Keywords

Table F–2 lists obsolete keywords and preferred substitutes for SQL statements.

Table F–2 Obsolete SQL Keywords

Obsolete Keyword	Preferred Keyword
COMMIT_TIME	COMMIT TIME
CREATETAB	CREATE
DIAGNOSTIC	CONSTRAINT
QUADWORD	BIGINT

(continued on next page)

Table F–2 (Cont.) Obsolete SQL Keywords

Obsolete Keyword	Preferred Keyword
READ_ONLY	READ ONLY
READ_WRITE	READ WRITE
VERB_TIME	VERB TIME

If you use the obsolete keywords, you receive the following diagnostic message:

```
SET TRANSACTION READ_ONLY;  
1  
%SQL-I-DEPR_FEATURE, (1) Deprecated Feature: READ_ONLY
```

F.2.5 Obsolete Built-in Functions

Several functions that were supplied as SQL or external routines in the SYSSLIBRARY:SQL_FUNCTIONS library are now obsolete and have been replaced with native SQL builtin functions.

- The function ABS was provided as an SQL function that accepted a DOUBLE PRECISION argument and returned a DOUBLE PRECISION result.
The native SQL function supports a wider range of data types (numeric and INTERVAL types) and is more generally useful.
- The functions LEAST and GREATEST were provided as SQL functions that accepted only two integer arguments.
The native SQL functions allow for a wider range of data types, and support a parameter list of arbitrary length.
- The functions LENGTH and LENGTHB were provided as SQL stored functions that accepted a VARCHAR (2000) parameter and performed the appropriate CHARACTER_LENGTH or OCTET_LENGTH operation on the argument.
The native SQL functions allow for a wider range of character set values and larger sizes for the character data types.
- The function SIGN was provided as a SQL function.
The native SQL function is more generally useful.
- The functions ROUND and TRUNC were provided as SQL functions that accepted a DOUBLE PRECISION argument and returned a DOUBLE PRECISION result.

The native SQL functions support a wider range of data types (TINYINT, SMALLINT, INTEGER, BIGINT, and FLOAT) and are more generally useful.

Note that the ROUND or TRUNC applied to date/time types (DATE, DATE VMS, TIMESTAMP) requires the use of OCI support libraries provided by OCI Services for Rdb.

Now that SQL implements these functions directly these definitions in SYSSLIBRARY:SQL_FUNCTIONS are no longer required. However, they are retained in the database for existing applications but new applications will now automatically use new native functions in Oracle Rdb.

F.3 Deprecated Logical Names

The following sections describe deprecated logical names and, if applicable, the logical name replacement.

See Appendix E for more information regarding any new logical names.

F.3.1 RDB\$CHARACTER_SET Logical Name

The logical name RDB\$CHARACTER_SET has been deprecated. It is used by SQL to allow compatibility for databases and applications from V4.1 and V4.0.

When you are using versions higher than V4.1 and V4.0, Oracle Rdb recommends that you use the following clauses and statements in place of the logical name:

- The DEFAULT CHARACTER SET and NATIONAL CHARACTER SET clauses in the DECLARE ALIAS statement.
- The IDENTIFIER CHARACTER SET, DEFAULT CHARACTER SET, and NATIONAL CHARACTER SET clauses of the SQL module header (Section 3.2) or the DECLARE MODULE statement.
- The SET IDENTIFIER CHARACTER SET statement, SET DEFAULT CHARACTER SET statement, and the SET NATIONAL CHARACTER SET statement for dynamic SQL.
- The IDENTIFIER CHARACTER SET, DEFAULT CHARACTER SET, and NATIONAL CHARACTER SET clauses in the CREATE DATABASE statement or the ALTER DATABASE statement.

F.4 Reserved Words Deprecated as Identifiers

The following lists contain reserved words from the:

- ANSI/ISO 1989 SQL standard
- ANSI/ISO 1992 SQL standard
- ANSI/ISO 1999 SQL standard

If these reserved words are used as identifiers without double quotation marks ("), SQL flags their use as being noncompliant with the ANSI/ISO 1989 standard and issues a deprecated feature message.

Oracle Rdb does not recommend using reserved words as identifiers because this capability is a deprecated feature and might not be supported in future versions of SQL. However, if you must use reserved words as identifiers, then you must enclose them within double quotation marks to be compliant with the ANSI/ISO 1989 standard. SQL does not allow lowercase letters, spaces, or tab stops within the double quotation marks.

For example, if you want to use the ANSI/ISO 1989 reserved word SELECT as a table identifier, the statement would be written as follows:

```
SELECT * FROM "SELECT";
```

F.4.1 ANSI/ISO 1989 SQL Standard Reserved Words

ALL	AND	ANY
AS	ASC	AUTHORIZATION
AVG	BEGIN	BETWEEN
BY	CHAR	CHARACTER
CHECK	CLOSE	COBOL
COMMIT	CONTINUE	COUNT
CREATE	CURRENT	CURSOR
DEC	DECIMAL	DECLARE
DEFAULT	DELETE	DESC
DISTINCT	DOUBLE	END
ESCAPE	EXEC	EXISTS
FETCH	FLOAT	FOR
FOREIGN	FORTRAN	FOUND
FROM	GO	GOTO

GRANT	GROUP	HAVING
IN	INDICATOR	INSERT
INT	INTEGER	INTO
IS	KEY	LANGUAGE
LIKE	MAX	MIN
MODULE	NOT	NULL
NUMERIC	OF	ON
OPEN	OPTION	OR
ORDER	PASCAL	PLI
PRECISION	PRIMARY	PRIVILEGES
PROCEDURE	PUBLIC	REAL
REFERENCES	ROLLBACK	SCHEMA
SECTION	SELECT	SET
SMALLINT	SOME	SQL
SQLCODE	SQLERROR	SUM
TABLE	TO	UNION
UNIQUE	UPDATE	USER
VALUES	VIEW	WHENEVER
WHERE	WITH	WORK

F.4.2 ANSI/ISO 1992 SQL Standard Reserved Words

In addition to the reserved words listed for the ANSI/ISO 1989 standard, the ANSI/ISO SQL standard also includes the following reserved words:

ABSOLUTE	ACTION	ADD
ALLOCATE	ALTER	ARE
ASSERTION	AT	BIT
BIT_LENGTH	BOTH	CASCADE
CASCADE	CASE	CAST
CATALOG	CHAR_LENGTH	CHARACTER_LENGTH
COALESCE	COLLATE	COLLATION
COLUMN	CONNECT	CONNECTION
CONSTRAINT	CONSTRAINTS	CONVERT
CORRESPONDING	CROSS	CURRENT_DATE

CURRENT_TIME	CURRENT_TIMESTAMP	CURRENT_USER
DATE	DAY	DEALLOCATE
DEFERRABLE	DEFERRED	DESCRIBE
DESCRIPTOR	DIAGNOSTICS	DISCONNECT
DOMAIN	DROP	ELSE
END-EXEC	EXCEPT	EXCEPTION
EXECUTE	EXTERNAL	EXTRACT
FALSE	FIRST	FULL
GET	GLOBAL	HOUR
IDENTITY	IMMEDIATE	INITIALLY
INNER	INPUT	INSENSITIVE
INTERSECT	INTERVAL	ISOLATION
JOIN	LAST	LEADING
LEFT	LEVEL	LOCAL
LOWER	MATCH	MINUTE
MONTH	NAMES	NATIONAL
NATURAL	NCHAR	NEXT
NO	NULLIF	OCTET_LENGTH
ONLY	OUTER	OUTPUT
OVERLAPS	PAD	PARTIAL
POSITION	PREPARE	PRESERVE
PRIOR	READ	RELATIVE
RESTRICT	REVOKE	RIGHT
ROWS	SCROLL	SECOND
SESSION	SESSION_USER	SIZE
SPACE	SQLSTATE	SUBSTRING
SYSTEM_USER	TEMPORARY	THEN
TIME	TIMESTAMP	TIMEZONE_HOUR
TIMEZONE_MINUTE	TRAILING	TRANSACTION
TRANSLATE	TRANSLATION	TRIM
TRUE	UNKNOWN	UPPER

USAGE	USING	VALUE
VARCHAR	VARYING	WHEN
WRITE	YEAR	ZONE

F.4.3 ANSI/ISO 1999 SQL Standard Reserved Words

In addition to the reserved words listed for the ANSI/ISO 1989 standard and the ANSI/ISO SQL 1992 standard, the ANSI/ISO SQL 1999 standard includes the following reserved words.

ADMIN	AFTER	AGGREGATE
ALIAS	ARRAY	BEFORE
BINARY	BLOB	BOOLEAN
BREADTH	CALL	CLASS
CLOB	COMPLETION	CONDITION
CONSTRUCTOR	CUBE	CURRENT_PATH
CURRENT_ROLE	CYCLE	DATA
DEPTH	DEREF	DESTROY
DESTRUCTOR	DETERMINISTIC	DICTIONARY
DO	DYNAMIC	EACH
ELSEIF	EQUALS	EVERY
EXIT	FREE	FUNCTION
GENERAL	GROUPING	HANDLER
HOST	IF	IGNORE
INITIALIZE	INOUT	ITERATE
LARGE	LATERAL	LEAVE
LESS	LIMIT	LIST
LOCALTIME	LOCALTIMESTAMP	LOCATOR
LONG	LOOP	MAP
MODIFIES	MODIFY	NCLOB
NEW	NONE	NUMBER
OBJECT	OFF	OLD
OPERATION	ORDINALITY	OUT
PARAMETER	PARAMETERS	PATH
POSTFIX	PREFIX	PREORDER

RAW	READS	RECURSIVE
REDO	REF	REFERENCING
REPEAT	RESIGNAL	RESULT
RETURN	RETURNS	ROLE
ROLLUP	ROUTINE	ROW
SAVEPOINT	SCOPE	SEARCH
SENSITIVE	SEQUENCE	SETS
SIGNAL	SIMILAR	SPECIFIC
SPECIFICTYPE	SQLEXCEPTION	SQLWARNING
START	STATEMENT	STATIC
STRUCTURE	TERMINATE	THAN
TREAT	TRIGGER	TYPE
UNDER	UNDO	UNNEST
UNTIL	VARIABLE	WHILE
WITHOUT		

F.4.4 Words From ANSI/ISO SQL3 Draft Standard No Longer Reserved

In previous releases, the following words were listed as reserved words according to the ANSI/ISO SQL3 draft standard but did not become part of the final ANSI/ISO 1999 SQL standard. The following words are no longer reserved by Oracle Rdb as previously documented:

ACTOR	ASync	ELEMENT
INSTEAD	MOVE	MULTISET
NEW_TABLE	OID	OLD_TABLE
OPERATORS	OTHERS	PENDANT
PRIVATE	PROTECTED	REPRESENTATION
TEMPLATE	TEST	THERE
TUPLE	VARIANT	VIRTUAL
VISIBLE	WAIT	

F.5 Punctuation Changes

The following changes apply to punctuation marks used in SQL.

F.5.1 Single Quotation Marks Required for String Literals

Use single (') instead of double (") quotation marks to delimit a string literal. SQL flags literals enclosed within double quotation marks with an informational, compile-time, diagnostic message stating that this is nonstandard usage. This message will appear even when you have specified that SQL not notify you of syntax that is not ANSI/ISO SQL standard.

F.5.2 Double Quotation Marks Required for ANSI/ISO SQL Delimited Identifiers

The leftmost name pair in a qualified name for a multischema object is a **delimited identifier**. You must enclose a delimited identifier within double quotation marks and use only uppercase characters. You must enable ANSI/ISO SQL quoting rules to use delimited identifiers. For more information, see Section 2.2.11.

F.5.3 Colons Required Before Host Language Variables in SQL Module Language

In SQL module language statements, Oracle Rdb recommends that you precede parameters with a colon (:) to distinguish them from column or table names. These colons are currently optional in SQL, but are required by the ANSI/ISO SQL standard. SQL may require these colons in a future version of Oracle Rdb.

F.6 Suppressing Diagnostic Messages

In interactive SQL, use the SET WARNING NODEPRECATE statement to suppress the diagnostic messages about deprecated features. For more information, see the SET Statement.

If you are using the SQL precompiler, you can suppress the diagnostic messages about deprecated features by using the SQLOPTIONS=WARN=(NODEPRECATE) qualifier in the precompiler command line. For details, see Section 4.3.

If you are using SQL module language, you can suppress the diagnostic messages about deprecated features by using the WARN=(NODEPRECATE) qualifier in the module language command line. For details, see Section 3.6.

Oracle Database Compatibility

G.1 Oracle Database Functions

SQL functions have been added to the OpenVMS Oracle Rdb SQL interface for compatibility with Oracle SQL. Complete descriptions of these functions can be found in the *Oracle Database SQL Language Reference Manual*.

G.1.1 Optional Oracle SQL Functions

Optionally, you can install the functions listed in Table G–1 in your database from interactive SQL as shown in the following examples.

The file is named `SQL_FUNCTIONSnn.SQL`, where “nn” is the version number. For example, use the following statement:

```
SQL> ATTACH 'FILENAME mydatabase';
SQL> @SYS$LIBRARY:SQL_FUNCTIONS72.SQL
```

If you wish to use a character set other than `DEC_MCS` with the installable functions, you must first define the `RDB$ORACLE_SQLFUNC_VCHAR_DOM` domain as a character type using the desired character set before executing the preceding statements. Similarly, if you wish to use a date data type other than `DATE VMS` with the installable functions, you must first define the `RDB$ORACLE_SQLFUNC_DATE_DOM` domain as a date data type before executing the preceding statements.

For example,

```
SQL> ATTACH 'FILENAME mydatabase';
SQL> CREATE DOMAIN RDB$ORACLE_SQLFUNC_VCHAR_DOM VARCHAR(2000)
cont> CHARACTER SET KANJI;
SQL> CREATE DOMAIN RDB$ORACLE_SQLFUNC_DATE_DOM DATE ANSI;
SQL> @SYS$LIBRARY:SQL_FUNCTIONS72.SQL
```

If you choose, you may remove the installable functions from your database at a later time. However, you must release any dynamic SQL statements and disconnect any sessions that reference any of these functions before you can remove the functions. Use the following statements from interactive SQL if you wish to remove the installable functions from your database:

```
SQL> ATTACH 'FILENAME mydatabase';
SQL> @SYS$LIBRARY:SQL_FUNCTIONS_DROP72.SQL
```

The file SYS\$LIBRARY:SQL_FUNCTIONS_DROPnn.SQL, where “nn” is the version number.

Table G–1 gives a brief description of each of the functions that you can optionally install in your database.

Table G–1 Optional Oracle SQL Functions

Function Name	Description	Restrictions
ADD_MONTHS (d,n)	Returns the date d plus n months.	d must be of the same date data type as the RDB\$ORACLE_SQLFUNC_DATE_DOM domain, which is bound when you install the Oracle SQL functions.
ACOS (n)	Returns the arc cosine of n.	n must be in the range of -1 to 1, and the function returns a DOUBLE PRECISION value in the range of 0 to pi, expressed in radians. If the passed expression results in NULL then the result of ACOS will be NULL.

The following example returns the arc cosine of .3:

```
SQL> SELECT ACOS(.3) "Arc_Cosine" FROM Rdb$DATABASE;
          Arc_Cosine
1.266103672779499E+000
1 row selected
```

(continued on next page)

Table G–1 (Cont.) Optional Oracle SQL Functions

Function Name	Description	Restrictions
ACOSH (n)	Returns the hyperbolic arc cosine of n.	n must be equal to or greater than 1. The function returns a DOUBLE PRECISION value. If either passed expression results in NULL then the result of ACOSH will be NULL.

```
SQL> SELECT ACOSH(1.0) "Hyperbolic Arc Cosine" FROM Rdb$DATABASE;
      Hyperbolic Arc Cosine
      0.000000000000000E+000
1 row selected
```

ASCII (str)	Returns the decimal representation of the first character of its argument.	str must be of the same character set as the RDB\$ORACLE_SQLFUNC_VCHAR_DOM domain, which is bound when you install the Oracle SQL functions.
-------------	--	--

ASIN (n)	Returns the arc sine of n.	n must be in the range of -1 to 1. The function returns a DOUBLE PRECISION value in the range of -pi/2 to pi/2, expressed in radians. If the passed expression results in NULL then the result of ASIN will be NULL.
----------	----------------------------	--

The following example returns the arc sine of .3:

```
SQL> SELECT ASIN(.3) "Arc_Sine" FROM Rdb$DATABASE;
      Arc_Sine
      3.046926540153975E-001
1 row selected
```

(continued on next page)

Table G–1 (Cont.) Optional Oracle SQL Functions

Function Name	Description	Restrictions
ASINH (n)	Returns the hyperbolic arc sine of n.	The function returns a DOUBLE PRECISION value. If either passed expression results in NULL then the result of ASINH will be NULL.

```
SQL> SELECT ASINH (-90.0) "Hyperbolic Arc Sine" FROM Rdb$DATABASE;
      Hyperbolic Arc Sine
      -5.192987713658941E+000
1 row selected
```

ATAN (n)	Returns the arc tangent of n.	n can be in an unbounded range and returns a value in the range of -pi/2 to pi/2, expressed in radians. If the passed expression results in NULL then the result of ATAN will be NULL.
----------	-------------------------------	--

The following example returns the arc tangent of .3:

```
SQL> SELECT ATAN(.3) "Arc_Tangent" FROM Rdb$DATABASE;
      Arc_Tangent
      2.914567944778671E-001
1 row selected
```

ATANH (n)	Returns the hyperbolic arc tangent of n (in radians).	n must be in the range of -1 to 1. The function returns a DOUBLE PRECISION value. If either passed expression results in NULL then the result of ATANH will be NULL.
-----------	---	--

(continued on next page)

Table G–1 (Cont.) Optional Oracle SQL Functions

Function Name	Description	Restrictions
<pre>SQL> SELECT ATANH(0.905148254) "Hyperbolic Arc Tangent" FROM Rdb\$DATABASE; Hyperbolic Arc Tangent 1.500000001965249E+000 1 row selected</pre>		
ATAN2 (n,m)	Returns the arc tangent of n and m.	n can be in an unbounded range and returns a value in the range of -pi to pi, depending on the signs of n and m, expressed in radians. ATAN2(n,m) is the same as ATAN(n/m). If either passed expression results in NULL then the result of ATAN2 will be NULL.
<pre>SQL> SELECT ATAN2(.3, .2) "Arc_Tangent2" FROM Rdb\$DATABASE; Arc_Tangent2 9.827937232473291E-001 1 row selected</pre>		
BITAND (expr1,expr2)	BITAND computes an AND operation on the bits of expr1 and expr2, both of which must resolve to integers, and returns an integer.	This function is commonly used with the DECODE function. If the passed expression results in NULL then the result of the BITAND will be null. (continued on next page)

Table G–1 (Cont.) Optional Oracle SQL Functions

Function Name	Description	Restrictions
---------------	-------------	--------------

Example 1: Checking bits in RDB\$FLAGS column

The first bit of the mask stored in the column RDB\$FLAGS of the table Rdb\$RELATIONS indicates that this relation is a view definition. This query displays the names of any views in the database.

```
SQL> -- Which objects in Rdb$RELATIONS are views?
SQL> select rdb$relation_name from rdb$relations where bitand(rdb$flags, 1) = 1;
```

```
RDB$RELATION_NAME
RDBVMS$COLLATIONS
RDBVMS$INTERRELATIONS
RDBVMS$PRIVILEGES
RDBVMS$RELATION_CONSTRAINTS
RDBVMS$RELATION_CONSTRAINT_FLDS
RDBVMS$STORAGE_MAPS
RDBVMS$STORAGE_MAP_AREAS
RDBVMS$TRIGGERS
8 rows selected
```

Example 2: Using BITAND with DECODE

This example uses the result of the BITAND in a DECODE list to display attributes of an Rdb database.

```
SQL> select
cont>      DECODE (BITAND (rdb$flags,1), 0, 'No Dictionary', 'Dictionary'),
cont>      DECODE (BITAND (rdb$flags,2), 0, 'ACL Style', 'ANSI Style'),
cont>      DECODE (BITAND (rdb$flags,64), 0, 'No Multischema', 'Multischema')
cont> from
cont>      Rdb$DATABASE;

No Dictionary  ACL Style  Multischema
1 row selected
SQL>
```

(continued on next page)

Table G–1 (Cont.) Optional Oracle SQL Functions

Function Name	Description	Restrictions
BITANDNOT (numeric-expression,numeric-expression)	BITANDNOT is used to clear bits in the first expression that are set in the second expression. First a bitwise NOT (BITNOT) is performed on the second numeric value expression and then a bitwise AND (BITAND) is performed of the first numeric value expression with the result.	If either of the passed expressions results in NULL, then the result of BITANDNOT will be NULL. Note that BITANDNOT is equivalent to BITAND (exp1, BITNOT (ex2)) but is more efficient.
BITNOT (numeric-expression)	BITNOT returns the bitwise NOT of the passed numeric value expression.	If the passed expression results in NULL, then the result of BITNOT will be NULL.
BITOR (numeric-expression, numeric-expression)	BITOR returns the bitwise OR of the passed numeric value expressions.	If either of the passed expressions results in NULL, then the result of BITOR will be NULL.
BITXOR (numeric-expression, numeric-expression)	BITXOR Returns the bitwise XOR of the passed numeric value expressions.	If either of the passed expressions results in NULL, then the result of BITXOR will be NULL.
CEIL (n)	Returns the smallest integer greater than or equal to n.	
CHR (n)	Returns the character having the binary equivalent to n.	The returned value is of type RDB\$ORACLE_SQLFUNC_VCHAR_DOM, the character set of which is bound when you install the Oracle SQL functions. In addition, only 1 octet (byte) of data is encoded.
COS (n)	Returns the cosine of n (an angle expressed in radians).	

(continued on next page)

Table G–1 (Cont.) Optional Oracle SQL Functions

Function Name	Description	Restrictions
COSH (n)	Returns the hyperbolic cosine of n (an angle expressed in radians).	
COT (n)	COT returns the cotangent of n.	The function returns a DOUBLE PRECISION value. If either passed expression results in NULL then the result of COT will be NULL.
<pre>SQL> SELECT COT (3.14159265358979/4) "Cotangent" FROM Rdb\$DATABASE; Cotangent 1.0000000000000002E+000 1 row selected</pre>		
EXP (n)	Returns e raised to the nth power (e=2.71828183 . . .).	
FLOOR (n)	Returns the largest integer equal to or less than n.	
HEXTORAW (str)	Converts its argument containing hexadecimal digits to a raw character value.	str must be of the same character set as the RDB\$ORACLE_SQLFUNC_VCHAR_DOM domain, which is bound when you install the Oracle SQL functions. The value returned is of type RDB\$ORACLE_SQLFUNC_VCHAR_DOM.

(continued on next page)

Table G–1 (Cont.) Optional Oracle SQL Functions

Function Name	Description	Restrictions
INITCAP (str)	Returns the string argument, with the first letter of each word in uppercase, all other letters in lowercase. Words are delimited by non-alphanumeric characters.	str must be of the same character set as the RDB\$ORACLE_SQLFUNC_VCHAR_DOM domain, which is bound when you install the Oracle SQL functions. The value returned is of type RDB\$ORACLE_SQLFUNC_VCHAR_DOM.
INSTR (s1,s2[,n[,m]])	Searches s1 beginning with its nth character and returns the character position of the mth occurrence of s2 or 0 if s2 does not occur m times. If n < 0, the search starts at the end of s1.	s1 and s2 must be of the same character set as the RDB\$ORACLE_SQLFUNC_VCHAR_DOM domain, which is bound when you install the Oracle SQL functions. If either n or m is omitted, they default to 1.
INSTRB (s1,s2[,n[,m]])	Searches s1 beginning with its nth octet and returns the octet position of the mth occurrence of s2 or 0 if s2 does not occur m times. If n < 0, the search starts at the end of s1.	s1 and s2 must be of the same character set as the RDB\$ORACLE_SQLFUNC_VCHAR_DOM domain, which is bound when you install the Oracle SQL functions. If either n or m is omitted, they default to 1.
LAST_DAY (d)	Returns the last day of the month that contains d.	d must be of the same date data type as the RDB\$ORACLE_SQLFUNC_DATE_DOM domain, which is bound when you install the Oracle SQL functions. The value returned is of type RDB\$ORACLE_SQLFUNC_DATE_DOM.

(continued on next page)

Table G–1 (Cont.) Optional Oracle SQL Functions

Function Name	Description	Restrictions
LN (n)	Returns the natural logarithm of n where n is greater than 0.	
LOG (m,n)	Returns the logarithm base m of n. The base m can be any positive number other than 0 or 1 and n can be any positive number.	
LPAD (s,l,p)	Returns s left-padded to length l with the sequence of characters in p. If s is longer than l, this function returns that portion of s that fits in l.	s and p must be of the same character set as the RDB\$ORACLE_SQLFUNC_VCHAR_DOM domain, which is bound when you install the Oracle SQL functions. The value returned is of type RDB\$ORACLE_SQLFUNC_VCHAR_DOM. There is no default for p as with Oracle.
LTRIM (s1[,s2])	Removes characters from the left of s1, with initial characters removed up to the first character not in s2.	s1 and s2 must be of the same character set as the RDB\$ORACLE_SQLFUNC_VCHAR_DOM domain, which is bound when you install the Oracle SQL functions. The value returned is of type RDB\$ORACLE_SQLFUNC_VCHAR_DOM. If omitted, s2 defaults to space.
MOD (m,n)	Returns the remainder of m divided by n. Returns m if n is 0.	

(continued on next page)

Table G–1 (Cont.) Optional Oracle SQL Functions

Function Name	Description	Restrictions
MONTHS_BETWEEN (d1,d2)	Returns the number of months between dates d1 and d2.	d1 and d2 must be of the same date data type as the RDB\$ORACLE_SQLFUNC_DATE_DOM domain, which is bound when you install the Oracle SQL functions.
NEW_TIME (d1,z1,z2)	Returns the date and time in time zone z2 when the date and time in time zone z1 is d. Time zones z1 and z2 can be: AST, ADT, BST, BDT, CST, CDT, EST, EDT, GMT, HST, HDT, MST, MDT, NST, PST, PDT, YST, or YDT.	d1 must be of the same date data type as the RDB\$ORACLE_SQLFUNC_DATE_DOM domain, which is bound when you install the Oracle SQL functions. z1 and z2 must be of the same character set as the RDB\$ORACLE_SQLFUNC_VCHAR_DOM domain, which is also bound when you install the Oracle SQL functions. The value returned is of type RDB\$ORACLE_SQLFUNC_DATE_DOM.

(continued on next page)

Table G–1 (Cont.) Optional Oracle SQL Functions

Function Name	Description	Restrictions
NEXT_DAY (d,dayname)	Returns the date of the first weekday named by dayname that is later than the date d.	d must be of the same date data type as the RDB\$ORACLE_SQLFUNC_DATE_DOM domain, which is bound when you install the Oracle SQL functions. dayname must be of the same character set as the RDB\$ORACLE_SQLFUNC_VCHAR_DOM domain, which is also bound when you install the Oracle SQL functions. The value returned is of type RDB\$ORACLE_SQLFUNC_DATE_DOM.
POWER (m,n)	Returns m raised to the nth power. The base m and the exponent n can be any number but if m is negative, n must be an integer.	
RAWTOHEX (str)	Converts its raw argument to a character value containing its hexadecimal equivalent.	str must be of the same character set as the RDB\$ORACLE_SQLFUNC_VCHAR_DOM domain, which is also bound when you install the Oracle SQL functions. The value returned is of type RDB\$ORACLE_SQLFUNC_VCHAR_DOM.

(continued on next page)

Table G–1 (Cont.) Optional Oracle SQL Functions

Function Name	Description	Restrictions
REPLACE (s1[,s2[,s3]])	Returns s1 with every occurrence of s2 replaced by s3.	s1, s2, and s3 must be of the same character set as the RDB\$ORACLE_SQLFUNC_VCHAR_DOM domain, which is bound when you install the Oracle SQL functions. The value returned is of type RDB\$ORACLE_SQLFUNC_VCHAR_DOM. If omitted, s2 and s3 default to an empty string.
RPAD (s[,l[,p]])	Returns s left-padded to length l with the sequence of characters in p. If s is longer than l, this function returns that portion of s that fits in l.	s and p must be of the same character set as the RDB\$ORACLE_SQLFUNC_VCHAR_DOM domain, which is also bound when you install the Oracle SQL functions. The value returned is of type RDB\$ORACLE_SQLFUNC_VCHAR_DOM. If omitted, p defaults to a space.
RTRIM (s1[,s2])	Returns s2 with final characters after the last character not in s2.	s1 and s2 must be of the same character set as the RDB\$ORACLE_SQLFUNC_VCHAR_DOM domain, which is also bound when you install the Oracle SQL functions. The value returned is of type RDB\$ORACLE_SQLFUNC_VCHAR_DOM. If omitted, s2 defaults to a space.
SIN (n)	Returns the sine of n (an angle expressed in radians).	

(continued on next page)

Table G–1 (Cont.) Optional Oracle SQL Functions

Function Name	Description	Restrictions
SINH (n)	Returns the hyperbolic sine of n (an angle expressed in radians).	
SQRT (n)	Returns the square root of n. The value of n cannot be negative. SQRT returns a double precision result.	
SUBSTR (s[,p[,l]])	Returns a portion of s, l characters long, beginning at character position p. If p is negative, SUBSTR counts backward from the end of s.	s must be of the same character set as the RDB\$ORACLE_SQLFUNC_VCHAR_DOM domain, which is also bound when you install the Oracle SQL functions. The value returned is of type RDB\$ORACLE_SQLFUNC_VCHAR_DOM. If omitted, l defaults to zero (0).
SUBSTRB (s[,p[,l]])	Same as SUBSTR, except p and l are expressed in octets (bytes) rather than characters.	s must be of the same character set as the RDB\$ORACLE_SQLFUNC_VCHAR_DOM domain, which is also bound when you install the Oracle SQL functions. The value returned is of type RDB\$ORACLE_SQLFUNC_VCHAR_DOM. If omitted, l defaults to zero (0).
TAN (n)	Returns the tangent of n (an angle expressed in radians).	
TANH (n)	Returns the hyperbolic tangent of n (an angle expressed in radians).	

G.2 Oracle Style Outer Join

Oracle Rdb supports the SQL Database Language Standard syntax for performing outer join between two or more tables, namely the LEFT, RIGHT, and FULL OUTER JOIN clauses. Oracle Rdb also supports alternative syntax and semantics that conform to those available in Oracle RDMS SQL language to enhance the compatibility between these two products. The special operator (+) can be placed in the WHERE clause to instruct SQL to join tables using outer join semantics.

An outer join extends the result of a simple join. An outer join returns all rows that satisfy the join condition and those rows from one table for which no rows from the other satisfy the join condition. Such rows are not returned by a simple join. To write a query that performs an outer join of tables A and B and returns all rows from A, apply the outer join operator (+) to all columns of B in the join condition. For all rows in A that have no matching rows in B, Oracle Rdb returns NULL for any select list expressions containing columns of B.

Outer join queries are subject to the following rules and restrictions:

- The (+) operator can appear only in the WHERE clause and can be applied only to a column of a table or view.
- If A and B are joined by multiple join conditions, you must use the (+) operator in all of these conditions. If you do not, Oracle Rdb will return only the rows resulting from a simple join, but without a warning or error to advise you that you do not have the results of an outer join.
- The (+) operator can be applied only to a column, not to an arbitrary expression. However, an arbitrary expression can contain a column marked with the (+) operator.
- A condition containing the (+) operator cannot be combined with another condition using the OR logical operator.
- A condition cannot use the IN comparison operator to compare a column marked with the (+) operator with an expression.
- A condition cannot compare any column marked with the (+) operator with a subquery.

If the WHERE clause contains a condition that compares a column from table B with a constant, the (+) operator must be applied to the column so that Oracle Rdb returns the rows from table A for which it has generated NULLs for this column. Otherwise Oracle Rdb will return only the results of a simple join.

In a query that performs outer joins of more than two pairs of tables, a single table can be the NULL-generated table for only one other table. For this reason, you cannot apply the (+) operator to columns of B in the join condition for A and B and the join condition for B and C.

G.2.1 Outer Join Examples

The examples in this section extend the results of this inner join (Equijoin) between EMP and DEPT tables.

```
SQL> SELECT ename, job, dept.deptno, dname
cont> FROM emp, dept
cont> WHERE emp.deptno = dept.deptno;
EMP.ENAME    EMP.JOB      DEPT.DEPTNO  DEPT.DNAME
King         President    10    Accounting
Blake        Manager     30    Sales
Clark        Manager     10    Accounting
Jones        Manager     20    Research
Ford         Analyst     20    Research
Smith        Clerk       20    Research
Allen        Salesman    30    Sales
Ward         Salesman    30    Sales
Martin       Salesman    30    Sales
Scott        Analyst     20    Research
Turner       Salesman    30    Sales
Adams        Clerk       20    Research
James        Clerk       30    Sales
Miller       Clerk       10    Accounting
14 rows selected
```

The following query uses an outer join to extend the results of this Equijoin example above:

```

SQL> SELECT ename, job, dept.deptno, dname
cont> FROM emp, dept
cont> WHERE emp.deptno (+) = dept.deptno;
  EMP.ENAME    EMP.JOB      DEPT.DEPTNO  DEPT.DNAME
King           President      10    Accounting
Clark          Manager       10    Accounting
Miller         Clerk         10    Accounting
Jones          Manager       20    Research
Ford           Analyst       20    Research
Smith          Clerk         20    Research
Scott          Analyst       20    Research
Adams          Clerk         20    Research
Blake          Manager       30    Sales
Allen          Salesman      30    Sales
Ward           Salesman      30    Sales
Martin         Salesman      30    Sales
Turner         Salesman      30    Sales
James          Clerk         30    Sales
NULL           NULL          40    Operations
15 rows selected

```

In this outer join, Oracle Rdb returns a row containing the OPERATIONS department even though no employees work in this department. Oracle Rdb returns NULL in the ENAME and JOB columns for this row. The join query in this example selects only departments that have employees.

The following query uses an outer join to extend the results of the preceding example:

```

SQL> SELECT ename, job, dept.deptno, dname
cont> FROM emp, dept
cont> WHERE emp.deptno (+) = dept.deptno
cont> AND job (+) = 'Clerk';
  EMP.ENAME    EMP.JOB      DEPT.DEPTNO  DEPT.DNAME
Miller         Clerk         10    Accounting
Smith          Clerk         20    Research
Adams          Clerk         20    Research
James          Clerk         30    Sales
NULL           NULL          40    Operations
5 rows selected

```

In this outer join, Oracle Rdb returns a row containing the OPERATIONS department even though no clerks work in this department. The (+) operator on the JOB column ensures that rows for which the JOB column is NULL are also returned. If this (+) were omitted, the row containing the OPERATIONS department would not be returned because its JOB value is not 'CLERK'.

```

SQL> SELECT ename, job, dept.deptno, dname
cont> FROM emp, dept
cont> WHERE emp.deptno (+) = dept.deptno
cont> AND job = 'Clerk';
  EMP.ENAME  EMP.JOB    DEPT.DEPTNO  DEPT.DNAME
  Miller      Clerk         10  Accounting
  Smith       Clerk         20  Research
  Adams       Clerk         20  Research
  James       Clerk         30  Sales
4 rows selected

```

This example shows four outer join queries on the CUSTOMERS, ORDERS, LINEITEMS, and PARTS tables. These tables are shown here:

```

SQL> SELECT custno, custname
cont> FROM customers
cont> ORDER BY custno;
  CUSTNO  CUSTNAME
    1    Angelic Co
    2    Believable Co
    3    Cables R Us
3 rows selected

SQL>
SQL> SELECT orderno, custno, orderdate
cont> FROM orders
cont> ORDER BY orderno;
  ORDerno  CUSTNO  ORDERDATE
    9001         1    1999-10-13
    9002         2    1999-10-13
    9003         1    1999-10-20
    9004         1    1999-10-27
    9005         2    1999-10-31
5 rows selected

SQL>
SQL> SELECT orderno, lineno, partno, quantity
cont> FROM lineitems
cont> ORDER BY orderno, lineno;
  ORDerno  LINENO  PARTNO  QUANTITY
    9001         1     101         15
    9001         2     102         10
    9002         1     101         25
    9002         2     103         50
    9003         1     101         15
    9004         1     102         10
    9004         2     103         20
7 rows selected

SQL>
SQL> SELECT partno, partname
cont> FROM parts
cont> ORDER BY partno;
  PARTNO  PARTNAME
    101    X-Ray Screen

```

```

102 Yellow Bag
103 Zoot Suit
3 rows selected

```

The customer Cables R Us has placed no orders, and order number 9005 has no line items.

The following outer join returns all customers and the dates they placed orders. The (+) operator ensures that customers who placed no orders are also returned:

```

SQL> SELECT custname, orderdate
cont> FROM customers, orders
cont> WHERE customers.custno = orders.custno (+)
cont> ORDER BY customers.custno, orders.orderdate;
CUSTOMERS.CUSTNAME  ORDERS.ORDERDATE
Angelic Co          1999-10-13
Angelic Co          1999-10-20
Angelic Co          1999-10-27
Believable Co       1999-10-13
Believable Co       1999-10-31
Cables R Us         NULL
6 rows selected

```

The following outer join builds on the result of the previous one by adding the LINEITEMS table to the FROM clause, columns from this table to the select list, and a join condition joining this table to the ORDERS table to the where_ clause. This query joins the results of the previous query to the LINEITEMS table and returns all customers, the dates they placed orders, and the part number and quantity of each part they ordered. The first (+) operator serves the same purpose as in the previous query. The second (+) operator ensures that orders with no line items are also returned:

```

SQL> SELECT custname, orderdate, partno, quantity
cont> FROM customers, orders, lineitems
cont> WHERE customers.custno = orders.custno (+)
cont> AND orders.orderno = lineitems.orderno (+)
cont> ORDER BY customers.custno, orders.orderdate, lineitems.partno;
CUSTOMERS.CUSTNAME  ORDERS.ORDERDATE  LINEITEMS.PARTNO  LINEITEMS.QUANTITY
Angelic Co          1999-10-13        101               15
Angelic Co          1999-10-13        102               10
Angelic Co          1999-10-20        101               15
Angelic Co          1999-10-27        102               10
Angelic Co          1999-10-27        103               20
Believable Co       1999-10-13        101               25
Believable Co       1999-10-13        103               50
Believable Co       1999-10-31        NULL              NULL
Cables R Us         NULL              NULL              NULL
9 rows selected

```

The following outer join builds on the result of the previous one by adding the PARTS table to the FROM clause, the PARTNAME column from this table to the select list, and a join condition joining this table to the LINEITEMS table to the where_clause. This query joins the results of the previous query to the PARTS table to return all customers, the dates they placed orders, and the quantity and name of each part they ordered. The first two (+) operators serve the same purposes as in the previous query. The third (+) operator ensures that rows with NULL part numbers are also returned:

```
SQL> SELECT custname, orderdate, quantity, partname
cont> FROM customers, orders, lineitems, parts
cont> WHERE customers.custno = orders.custno (+)
cont>       AND orders.orderno = lineitems.orderno (+)
cont>       AND lineitems.partno = parts.partno (+)
cont> ORDER BY customers.custno, orders.orderdate, parts.partno;
```

CUSTOMERS.CUSTNAME	ORDERS.ORDERDATE	LINEITEMS.QUANTITY	PARTS.PARTNAME
Angelic Co	1999-10-13	15	X-Ray Screen
Angelic Co	1999-10-13	10	Yellow Bag
Angelic Co	1999-10-20	15	X-Ray Screen
Angelic Co	1999-10-27	10	Yellow Bag
Angelic Co	1999-10-27	20	Zoot Suit
Believable Co	1999-10-13	25	X-Ray Screen
Believable Co	1999-10-13	50	Zoot Suit
Believable Co	1999-10-31	NULL	NULL
Cables R Us	NULL	NULL	NULL

9 rows selected

System Tables

This appendix describes the Oracle Rdb system tables.

Oracle Rdb stores information about the database as a set of special system tables. The system tables are the definitive source of Oracle Rdb metadata. **Metadata** defines the structure of the database; for example, metadata defines the fields that comprise a particular table and the fields that can index that table.

The definitions of most system tables are standard and are likely to remain constant in future versions of Oracle Rdb.

In each description for a particular system table, BLR refers to binary language representation. This is low-level syntax used internally to represent Oracle Rdb data manipulation operations.

The following sections describe the usage of system tables with respect to particular versions of Oracle Rdb or in relation to other database constructs, operations, or products.

H.1 Using Data Dictionary

Although you can store your data definitions in the data dictionary, the database system refers only to the system tables in the database file itself for these definitions. In a sense, the system tables are an internal data dictionary for the database. This method improves performance as Oracle Rdb does not have to access the data dictionary at run time.

H.2 Modifying System Tables

When you create a database, Oracle Rdb defines, populates, and manipulates the system tables. As the user performs data definition operations on the database, Oracle Rdb reads and modifies the system tables to reflect those operations. You should not modify any of the Oracle Rdb system tables using data manipulation language, nor should you define any domains based on system table fields. However, you can use regular Oracle Rdb data manipulation statements to retrieve the contents of the system tables. This

means that your program can determine the structure and characteristics of the database by retrieving the fields of the system tables.

H.3 Updating Metadata

When you use the SQL SET TRANSACTION . . . RESERVING statement to lock a set of tables for an Oracle Rdb operation, you normally exclude from the transaction all the tables not listed in the RESERVING clause. However, Oracle Rdb accesses and updates system tables as necessary, no matter which tables you have locked with the SQL SET TRANSACTION statement.

When your transaction updates database metadata, Oracle Rdb reserves the system tables involved in the update in the EXCLUSIVE share mode. Other users are unable to perform data definition operations on these tables until you complete your transaction. For example:

- When you refer to a domain (global field) in an update transaction that changes data definitions, Oracle Rdb locks an index for the system table, RDB\$RELATION_FIELDS. No other users can refer to the same domain until you commit your transaction.
- When you change a relation (table) or domain definition, Oracle Rdb locks an index in the system table, RDB\$FIELD_VERSIONS. No other users can change table or global field definitions until you commit your transaction.
- When you change a table definition, Oracle Rdb locks an index in the system table, RDB\$RELATION_FIELDS. No other users can change tables in the same index node until you commit your transaction.

H.4 LIST OF BYTE VARYING Metadata

Oracle Rdb has supported multiple segment LIST OF BYTE VARYING data types for user-defined data. However in previous versions, Oracle Rdb maintained its own LIST OF BYTE VARYING metadata columns as single segments. This restricted the length to approximately 65530 bytes. An SQL CREATE TRIGGER or CREATE MODULE statement could fail due to this restriction.

This limit was lifted by changing the way Oracle Rdb stores its own metadata.

- For columns containing binary data, such as the binary representation of query, routine, constraint, trigger action, computed by column, or query outline, Oracle Rdb breaks the data into pieces that best fit the system storage area page size. Thus, the segments are all the same size with a possible small trailing segment.

The LIST OF BYTE VARYING column value is no longer fragmented, improving performance when reading system metadata.

- For columns containing text data such as the SQL source (for elements such as triggers and views) and user-supplied comment strings, Oracle Rdb breaks the text at line boundaries (indicated by ASCII carriage returns and line feeds) and stores the text without the line separator. Thus, the segments are of varying size with a possible zero length for blank lines.

An application can now easily display the LIST OF BYTE VARYING column value and the application no longer needs to break up the single text segment for printing.

No change is made to the LIST OF BYTE VARYING column values when a database is converted (using the RMU Convert command, RMU Restore command, or SQL EXPORT/IMPORT statements) from a previous version.

Applications that read the Oracle Rdb system LIST OF BYTE VARYING column values must be changed to understand multiple segments. Applications that do not read these system column values should see no change to previous behavior. Tools such as the RMU Extract command and the SQL SHOW and EXPORT statements handle both the old and new formats of the system metadata.

H.5 Read Only Access

The following is a list of fields of various system tables that are set to read-only access.

- RDB\$ACCESS_CONTROL
- RDB\$COLLATION_CREATOR
- RDB\$CONSTRAINT_CREATOR
- RDB\$DATABASE_CREATOR
- RDB\$FIELD_CREATOR
- RDB\$FLAGS
- RDB\$INDEX_CREATOR
- RDB\$MODULE_CREATOR
- RDB\$MODULE_OWNER
- RDB\$OUTLINE_CREATOR
- RDB\$PROFILE_CREATOR

- RDB\$RELATION_CREATOR
- RDB\$ROUTINE_CREATOR
- RDB\$ROUTINE_OWNER
- RDB\$SEQUENCE_CREATOR
- RDB\$SYNONYM_CREATOR
- RDB\$TRIGGER_CREATOR

The following BASIC program uses an SQL Module to query system tables

```

PROGRAM SYSTEM_RELATION
! This BASIC program interactively prompts a user to enter a name
! of a system table (table). Next, the program calls an SQL
! Module which uses a cursor to read the system table that the
! user entered. Upon reading the fields (domains) of the system
! table, the program displays a message to the user as to whether
! the fields in a system table can be updated.
OPTION TYPE = EXPLICIT, SIZE = INTEGER LONG
ON ERROR GOTO ERR_ROUTINE
!
! Declare variables and constants
!
DECLARE STRING Column_name, Table_name
DECLARE INTEGER Update_yes, sqlcode
DECLARE INTEGER CONSTANT TRIM_BLANKS = 128, UPPER_CASE = 32
EXTERNAL SUB SET_TRANSACTION (LONG)
EXTERNAL SUB OPEN_CURSOR (LONG, STRING)
EXTERNAL SUB FETCH_COLUMN (LONG, STRING, INTEGER)
EXTERNAL SUB CLOSE_CURSOR (LONG)
EXTERNAL SUB COMMIT_TRANS (LONG)
!
! Prompt for table name
!
INPUT 'Name of Table'; Table_name
Table_name = EDIT$(Table_name, UPPER_CASE)
PRINT 'Starting query'
PRINT 'In '; Table_name; ' Table, columns:'
!
! Call the SQL module to start the transaction.
!
CALL SET_TRANSACTION(Sqlcode)
!
! Open the cursor.
!
CALL OPEN_CURSOR(Sqlcode, Table_name)
GET_LOOP:
WHILE (Sqlcode = 0)

```

```

!
! Fetch each column
!
CALL FETCH_COLUMN(Sqlcode, Column_name, Update_yes)
  IF (Sqlcode = 0)
    THEN
      !
      ! Display returned column
      !
      PRINT '  '; EDIT$(Column_name, TRIM_BLANKS);
      IF (update_yes = 1)
        THEN
          PRINT ' can be updated'
        ELSE
          PRINT ' cannot be updated'
        END IF
      END IF
    END IF
  NEXT
ERR_ROUTINE:
  IF Sqlcode = 100
    THEN
      PRINT "No more rows."
      RESUME PROG_END
    ELSE
      PRINT "Unexpected error: ", Sqlcode, Err
      RESUME PROG_END
    END IF
PROG_END:
!
! Close the cursor, commit work and exit
!
CALL CLOSE_CURSOR(Sqlcode)
CALL COMMIT_TRANS(Sqlcode)
END PROGRAM

```

The following module provides the SQL procedures that are called by the preceding BASIC program.

```

-- This SQL module provides the SQL procedures that are called by the
-- preceding BASIC program, system table

-----
-- Header Information Section
-----
MODULE          SQL_SYSTEM_REL_BAS  -- Module name
LANGUAGE        BASIC               -- Language of calling program
AUTHORIZATION    SQL_SAMPLE         -- Authorization ID

```

```

-----
-- DECLARE Statements Section
-----
DECLARE ALIAS FILENAME 'MF_PERSONNEL' -- Declaration of the database.

DECLARE SELECT_UPDATE CURSOR FOR
    SELECT RDB$FIELD_NAME, RDB$UPDATE_FLAG
    FROM RDB$RELATION_FIELDS
    WHERE RDB$RELATION_NAME = table_name
    ORDER BY RDB$FIELD_POSITION
-----

-- Procedure Section
-----
-- Start a transaction.
PROCEDURE SET_TRANSACTION
    SQLCODE;

    SET TRANSACTION READ WRITE;

-- Open the cursor.
PROCEDURE OPEN_CURSOR
    SQLCODE
    table_name          RDB$RELATION_NAME;

    OPEN SELECT_UPDATE;

-- Fetch a row.
PROCEDURE FETCH_COLUMN
    SQLCODE
    field_name          RDB$FIELD_NAME
    update_flag         RDB$UPDATE_FLAG;

    FETCH SELECT_UPDATE INTO :field_name, :update_flag;

-- Close the cursor.
PROCEDURE CLOSE_CURSOR
    SQLCODE;

    CLOSE SELECT_UPDATE;

-- Commit the transaction.
PROCEDURE COMMIT_TRANS
    SQLCODE;

    COMMIT;

```

H.6 All System Tables

The Oracle Rdb system tables are as follows:

RDB\$CATALOG_SCHEMA	Contains the name and definition of each SQL catalog and schema. This table is present only in databases with the SQL multischema feature enabled.
RDB\$COLLATIONS	The collating sequences used by this database.
RDB\$CONSTRAINTS	Name and definition of each constraint.
RDB\$CONSTRAINT_RELATIONS	Name of each table that participates in a given constraint.
RDB\$DATABASE	Database-specific information.
RDB\$FIELD_VERSIONS	One row for each version of each column definition in the database.
RDB\$FIELDS	Characteristics of each domain in the database.
RDB\$GRANTED_PROFILES	Description of roles and profiles granted to users and other roles.
RDB\$INDEX_SEGMENTS	Columns that make up an index.
RDB\$INDICES	Characteristics of the indexes for each table.
RDB\$INTERRELATIONS	Interdependencies of entities used in the database.
RDB\$MODULES	Module definition as defined by a user, including the header and declaration section.
RDB\$OBJECT_SYNONYMS	When synonyms are enabled, this system table is created to describe the synonym name, type, and target.
RDB\$PARAMETERS	Interface definition for each routine stored in RDB\$ROUTINES. Each parameter to a routine (procedure or function) is described by a row in RDB\$PARAMETERS.
RDB\$PRIVILEGES	Protection for the database objects.
RDB\$PROFILES	Description of any profiles, roles or users in the database.

RDB\$QUERY_OUTLINES	Query outline definitions used by the optimizer to retrieve known query outlines prior to optimization.
RDB\$RELATION_CONSTRAINTS	Lists all table-specific constraints.
RDB\$RELATION_CONSTRAINT_FLDS	Lists the columns that participate in unique, primary, or foreign key declarations for table-specific constraints.
RDB\$RELATION_FIELDS	Columns defined for each table.
RDB\$RELATIONS	Tables and views in the database.
RDB\$ROUTINES	Description of each function and procedure in the database. The routine may be standalone or part of a module.
RDB\$SEQUENCES	Characteristics of any sequences defined for the database.
RDB\$STORAGE_MAPS	Characteristics of each storage map.
RDB\$STORAGE_MAP_AREAS	Characteristics of each partition of a storage map.
RDB\$SYNONYMS	Connects an object's user-specified name to its internal database name. This table is only present in databases with the SQL multischema feature enabled.
RDB\$TRIGGERS	Definition of a trigger.
RDB\$VIEW_RELATIONS	Interdependencies of tables used in views.
RDB\$WORKLOAD	Collects workload information.
RDB\$TRIGGER_ACTIONS	Collects workload information.

H.6.1 RDB\$CATALOG_SCHEMA

The RDB\$CATALOG_SCHEMA system table contains the name and definition of each SQL catalog and schema. This table is present only in databases that have the SQL multischema feature enabled. The following table provides information on the columns of the RDB\$CATALOG_SCHEMA system table.

Column Name	Data Type	Summary Description
RDB\$PARENT_ID	integer	For a schema, this is the RDB\$CATALOG_SCHEMA_ID of the catalog to which this schema belongs. For a catalog, this column is always 0.
RDB\$CATALOG_SCHEMA_NAME	char(31)	The name of the catalog or schema.
RDB\$CATALOG_SCHEMA_ID	integer	A unique identifier indicating whether this is a catalog or a schema. Schema objects have positive identifiers starting at 1 and increasing. Catalog objects have negative identifiers starting at -1 and decreasing. 0 is reserved.
RDB\$DESCRIPTION	list of byte varying	A user-supplied description of the catalog or schema.
RDB\$SCHEMA_AUTH_ID	char(31)	The authorization identifier of the creator of the schema.
RDB\$SECURITY_CLASS	char(20)	Reserved for future use.
RDB\$CREATED	date vms	Set when the schema or catalog is created.
RDB\$LAST_ALTERED	date vms	Set when SQL ALTER CATALOG or ALTER SCHEMA statement is used (future).
RDB\$CATALOG_SCHEMA_CREATOR	char(31)	Creator of this schema or catalog.

H.6.2 RDB\$COLLATIONS

The RDB\$COLLATIONS system table describes the collating sequence to be used in the database. The following table provides information on the columns of the RDB\$COLLATIONS system table.

Column Name	Data Type	Summary Description
RDB\$COLLATION_NAME	char(31)	Supplies the name by which the database's collating sequences are known within the database.
RDB\$COLLATION_SEQUENCE	list of byte varying	Internal representation of the collating sequence.

Column Name	Data Type	Summary Description
RDB\$DESCRIPTION	list of byte varying	A user-supplied description of the collating sequence.
RDB\$FLAGS	integer	A bit mask where the following bits are set: <ul style="list-style-type: none"> • Bit 0 If an ASCII collating sequence. • Bit 1 If a DEC_MCS collating sequence.
RDB\$SECURITY_CLASS	char(20)	Reserved for future use.
RDB\$CREATED	date vms	Set when the collating sequence is created.
RDB\$LAST_ALTERED	date vms	Reserved for future use.
RDB\$COLLATION_CREATOR	char(31)	Creator of this collating sequence.

H.6.3 RDB\$CONSTRAINTS

The RDB\$CONSTRAINTS system table contains the name and definition of each constraint. The following table provides information on the columns of the RDB\$CONSTRAINTS system table.

Column Name	Data Type	Summary Description
RDB\$CONSTRAINT_NAME	char(31)	The system-wide unique name of the constraint.
RDB\$CONSTRAINT_BLR	list of byte varying	The BLR that defines the constraint.
RDB\$CONSTRAINT_SOURCE	list of byte varying	The user's source for the constraint.
RDB\$DESCRIPTION	list of byte varying	A user-supplied description of this constraint.

Column Name	Data Type	Summary Description
RDB\$EVALUATION_TIME	integer	<p>A value that represents when a constraint is evaluated, as follows:</p> <ul style="list-style-type: none"> 0 At commit time (deferred initially deferred). 1 At verb time (deferrable initially immediate). 2 At verb time (not deferrable).
RDB\$EXTENSION_PARAMETERS	list of byte varying	Reserved for future use.
RDB\$SECURITY_CLASS	char(20)	Reserved for future use.
RDB\$CREATED	date vms	Set when the constraint is created.
RDB\$LAST_ALTERED	date vms	Reserved for future use.
RDB\$CONSTRAINT_CREATOR	char(31)	Creator of this constraint.
RDB\$FLAGS	integer	Flags.

H.6.3.1 RDB\$FLAGS

Represents flags for RDB\$CONSTRAINTS system table.

Bit Position	Description
0	Currently disabled.
1	Currently enabled without validation.
2	Tracking if SQL generated the name.
3	A not null constraint.
4	A primary key constraint.
5	A foreign key constraint.
6	A check constraint.
7	A unique constraint.

H.6.4 RDB\$CONSTRAINT_RELATIONS

The RDB\$CONSTRAINT_RELATIONS system table lists all tables that participate in a given constraint. The following table provides information on the columns of the RDB\$CONSTRAINT_RELATIONS system table.

Column Name	Data Type	Summary Description
RDB\$CONSTRAINT_NAME	char(31)	The system-wide unique name of the constraint.
RDB\$RELATION_NAME	char(31)	The name of a table involved in the constraint.
RDB\$FLAGS	integer	Flags.
RDB\$CONSTRAINT_CONTEXT	integer	The context variable of the table involved in the constraint.
RDB\$SECURITY_CLASS	char(20)	Reserved for future use.

H.6.4.1 RDB\$FLAGS

Represents flags for RDB\$CONSTRAINT_RELATIONS system table.

Bit Position	Description
0	Reserved for future use.
1	Reserved for future use.
2	If the constraint is on the specified table.
3	If the constraint evaluates with optimization by dbkey lookup.
4	If the constraint checks for existence.
5	If the constraint checks for uniqueness.
6	If the constraint needs to evaluate on store of specified table row.
7	If the constraint need not evaluate on store of specified table row.
8	If the constraint needs to evaluate on erase of specified table row.
9	If the constraint need not evaluate on erase of specified table row.

H.6.5 RDB\$DATABASE

The RDB\$DATABASE system table contains information that pertains to the overall database. This table can contain only one row. The following table provides information on the columns of the RDB\$DATABASE system table.

Column Name	Data Type	Summary Description
RDB\$CDD_PATH	char(256)	The dictionary path name for the database.
RDB\$FILE_NAME	char(255)	Oracle Rdb returns the file specification of the database root file. [1]
RDB\$MAJ_VER	integer	Derived from the database major version.
RDB\$MIN_VER	integer	Derived from the database minor version.
RDB\$MAX_RELATION_ID	integer	The largest table identifier assigned. Oracle Rdb assigns the next table an ID of MAX_RELATION_ID + 1.
RDB\$RELATION_ID	integer	The unique identifier of the RDB\$RELATIONS table. If you drop a table, that identifier is not assigned to any other table.
RDB\$RELATION_ID_ROOT_DBK	char(8)	A pointer (database key or dbkey) to the base of the RDB\$REL_REL_ID_NDX index on column RDB\$RELATION_ID.
RDB\$RELATION_NAME_ROOT_DBK	char(8)	A pointer (dbkey) to the base of the RDB\$REL_REL_NAME_NDX index on column RDB\$RELATION_NAME.
RDB\$FIELD_ID	integer	The identifier of the RDB\$FIELD_VERSIONS table.
RDB\$FIELD_REL_FLD_ROOT_DBK	char(8)	A pointer (dbkey) to the base of the RDB\$VER_REL_ID_VER_NDX index on columns RDB\$RELATION_ID and RDB\$VERSION.
RDB\$INDEX_ID	integer	The identifier of the RDB\$INDICES table.
RDB\$INDEX_NDX_ROOT_DBK	char(8)	A pointer (dbkey) to the base of the RDB\$NDX_NDX_NAME_NDX index on column RDB\$INDEX_NAME.
RDB\$INDEX_REL_ROOT_DBK	char(8)	A pointer (dbkey) to the base of the RDB\$NDX_REL_NAM_NDX index on column RDB\$RELATION_ID.

Column Name	Data Type	Summary Description
RDB\$INDEX_SEG_ID	integer	The identifier of the RDB\$INDEX_SEGMENTS table.
RDB\$INDEX_SEG_FLD_ROOT_DBK	char(8)	A pointer (dbkey) to the base of the RDB\$NDX_SEG_NAM_FLD_POS_NDX index on columns RDB\$INDEX_NAME and RDB\$FIELD_POSITION.
RDB\$SEGMENTED_STRING_ID	integer	The logical area ID that contains the segmented strings.
RDB\$ACCESS_CONTROL	list of byte varying	The access control policy for the database.
RDB\$DESCRIPTION	list of byte varying	A user-supplied description of the database.
RDB\$DATABASE_PARAMETERS	list of byte varying	Reserved for future use.
RDB\$EXTENSION_PARAMETERS	list of byte varying	Reserved for future use.
RDB\$FLAGS	integer	Flags.
RDBVMS\$MAX_VIEW_ID	integer	The largest view identifier assigned to the RDB\$RELATION_ID column in the RDB\$RELATIONS system table. Oracle Rdb assigns the next view an ID of MAX_VIEW_ID + 1.
RDBVMS\$SECURITY_AUDIT	integer	A bit mask that indicates the privileges that will be audited for the database, as specified in the RMU Set Audit command.
RDBVMS\$SECURITY_ALARM	integer	A bit mask that indicates the privileges that will produce alarms for the database, as specified in the RMU Set Audit command.
RDBVMS\$SECURITY_USERS	list of byte varying	An access control list that identifies users who will be audited or who will produce alarms for DAC (discretionary access control) events when DACCESS (discretionary access) auditing is enabled for specific database objects.
RDB\$SECURITY_CLASS	char(20)	Reserved for future use.
RDBVMS\$SECURITY_AUDIT2	integer	Reserved for future use.

Column Name	Data Type	Summary Description
RDBVMS\$SECURITY_ALARM2	integer	Reserved for future use.
RDBVMS\$CHARACTER_SET_ID	integer	Value is the character set ID used for the identifier character set.
RDBVMS\$CHARACTER_SET_NATIONAL	integer	Value is the character set ID used for all NCHAR (also called NATIONAL CHAR or NATIONAL CHARACTER) data types and literals.
RDBVMS\$CHARACTER_SET_DEFAULT	integer	Value is the character set ID used for the default character set.
RDB\$MAX_ROUTINE_ID	integer	Maintains a count of the modules and routines added to the database. Value is 0 if no routines or modules have been added to the database.
RDB\$CREATED	date vms	Set when the database is created.
RDB\$LAST_ALTERED	date vms	Set when SQL ALTER DATABASE statement is used.
RDB\$DATABASE_CREATOR	char(31)	Creator of this database.
RDB\$DEFAULT_STORAGE_AREA_ID	integer	Default storage area used for unmapped, persistent tables and indices.
RDB\$DEFAULT_TEMPLATE_AREA_ID	integer	Reserved for future use.

Footnote:

- [1] The root file specification is not stored on disk (an RMU Dump command with the Areas qualifier shows that this field is blank) and is only returned to queries at runtime. Therefore, the root file specification remains correct after you use the RMU Move_Area, RMU Copy_Database, and RMU Backup commands, and the SQL EXPORT and IMPORT statements.

The following ALTER DATABASE clauses modify the RDB\$LAST_ALTERED column in the RDB\$DATABASE system table:

- CARDINALITY COLLECTION IS {ENABLED | DISABLED}
- DICTIONARY IS [NOT] REQUIRED
- DICTIONARY IS NOT USED
- METADATA CHANGES ARE {ENABLED | DISABLED}
- MULTISCHEMA IS {ON | OFF}

- SECURITY CHECKING IS EXTERNAL (PERSONA SUPPORT IS {ENABLED | DISABLED})
- SECURITY CHECKING IS INTERNAL (ACCOUNT CHECK IS {ENABLED | DISABLED})
- SYNONYMS ARE ENABLED
- WORKLOAD COLLECTION IS {ENABLED | DISABLED}

The following SQL statements modify the RDB\$LAST_ALTERED column in the RDB\$DATABASE system table:

- GRANT statement
- REVOKE statement
- COMMENT ON DATABASE statement

H.6.5.1 RDB\$FLAGS

Represents flags for RDB\$DATABASE system table.

Bit Position	Description
0	If dictionary required.
1	If ANSI protection used.
2	If database file is a CDD\$DATABASE database.
3	Reserved for future use.
4	Reserved for future use.
5	Reserved for future use.
6	Multischema is enabled.
7	Reserved for future use.
8	System indexes use run length compression.
9	The optimizer saves workload in RDB\$WORKLOAD system table.
10	The optimizer is not updating table and index cardinalities.
11	All metadata changes are disabled.
12	Oracle Rdb uses database for user and role names.
13	When true use PERSONA services.
14	Synonyms are supported.
15	Prefix cardinalities are not collected for system indexes.
16	If collecting, use full algorithm for system indexes.

Bit Position	Description
17	Use sorted ranked index for system indexes.

H.6.6 RDB\$FIELD_VERSIONS

The RDB\$FIELD_VERSIONS system table is an Oracle Rdb extension. This table contains one row for each version of each column definition in the database. The following table provides information on the columns of the RDB\$FIELD_VERSIONS system table.

Column Name	Data Type	Summary Description
RDB\$RELATION_ID	integer	The identifier for a table within the database.
RDB\$FIELD_ID	integer	An identifier used internally to name the column represented by this row.
RDB\$FIELD_NAME	char(31)	The name of the column.
RDB\$VERSION	integer	The version number for the table definition to which this column belongs.
RDB\$FIELD_TYPE	integer	The data type of the column represented by this row. This data type must be interpreted according to the rules for interpreting the DSC\$B_DTYPE field of class S descriptors (as defined in the OpenVMS Calling Standard). Segmented strings require a unique field type identifier. This identifier is currently 261.
RDB\$FIELD_LENGTH	integer	The length of the column represented by this row. This length must be interpreted according to the rules for interpreting the DSC\$W_LENGTH field within class S and SD descriptors (as defined in the OpenVMS Calling Standard).
RDB\$OFFSET	integer	The byte offset of the column from the beginning of the row.

Column Name	Data Type	Summary Description
RDB\$FIELD_SCALE	integer	<p>For numeric data types, the scale factor to be applied when interpreting the contents of the column represented by this row.</p> <p>This scale factor must be interpreted according to the rules for interpreting the DSC\$B_SCALE field of class SD descriptors (as defined in the OpenVMS Calling Standard).</p> <p>For date-time data types, RDB\$FIELD_SCALE is fractional seconds precision. For other non-numeric data types, RDB\$FIELD_SCALE is 0.</p>
RDB\$FLAGS	integer	Flags.
RDB\$VALIDATION_BLR	list of byte varying	The BLR that represents the SQL check constraint clause defined in this version of the column.
RDB\$COMPUTED_BLR	list of byte varying	The BLR that represents the SQL clause, COMPUTED BY, defined in this version of the column.
RDB\$MISSING_VALUE	list of byte varying	The BLR that represents the SQL clause, MISSING_VALUE, defined in this version of the column.
RDB\$SEGMENT_LENGTH	integer	The length of a segmented string segment. For date-time interval fields, the interval leading field precision.
RDBVMS\$COLLATION_NAME	char(31)	The name of the collating sequence for the column.
RDB\$ACCESS_CONTROL	list of byte varying	The access control list for the column.
RDB\$DEFAULT_VALUE2	list of byte varying	The SQL default value.
RDBVMS\$SECURITY_AUDIT	integer	A bit mask that indicates the privileges that will be audited for the database, as specified in the RMU Set Audit command.

Column Name	Data Type	Summary Description
RDBVMS\$SECURITY_ALARM	integer	A bit mask that indicates the privileges that will produce alarms for the database, as specified in the RMU Set Audit command.
RDB\$FIELD_SUB_TYPE	integer	A value that describes the data subtype of RDB\$FIELD_TYPE as shown in Help topic RDB\$FIELD_SUB_TYPE.
RDB\$SECURITY_CLASS	char(20)	Reserved for future use.

H.6.6.1 RDB\$FLAGS

Represents flags for RDB\$FIELD_VERSIONS system table.

Bit Position	Description
0	Not used.
1	Not used.
2	Not used.
3	Used by Oracle Rdb internally.
4	Set if column references a local temporary table (usually a COMPUTED BY column).
5	Use SQL semantics for check constraint processing.
6	AUTOMATIC set on insert.
7	AUTOMATIC set on update.
8	If check constraint fails, use name in message.
9	Column is computed by an IDENTITY sequence.
10	View column is based on a read-only, or dbkey column.

H.6.7 RDB\$PARAMETER_SUB_TYPE

For details, see the Help topic RDB\$FIELD_SUB_TYPE.

H.6.8 RDB\$FIELD_SUB_TYPE

The following table lists the values for the RDB\$FIELD_SUB_TYPE and the RDB\$PARAMETER_SUB_TYPE columns.

RDB\$FIELD_TYPE = DSC\$K_DTYPE_ADT	
RDB\$FIELD_SUB_TYPE [1]	Summary Description
Less than 0 Equal to 0	Reserved for future use. Traditional OpenVMS timestamp, which includes year, month, day, hour, minute, second.
7	DATE ANSI, which includes year, month, day.
56	TIME, which includes hour, minute, second.
63	TIMESTAMP, which includes year, month, day, hour, minute, second.
513	INTERVAL YEAR.
514	INTERVAL MONTH.
515	INTERVAL YEAR TO MONTH.
516	INTERVAL DAY.
520	INTERVAL HOUR.
524	INTERVAL DAY TO HOUR.
528	INTERVAL MINUTE.
536	INTERVAL HOUR TO MINUTE.
540	INTERVAL DAY TO MINUTE.
544	INTERVAL SECOND.
560	INTERVAL MINUTE TO SECOND.
568	INTERVAL HOUR TO SECOND.
572	INTERVAL DAY TO SECOND.

RDB\$FIELD_TYPE = DSC\$K_DTYPE_T or DSC\$K_DTYPE_VT	
RDB\$FIELD_SUB_TYPE	Summary Description
Equal to 0	ASCII or DEC_MCS character set. Character set other than ASCII or DEC_MCS.
Greater than 0	
Less than 0	Special use of character data.
RDB\$FIELD_TYPE = DSC\$K_DTYPE_BLOB [2]	
RDB\$FIELD_SUB_TYPE	Summary Description
Less than 0	User-specified.
Equal to 0	Default.
Equal to 1	BLR (query) type.
Equal to 2	Character type.
Equal to 3	MBLR (definition) type.
Equal to 4	Binary type.
Equal to 5	OBLR (outline) type.
Greater than 5	Reserved for future use.

Footnotes:

- [1] When RDB\$FIELD_SUB_TYPE is not equal to 0, then RDB\$SEGMENT_LENGTH can hold the interval leading field precision for intervals, and RDB\$FIELD_SCALE can hold the fractional seconds precision for interval, time, or timestamp.
- [2] RDB\$SEGMENT_LENGTH is the suggested size for a single binary large object (BLOB) segment.

H.6.9 RDB\$FIELDS

The RDB\$FIELDS system table describes the global (generic) characteristics of each domain in the database. There is one row for each domain in the database. The following table provides information on the columns of the RDB\$FIELDS system table.

Column Name	Data Type	Summary Description
RDB\$FIELD_NAME	char(31)	The name of the domain represented by this row. Each row within RDB\$FIELDS must have a unique RDB\$FIELD_NAME value.
RDB\$FIELD_TYPE	integer	The data type of the domain represented by this row. This data type must be interpreted according to the rules for interpreting the DSC\$B_DTYPE field of class S descriptors (as defined in the OpenVMS Calling Standard). Segmented strings require a unique field type identifier. This identifier is 261.
RDB\$FIELD_LENGTH	integer	The length of the field represented by this row. This length must be interpreted according to the rules for interpreting the DSC\$W_LENGTH field within class S and SD descriptors (as defined in OpenVMS Calling Standard). For strings, this field contains the length in octets (8-bit bytes), not in characters.
RDB\$FIELD_SCALE	integer	For numeric data types, the scale factor to be applied when interpreting the contents of the field represented by this row. This scale factor must be interpreted according to the rules for interpreting the DSC\$B_SCALE field of class SD descriptors (as defined in the OpenVMS Calling Standard). For date-time data types, RDB\$FIELD_SCALE is fractional seconds precision. For other non-numeric data types, RDB\$FIELD_SCALE is 0.

Column Name	Data Type	Summary Description
RDB\$SYSTEM_FLAG	integer	<p>A bit mask where the following bits are set:</p> <ul style="list-style-type: none"> • If Bit 0 is clear, this is a user-defined domain. • If Bit 0 is set, this is a system domain.
RDB\$VALIDATION_BLR	list of byte varying	The BLR that represents the validation expression to be checked each time a column based on this domain is updated.
RDB\$COMPUTED_BLR	list of byte varying	The BLR that represents the expression used to calculate a value for the column based on this domain.
RDB\$EDIT_STRING	varchar(255)	The edit string used by interactive SQL when printing the column based on this domain. RDB\$EDIT_STRING can be null.
RDB\$MISSING_VALUE	list of byte varying	The value used when the missing value of the column based on this domain is retrieved or displayed. RDB\$MISSING_VALUE does not store any value in a column; instead, it flags the column value as missing.
RDB\$FIELD_SUB_TYPE	integer	A value that describes the data subtype of RDB\$FIELD_TYPE as shown in the RDB\$FIELD_SUB_TYPE Help topic.
RDB\$DESCRIPTION	list of byte varying	A user-supplied description of this domain.
RDB\$VALIDATION_SOURCE	list of byte varying	The user's source text for the validation criteria.
RDB\$COMPUTED_SOURCE	list of byte varying	The user's source used to calculate a value at execution time.

Column Name	Data Type	Summary Description
RDB\$QUERY_NAME	char(31)	<p>The query name of this domain. Column attributes in RDB\$RELATION_FIELDS take precedence over attributes in RDB\$FIELDS.</p> <p>If the attribute value is missing in RDB\$RELATION_FIELDS, the value from RDB\$FIELDS is used. RDB\$QUERY_NAME can be null.</p>
RDB\$QUERY_HEADER	list of byte varying	<p>The query header of the domain is used by interactive SQL. Column attributes in RDB\$RELATION_FIELDS take precedence over attributes in RDB\$FIELDS.</p> <p>If the attribute value is missing in RDB\$RELATION_FIELDS, the value from RDB\$FIELDS is used.</p>
RDB\$DEFAULT_VALUE	list of byte varying	<p>The default value used by non-SQL interfaces when no value is specified for a column during a STORE clause. It differs from RDB\$MISSING_VALUE in that it holds an actual column value.</p> <p>Column attributes in RDB\$RELATION_FIELDS take precedence over attributes in RDB\$FIELDS.</p> <p>If the attribute value is missing in RDB\$RELATION_FIELDS, the value from RDB\$FIELDS is used.</p>
RDB\$SEGMENT_LENGTH	integer	<p>The length of a segmented string segment. For date-time interval fields, the interval leading field precision.</p>
RDB\$EXTENSION_PARAMETERS	list of byte varying	Reserved for future use.
RDB\$CDD_NAME	list of byte varying	<p>The fully qualified name of the dictionary entity upon which the domain definition is based, as specified in the SQL clause, FROM PATHNAME.</p>

Column Name	Data Type	Summary Description
RDBVMS\$COLLATION_NAME	char(31)	The name of the collating sequence for the domain.
RDB\$DEFAULT_VALUE2	list of byte varying	The BLR for the SQL default value. This value is used when no value is provided in an SQL INSERT statement.
RDB\$SECURITY_CLASS	char(20)	Reserved for future use.
RDB\$FLAGS	integer	Flags.
RDB\$CREATED	date vms	Set when the domain is created.
RDB\$LAST_ALTERED	date vms	Set when SQL ALTER DOMAIN statement used.
RDB\$FIELD_CREATOR	char(31)	Creator of this domain.

H.6.9.1 RDB\$FLAGS

Represents flags for RDB\$FIELDS system table.

Bit Position	Description
0	A SQL CHECK constraint is defined on this domain.
1	AUTOMATIC set on insert.
2	AUTOMATIC set on update.
3	If check constraint fails, use name in message.
4	Column is computed an IDENTITY sequence.
5	View column is based on a read-only, or dbkey column.
6	Domain is hidden by user.

H.6.10 RDB\$GRANTED_PROFILES

The RDB\$GRANTED_PROFILES system table contains information about each profile, and role granted to other roles and users. The following table provides information on the columns of the RDB\$GRANTED_PROFILES system table. See also the related RDB\$PROFILES system table.

Column Name	Data Type	Summary Description
RDB\$GRANTEE_PROFILE_ID	integer	This is a unique identifier generated for the parent RDB\$PROFILES row.
RDB\$PROFILE_TYPE	integer	Class of profile information: role (1), default role (2), profile (0).
RDB\$PROFILE_ID	integer	Identification of the profile or role granted to this user.

H.6.11 RDB\$INDEX_SEGMENTS

The RDB\$INDEX_SEGMENTS system table describes the columns that make up an index's key. Each index must have at least one column within the key. The following table provides information on the columns of the RDB\$INDEX_SEGMENTS system table.

Column Name	Data Type	Summary Description
RDB\$INDEX_NAME	char(31)	The name of the index of which this row is a segment.
RDB\$FIELD_NAME	char(31)	The name of a column that participates in the index key. This column name matches the name in the RDB\$FIELD_NAME column of the RDB\$RELATION_FIELDS table.
RDB\$FIELD_POSITION	integer	The ordinal position of this key segment within the total index key. No two segments in the key may have the same RDB\$FIELD_POSITION.
RDB\$FLAGS	integer	A bit mask where Bit 0 is set for descending segments, otherwise the segments are ascending.
RDB\$FIELD_LENGTH	integer	Shortened length of text for compressed indexes.
RDBVMS\$FIELD_MAPPING_LOW	bigint	Shows the lower limit of the mapping range.
RDBVMS\$FIELD_MAPPING_HIGH	bigint	Shows the higher limit of the mapping range.
RDB\$SECURITY_CLASS	char(20)	Reserved for future use.

Column Name	Data Type	Summary Description
RDB\$CARDINALITY	bigint	Prefix cardinality for this and all prior key segments (assumes sorting by ordinal position).

H.6.11.0.1 RDB\$FLAGS Represents flags for RDB\$INDEX_SEGMENTS system table.

Bit Position	Description
0	Is set for descending segments.

H.6.12 RDB\$INDICES

The RDB\$INDICES system table contains information about indexes in the database. The following table provides information on the columns of the RDB\$INDICES system table.

Column Name	Data Type	Summary Description
RDB\$INDEX_NAME	char(31)	A unique index name.
RDB\$RELATION_NAME	char(31)	The name of the table in which the index is used.
RDB\$UNIQUE_FLAG	integer	A value that indicates whether duplicate values are allowed in indexes, as follows: <ul style="list-style-type: none"> 0 If duplicate values are allowed. 1 If no duplicate values are allowed.
RDB\$ROOT_DBK	char(8)	A pointer to the base of the index.
RDB\$INDEX_ID	integer	The identifier of the index.
RDB\$FLAGS	integer	Flags.
RDB\$SEGMENT_COUNT	integer	The number of segments in the key.
RDB\$DESCRIPTION	list of byte varying	A user-supplied description of this index.

Column Name	Data Type	Summary Description
RDB\$EXTENSION_PARAMETERS	list of byte varying	Stores NODE SIZE value, PERCENT FILL value, compression algorithm, and compression run length for this index. Also reserved for other future use.
RDB\$CARDINALITY	bigint	The number of unique entries for a non-unique index. For a unique index, the number is 0.
RDB\$SECURITY_CLASS	char(20)	Reserved for future use.
RDB\$CREATED	date vms	Set when the index is created.
RDB\$LAST_ALTERED	date vms	Set when SQL ALTER INDEX statement is used.
RDB\$INDEX_CREATOR	char(31)	Creator of this index.
RDB\$KEY_CLUSTER_FACTOR	bigint(7)	Sorted Index: The ratio of the number of clump changes that occur when you traverse level-1 index nodes and the duplicate node chains to the number of keys in the index. This statistic is based on entire index traversal. This means last duplicate node of current key is compared with first duplicate node of next key for clump change. Hash Index: The average number of clump changes that occur when you go from system record to hash bucket to overflow hash bucket (if fragmented), and traverse the duplicate node chain for each key. This statistic is based on per key traversal.

Column Name	Data Type	Summary Description
RDB\$DATA_CLUSTER_FACTOR	bigint(7)	Sorted Index: The ratio of the number of clump changes that occur between adjacent dbkeys in duplicate chains of all keys to the number of keys in the index. For unique index, the dbkeys of adjacent keys are compared for clump change. This statistic is based on entire index traversal. This means last dbkey of current key is compared with first dbkey of next key for clump change. Hashed Index: The average number of clump changes that occur between adjacent dbkeys in a duplicate chain for each key. For a unique index, this value will be always 1. This statistic is based on per key traversal.
RDB\$INDEX_DEPTH	integer	Sorted Index: The depth of the B-tree. Hashed Index: This column is not used for hashed indices and is left as 0.

H.6.12.1 RDB\$FLAGS

Represents flags for RDB\$INDICES system table.

Bit Position	Description
0	HASHED index. (If bit is clear, SORTED index.)
1	Index segments are numeric with MAPPING VALUES compression.
2	HASHED ORDERED index. (If bit is clear, HASHED SCATTERED.)
3	Reserved for future use.
4	Run-length compression.
5	Index is disabled or enabled deferred.
6	Build pending (enabled deferred).
7	Reserved for future use.
8	Reserved for future use.

Bit Position	Description
9	Reserved for future use.
10	Reserved for future use.
11	Internal use only.
12	SORTED RANKED index.
13	Prefix cardinalities disabled.
14	Use the full collection algorithm for prefix cardinality.
15	Index generated for a constraint when SET FLAGS 'AUTO_INDEX' was enabled.

H.6.13 RDB\$INTERRELATIONS

The RDB\$INTERRELATIONS system table contains information that indicates the interdependencies of objects in the database. The RDB\$INTERRELATIONS table can be used to determine if an object can be dropped or if some other object depends upon its existence in the database. The following table provides information on the columns of the RDB\$INTERRELATIONS system table.

Column Name	Data Type	Summary Description
RDB\$OBJECT_NAME	char(31)	The name of the object that cannot be dropped or altered because it is used by some other entity in the database.
RDB\$SUBOBJECT_NAME	char(31)	The name of the associated sub-object that cannot be dropped or altered because it is used by another entity in the database.
RDB\$ENTITY_NAME1	char(31)	The name of the entity that depends on the existence of the object identified by the RDB\$OBJECT_NAME and RDB\$SUBOBJECT_NAME.
RDB\$ENTITY_NAME2	char(31)	If used, the name of the entity, together with RDB\$ENTITY_NAME1, that depends on the existence of the object specified in RDB\$OBJECT_NAME and RDB\$SUBOBJECT_NAME.

Column Name	Data Type	Summary Description
RDB\$USAGE	char(31)	The relationship among RDB\$OBJECT_NAME, RDB\$SUBOBJECT_NAME, RDB\$ENTITY_NAME1, and RDB\$ENTITY_NAME2. RDB\$USAGE contains a short description.
RDB\$FLAGS	integer	Flags.
RDB\$CONSTRAINT_NAME	char(31)	This column is the name of a constraint that is referred to from another system table. The value in this column equates to a value for the same column in the RDB\$CONSTRAINTS system table.
RDB\$SECURITY_CLASS	char(20)	Reserved for future use.

H.6.13.1 RDB\$USAGE

Describes the field values for RDB\$USAGE fields for the RDB\$INTERRELATIONS system table.

The table Rdb\$INTERRELATIONS records much of the dependency information when one object references another in the database. Such information is used by DROP ... RESTRICT statements to prevent an object being deleted when it is required by some other object. For instance, a function may use one or more columns from a table in a query. That table and its columns will be recorded with a value in RDB\$USAGE of 'Storage Map'.

Many reported errors include text from the RDB\$USAGE field to explain the type of dependency preventing the DROP from succeeding. These text strings are described in the following table.

Field value	Description
Computed Field	A Computed by or Automatic column references this table, view, column or function.
Constraint	Constraint definition references table, view, column, sequence or function.
Storage Map	Storage map references table and column.
View	View definition requires table, view, column, sequence or function.

Field value	Description
View Field	View column requires table, view, column, sequence or function.
Trigger	Trigger definition requires table, view, column, sequence or function.
RelConstraint	A table (relation) constraint references a table.
Domain Constraint (VALID IF)	A domain constraint (or VALID IF) references this routine or sequence.
Requires	This table, temporary table (with module name), or index is used by a query outline.
Procedure	Procedure definition requires table, view, column, sequence or function.
Function	Function definition requires table, view, column, sequence or function.
Default Txn Reserving	A stored module DECLARE TRANSACTION references a table or view in the RESERVING clause.
Default Txn Evaluating	A stored module DECLARE TRANSACTION references a constraint in the EVALUATING clause.
Lang Semantics	A stored function, procedure or trigger uses wildcard for column list. This includes SELECT * or INSERT with an omitted column list.
Cast As Domain	A CAST function referenced a domain name.
Temp Table Using Domain	A DECLARE LOCAL TEMPORARY TABLE used a domain for a column's data type.
Computed Column in Temp Table	A computed by or automatic column defined by a DECLARE LOCAL TEMPORARY TABLE or DECLARE LOCAL TEMPORARY VIEW references this object.
Module Variable Default Value	A module global DECLARE statement used a DEFAULT clause. Table, view, function, domain and sequence dependencies are recorded.
Referenced by Synonym	When a synonym is created, a dependency is stored.
Default Value	A table column uses a DEFAULT clause. Table, view, function, domain and sequence dependencies are recorded.
Constraint Index	When SET FLAGS 'AUTO_INDEX' is active, any constraint definition will define an index matching the columns of the constraint.

Field value	Description
Module Variable	This module variable uses this domain.
Routine Parameter	Not currently used. Reserved for future use.
Temp Table Reference	A DECLARE LOCAL TEMPORARY TABLE references this table in the LIKE clause.
Storage Map Function	When CREATE STORAGE MAP is executed, a system routine is created to reflect the mapping. Those column dependencies are recorded.

H.6.13.2 RDB\$FLAGS

Represents flags for RDB\$INTERRELATIONS system table.

Bit Position	Description
0	Entity is a module.
1	Object is a module.
2	Entity is a routine.
3	Object is a routine.
4	Entity is a trigger.
5	Object is a trigger.
6	Entity is a constraint.
7	Object is a constraint.
8	Reserved.
9	Reserved.
10	Reserved.
11	Reserved.
12	Reserved.
13	Reserved.
14	Entity is a sequence.
15	Object is a sequence.
16	Entity is a variable.
17	Object is a variable.
18	Entity is an index.
19	Object is an index.

H.6.14 RDB\$MODULES

The RDB\$MODULES system table describes a module as defined by a user. A module can contain a stored procedure or an external function. Each module has a header, a declaration section, and a series of routines. The header and declaration section are defined in RDB\$MODULES. (Each routine is defined by an entry in RDB\$ROUTINES.) A row is stored in the RDB\$MODULES table for each module that is defined by a user. The following table provides information on the columns of the RDB\$MODULES system table.

Column Name	Data Type	Summary Description
RDB\$MODULE_NAME	char(31)	Name of the module.
RDB\$MODULE_OWNER	char(31)	Owner of the module. If the module is an invoker rights module, this column is set to NULL. Otherwise, defines username from this column is used for definers rights checking.
RDB\$MODULE_ID	integer	Unique identifier assigned to this module by Oracle Rdb.
RDB\$MODULE_VERSION	char(16)	Module version and checksum. Allows runtime validation of the module with respect to the database.
RDB\$EXTENSION_PARAMETERS	list of byte varying	Encoded information for module level declarations.
RDB\$MODULE_HDR_SOURCE	list of byte varying	Source of the module header as provided by the definer.
RDB\$DESCRIPTION	list of byte varying	Description of the module.
RDB\$ACCESS_CONTROL	list of byte varying	Access Control List (ACL) to control access to the module. This value can be NULL.
RDB\$SECURITY_CLASS	char(20)	Reserved for future use.
RDB\$CREATED	date vms	Set when the module is created.
RDB\$LAST_ALTERED	date vms	Set when module is altered by the ALTER, RENAME, DROP, GRANT and REVOKE statements.
RDB\$MODULE_CREATOR	char(31)	Creator of this module. Differentiates between OWNER and AUTHORIZATION.
RDB\$VARIABLE_COUNT	integer	Number of global variables.

Column Name	Data Type	Summary Description
RDB\$FLAGS	integer	Flags.

H.6.15 RDB\$OBJECT_SYNONYMS

The RDB\$OBJECT_SYNONYMS system table is created with synonyms are enabled to record the synonym name, type, and target. The following table provides information on the columns of the RDB\$OBJECT_SYNONYMS system table.

Column Name	Data Type	Summary Description
RDB\$CREATED	date vms	Time and date when synonym entry was created.
RDB\$LAST_ALTERED	date vms	Time and date when synonym entry was last altered.
RDB\$DESCRIPTION	list of byte varying	A user-supplied description of the synonym.
RDB\$EXTENSION_PARAMETERS	list of byte varying	Reserved for future use.
RDB\$FLAGS	integer	Flags.
RDB\$OBJECT_TYPE	integer	The type of synonym.
RDB\$SYNONYM_NAME	char(31)	The synonym to be used by queries. This name is unique within the RDB\$OBJECT_SYNONYMS system table.
RDB\$SYNONYM_VALUE	char(31)	name of the object for which the synonym is defined.
RDB\$SYNONYM_CREATOR	char(31)	Creator of the synonym entry.

H.6.15.1 RDB\$FLAGS

Represents flags for RDB\$OBJECT_SYNONYMS system table.

Bit Position	Description
0	When set, this bit indicates that this synonym references another synonym.
1	Reserved for future use.
2	Indicates that the synonym was created by RENAME statement.

H.6.16 RDB\$PARAMETERS

The RDB\$PARAMETERS system table defines the routine interface for each routine stored in RDB\$ROUTINES. Each parameter to a routine (procedure or function) is described by a row in RDB\$PARAMETERS. The following table provides information on the columns of the RDB\$PARAMETERS system table.

Column Name	Data Type	Summary Description
RDB\$PARAMETER_NAME	char(31)	Name of the parameter.
RDB\$PARAMETER_SOURCE	char(31)	Source (domain or table) to the routine containing the parameter.
RDB\$ROUTINE_ID	integer	Unique identifier assigned to the routine containing this parameter by Oracle Rdb.
RDB\$ORDINAL_POSITION	integer	Position in parameter list. Position 0 indicates function result description.
RDB\$PARAMETER_TYPE	integer	Data type of the parameter.
RDB\$PARAMETER_SUB_TYPE	integer	A value that describes the data subtype of RDB\$PARAMETER_TYPE as shown in RDB\$FIELD_SUB_TYPE Help topic.
RDB\$PARAMETER_LENGTH	integer	Length of the parameter.
RDB\$PARAMETER_SCALE	integer	Scale of the data type.
RDB\$PARAMETER_SEG_LENGTH	integer	The length of the segmented string segment. For date-time interval fields, the interval leading field precision.
RDB\$DEFAULT_VALUE2	list of byte varying	Parameter default.
RDB\$FLAGS	integer	Flags.
RDB\$DESCRIPTION	list of byte varying	Description of the parameter.
RDB\$SECURITY_CLASS	char(20)	Reserved for future use.

H.6.16.1 RDB\$FLAGS

Represents flags for RDB\$PARAMETERS system table.

Bit Position	Description
0	IN (read) INOUT (modify).
1	OUT (write) INOUT (modify).
2	Reserved for future use.
3	BY DESCRIPTOR (default is BY REFERENCE).
4	BY VALUE (Bit number 3 is ignored).
5	Reserved for future use.
6	Set if parameter mode is undefined. If Bits 0 and 1 are both clear, then the parameter is the RETURN TYPE of a function.

H.6.17 RDB\$PRIVILEGES

The RDB\$PRIVILEGES system table describes the protection for the database objects. There is one row per grantor, grantee, and privileges combination per entity in the database.

A row is stored in the RDB\$PRIVILEGES table for each user who grants another user privileges for a database object.

If the privilege for a database object was granted without the SQL GRANT option, the row of the grantor and grantee is modified.

The privilege change takes effect at commit time of the command.

Note

The RDB\$PRIVILEGES system table is used only in ANSI databases.

The following table provides information on the columns of the RDB\$PRIVILEGES system table.

Column Name	Data Type	Summary Description
RDB\$SUBOBJECT_ID	integer	The id of the column or routine for which protection is defined. If protection is on a database, module, table, or view, this column is NULL. The value stored in this column must be unique within the database.

Column Name	Data Type	Summary Description
RDB\$OBJECT_ID	integer	The id of the module, table, sequence, or view for which protection is defined. The column is NULL if the protection is defined for the database. The value stored in this column must be unique within the database.
RDB\$GRANTOR	integer	The binary format UIC of the person who defined or changed the privileges. This is usually the UIC of the person who executed the protection command. For an SQL IMPORT statement, the UIC is that of the person who originally defined the protection for the user; not necessarily the person who performed the SQL IMPORT statement.
RDB\$GRANTEE	list of byte varying	The binary format of the UICs of the persons who hold privileges on the database object.
RDB\$PRIV_GRANT	integer	Specifies the access mask of privileges that the grantee has that he can grant to other users.
RDB\$PRIV_NOGRANT	integer	Specifies the access mask of privileges that the grantee has that he can use himself but cannot give to other users.
RDB\$FLAGS	integer	Flags.
RDB\$SECURITY_CLASS	char(20)	Reserved for future use.

H.6.17.1 RDB\$FLAGS

Represents flags for RDB\$PRIVILEGES system table.

Bit Position	Description
0	Privilege is defined for a module and procedure.
1	The data is related to sequences.

H.6.18 RDB\$PROFILES

The RDB\$PROFILES system table contains information about each profile, user and role defined for the database. The following table provides information on the columns of the RDB\$PROFILES system table. See also the related RDB\$GRANTED_PROFILES system table.

Column Name	Data Type	Summary Description
RDB\$CREATED	date vms	time and date when profile entry was created.
RDB\$LAST_ALTERED	date vms	time and date when profile entry was last altered.
RDB\$DESCRIPTION	list of byte varying	Comment for this entry.
RDB\$EXTENSION_PARAMETERS	list of byte varying	Extra definitions such as default transaction.
RDB\$SYSTEM_FLAG	integer	Set to TRUE (1) if this is a system define role or user, otherwise it is set to FALSE (0). When the RDB\$SYSTEM_FLAG is set these entries may not be deleted by a DROP statement.
RDB\$FLAGS	integer	Flags.
RDB\$DEFINE_ACCESS	bigint	Which objects can be defined.
RDB\$CHANGE_ACCESS	bigint	Which objects can be changed.
RDB\$DELETE_ACCESS	bigint	Which objects can be deleted.
RDB\$PROFILE_ID	integer	This is a unique identifier generated for each USER, PROFILE and ROLE added to the database.
RDB\$PROFILE_TYPE	integer	Class of profile information: role (1), user (3), profile (0).
RDB\$PROFILE_NAME	char(31)	Name of the user, profile or role. This name is unique within the RDB\$PROFILES table.
RDB\$PROFILE_CREATOR	char(31)	Creator of entry.

H.6.18.1 RDB\$FLAGS

Represents flags for RDB\$PROFILES system table.

Bit Position	Description
0	The user entry is disabled (ACCOUNT LOCK).
1	Means that the user/role is identified externally.
2	Reserved for future use.
3	This is a system role.
4	Means the user is assigned a profile.

H.6.19 RDB\$QUERY_OUTLINES

The RDB\$QUERY_OUTLINES system table contains query outline definitions that are used by the optimizer to retrieve known query outlines prior to optimization. The following table provides information on the columns of the RDB\$QUERY_OUTLINES system table.

Column Name	Data Type	Summary Description
RDB\$OUTLINE_NAME	char(31)	The query outline name.
RDB\$BLR_ID	char(16)	The BLR hashed identifier. This identifier is generated by the optimizer whenever a query outline is created.
RDB\$MODE	integer	The query mode (MANDATORY or OPTIONAL).
RDB\$FLAGS	integer	Flags.
RDB\$DESCRIPTION	list of byte varying	A user-supplied description of this outline.
RDB\$OUTLINE_BLR	list of byte varying	The compiled query outline.
RDB\$SECURITY_CLASS	char(20)	Reserved for future use.
RDB\$CREATED	date vms	Set when the outline is created.
RDB\$LAST_ALTERED	date vms	Reserved for future use.
RDB\$OUTLINE_CREATOR	char(31)	Creator of this outline.

H.6.19.1 RDB\$FLAGS

Represents flags for RDB\$QUERY_OUTLINES system table.

Bit Position	Description
0	This outline has been invalidated by some action, such as dropping a required table or index.

H.6.20 RDB\$RELATION_CONSTRAINTS

The RDB\$RELATION_CONSTRAINTS system table lists all table-specific constraints. The following table provides information on the columns of the RDB\$RELATION_CONSTRAINTS system table.

Column Name	Data Type	Summary Description
RDB\$CONSTRAINT_MATCH_TYPE	integer	The match type associated with a referential integrity table-specific constraint. This column is reserved for future use. The value is always 0.
RDB\$CONSTRAINT_NAME	char(31)	The name of the constraint defined by the table specified by RDB\$RELATION_NAME. The value in this column equates to a value for the same column in the RDB\$CONSTRAINTS system table.
RDB\$CONSTRAINT_SOURCE	list of byte varying	This text string contains the source of the constraint from the table definition.
RDB\$CONSTRAINT_TYPE	integer	The type of table-specific constraint defined. The values are shown in the RDB\$CONSTRAINT_TYPE Help topic.
RDB\$ERASE_ACTION	integer	The type of referential integrity erase action specified. This column is reserved for future use. The value is always 0.
RDB\$FIELD_NAME	char(31)	The name of the column for which a column-level, table-specific constraint is defined. The column is blank for a table-level constraint.
RDB\$FLAGS	integer	Flags.

Column Name	Data Type	Summary Description
RDB\$MODIFY_ACTION	integer	The type of referential integrity modify action specified. This column is reserved for future use. The value is always 0.
RDB\$REFD_CONSTRAINT_NAME	char(31)	<p>The name of the unique or primary key constraint referred to by a referential integrity foreign key constraint.</p> <p>If the constraint is not a referential integrity constraint or no referential integrity constraint was specified, this column will be null. Otherwise, the value in this column will equate to a value for the same columns in the RDB\$CONSTRAINTS and RDB\$RELATION_CONSTRAINT_FLDS system tables.</p> <p>This column is used to determine the foreign key referenced table name and referenced column names.</p>
RDB\$RELATION_NAME	char(31)	The name of the table on which the specified constraint is defined. The value in this column equates to a value for the same column in the RDB\$RELATIONS system table.
RDB\$SECURITY_CLASS	char(20)	Reserved for future use.

H.6.20.1 RDB\$FLAGS

Represents flags for RDB\$RELATION_CONSTRAINTS system table.

Bit Position	Description
0	This is SQL standard UNIQUE constraint which allows unique values and ignores NULL.

H.6.20.2 RDB\$CONSTRAINT_TYPE

The following table lists the values for the RDB\$CONSTRAINT_TYPE column.

Value	Symbol	Meaning
1	RDB\$K_CON_CONDITION	Requires conditional expression constraint.
2	RDB\$K_CON_PRIMARY_KEY	Primary key constraint.
3	RDB\$K_CON_REFERENTIAL	Referential (foreign key) constraint.
4	RDB\$K_CON_UNIQUE	Unique constraint.
5		Reserved for future use.
6	RDB\$K_CON_NOT_NULL	Not null (missing) constraint.

H.6.21 RDB\$RELATION_CONSTRAINT_FLDS

The RDB\$RELATION_CONSTRAINT_FLDS system table lists the columns that participate in unique, primary, or foreign key declarations for table-specific constraints.

There is one row for each column that represents all or part of a unique, primary, or foreign key constraint.

The following table provides information on the columns of the RDB\$RELATION_CONSTRAINT_FLDS system table.

Column Name	Data Type	Summary Description
RDB\$CONSTRAINT_NAME	char(31)	The name of a constraint for which the specified column participates.
RDB\$FIELD_NAME	char(31)	The name of the column that is all or part of the specified constraint. The value in this column is the same as that stored in the RDB\$RELATION_FIELDS system table.
RDB\$FIELD_POSITION	integer	The ordinal position of the specified column within the column list that declares the unique, primary or foreign key constraint. For column-level constraints, there will always be only one column in the list. The first column in the list has position value 1, the second has position value 2, and so on.

Column Name	Data Type	Summary Description
RDB\$FLAGS	integer	Reserved for future use.
RDB\$SECURITY_CLASS	char(20)	Reserved for future use.

H.6.22 RDB\$RELATION_FIELDS

The RDB\$RELATION_FIELDS system table contains one row for each column in each table. The following table provides information on the columns of the RDB\$RELATION_FIELDS system table.

Column Name	Data Type	Summary Description
RDB\$RELATION_NAME	char(31)	The name of the table that contains the column represented by this row.
RDB\$FIELD_NAME	char(31)	The name of the column represented by this row within the table. Each RDB\$RELATION_FIELDS row that has the same RDB\$RELATION_NAME must have a unique RDB\$FIELD_NAME.
RDB\$FIELD_SOURCE	char(31)	The name of the domain (from the RDB\$FIELD_NAME column within the RDB\$FIELDS table) that supplies the definition for this column.
RDB\$FIELD_ID	integer	An identifier that can be used within the BLR to name the column represented by this row. Oracle Rdb assigns each column an id that is permanent for as long as the column exists within the table.
RDB\$FIELD_POSITION	integer	The ordinal position of the column represented by this row, relative to the other columns in the same table.
RDB\$QUERY_NAME	char(31)	The query name of this column. RDB\$QUERY_NAME can be null.

Column Name	Data Type	Summary Description
RDB\$UPDATE_FLAG	integer	<p>A value that indicates whether a column can be updated:</p> <ul style="list-style-type: none"> 0 If column cannot be updated. 1 If column can be updated.
RDB\$QUERY_HEADER	list of byte varying	<p>The query header of this column for use by SQL. Column attributes in RDB\$RELATION_FIELDS take precedence over RDB\$FIELDS.</p> <p>If the attribute value is missing in RDB\$RELATION_FIELDS, SQL uses the value from RDB\$FIELDS.</p>
RDB\$DESCRIPTION	list of byte varying	A user-supplied description of the contents of this row.
RDB\$VIEW_CONTEXT	integer	<p>For view tables, this column identifies the context variable used to qualify the view column.</p> <p>This context variable must be defined within the row selection expression that defines the view. The context variable appears in the BLR represented by the column RDB\$VIEW_BLR in RDB\$RELATIONS.</p>
RDB\$BASE_FIELD	char(31)	The local name of the column used as a component of a view. The name is qualified by the context variable identified in RDB\$VIEW_CONTEXT.

Column Name	Data Type	Summary Description
RDB\$DEFAULT_VALUE	list of byte varying	<p>The default value used by non-SQL interfaces when no value is specified for a column during a STORE clause.</p> <p>It differs from RDB\$MISSING_VALUE in that it holds an actual column value. Column attributes in RDB\$RELATION_FIELDS take precedence over attributes in RDB\$FIELDS.</p> <p>If the attribute value is missing in RDB\$RELATION_FIELDS, the value from RDB\$FIELDS is used.</p>
RDB\$EDIT_STRING	varchar(255)	The edit string to be used by interactive SQL when printing the column. RDB\$EDIT_STRING can be null.
RDB\$EXTENSION_PARAMETERS	list of byte varying	Reserved for future use.
RDB\$ACCESS_CONTROL	list of byte varying	The access control list for the column.
RDB\$DEFAULT_VALUE2	list of byte varying	The BLR for SQL default value. This value is used when no value is provided in an SQL INSERT statement.
RDBVMS\$SECURITY_AUDIT	integer	A bit mask that indicates the privileges that will be audited for the database, as specified in the RMU Set Audit command.
RDBVMS\$SECURITY_ALARM	integer	A bit mask that indicates the privileges that will produce alarms for the database, as specified in the RMU Set Audit command.
RDB\$SECURITY_CLASS	char(20)	Reserved for future use.

H.6.23 RDB\$RELATIONS

The RDB\$RELATIONS system table names all the tables and views within the database. There is one row for each table or view. The following table provides information on the columns of the RDB\$RELATIONS system table.

Column Name	Data Type	Summary Description
RDB\$RELATION_NAME	char(31)	The name of a table within the database. Each row within RDB\$RELATIONS must have a unique RDB\$RELATION_NAME.
RDB\$RELATION_ID	integer	An identification number used within the BLR to identify a table.
RDB\$STORAGE_ID	integer	A pointer to the database logical area where the data for this table is stored.
RDB\$SYSTEM_FLAG	integer	<p>A value that indicates whether a table is a system table or a user-defined table:</p> <ul style="list-style-type: none"> 0 If a user table. 1 If a system table.
RDB\$DBKEY_LENGTH	integer	<p>The length in bytes of the database key. A database key for a row in a table is 8 bytes, and "n times 8" for a view row, where "n" is the number of tables referred to in the view.</p> <p>If the view does not contain a dbkey, RDB\$DBKEY_LENGTH is 0. This occurs when the view uses GROUP BY, UNION, or returns a statistical value.</p>
RDB\$MAX_VERSION	integer	<p>The number of the current version of the table definition.</p> <p>This value is matched with the RDB\$VERSION column in RDB\$FIELD_VERSIONS to determine the current row format for the table.</p>
RDB\$CARDINALITY	bigint	The number of rows in the table (cardinality).
RDB\$FLAGS	integer	Flags.

Column Name	Data Type	Summary Description
RDB\$VIEW_BLR	list of byte varying	The BLR that describes the row selection expression used to select the rows for the view. If the table is not a view, RDB\$VIEW_BLR is missing.
RDB\$DESCRIPTION	list of byte varying	A user-supplied description of this table or view.
RDB\$VIEW_SOURCE	list of byte varying	The user's source text for the view definition.
RDB\$ACCESS_CONTROL	list of byte varying	The access control policy for the table.
RDB\$EXTENSION_PARAMETERS	list of byte varying	Reserved for future use.
RDB\$CDD_NAME	list of byte varying	The fully qualified name of the dictionary entity upon which the table definition is based, as specified in the SQL clause, FROM PATHNAME.
RDBVMS\$SECURITY_AUDIT	integer	A bit mask that indicates the privileges that will be audited for the table, as specified in the RMU Set Audit command.
RDBVMS\$SECURITY_ALARM	integer	A bit mask that indicates the privileges that produce alarms for the table, as specified in the RMU Set Audit command.
RDB\$SECURITY_CLASS	char(20)	Reserved for future use.
RDBVMS\$SECURITY_AUDIT2	integer	Reserved for future use.
RDBVMS\$SECURITY_ALARM2	integer	Reserved for future use.
RDB\$CREATED	date vms	Set when the table or view is created (for system tables it will be the same as the database creation timestamp).
RDB\$LAST_ALTERED	date vms	Set when SQL ALTER TABLE, CREATE/ALTER STORAGE MAP, ALTER DOMAIN, GRANT, or REVOKE statements cause changes to this system table.
RDB\$RELATION_CREATOR	char(31)	Creator of this system table.

Column Name	Data Type	Summary Description
RDB\$ROW_CLUSTER_FACTOR	bigint(7)	The ratio of the number of clump changes that occur when you sequentially read the rows to the number of rows in a table. If a row is fragmented and part of its fragment is located in a clump different than the current one or immediate next one then it should be counted as a clump change.
RDB\$TYPE_ID	integer	Reserved for future use.

H.6.23.1 RDB\$FLAGS

Represents flags for RDB\$RELATIONS system table.

Bit Position	Description
0	This table is a view.
1	This table is not compressed.
2	The SQL clause, WITH CHECK OPTION, is used in this view definition.
3	Indicates a special internal system table.
4	This view is not an ANSI updatable view.
5	Reserved for future use.
6	Reserved for future use.
7	Reserved for future use.
8	Ignore Bit 1 and use RDB\$STORAGE_MAPS for compression information.
9	Set for temporary table.
10	Set for global temporary table; clear for local temporary table.
11	Set for delete data on commit; clear for preserve data on commit.
12	Reserved for future use.
13	Set if view or table references a local temporary table.
14	Special read-only information table.
15	System table has storage map.
16	View references only temporary table.

H.6.24 RDB\$ROUTINES

The RDB\$ROUTINES system table describes each routine that is part of a stored module or a standalone external routine. An external routine can either be part of a module or standalone (outside the context of a module). The following table provides information on the columns of the RDB\$ROUTINES system table.

Column Name	Data Type	Summary Description
RDB\$ROUTINE_NAME	char(31)	Name of the routine.
RDB\$GENERIC_ROUTINE_NAME	char(31)	Reserved for future use.
RDB\$MODULE_ID	integer	The identifier of the module that contains this routine. If routine is standalone, value is 0.
RDB\$ROUTINE_ID	integer	Unique identifier assigned to this routine.
RDB\$ROUTINE_VERSION	char(16)	Routine version and checksum. Allows runtime validation of the routine with respect to the database.
RDB\$PARAMETER_COUNT	integer	The number of parameters for this routine.
RDB\$MIN_PARAMETER_COUNT	integer	Minimum number of parameters for this routine.
RDB\$ROUTINE_BLR	list of byte varying	The BLR for this routine. If the routine is external, this column is set to NULL.
RDB\$ROUTINE_SOURCE	list of byte varying	Source of the routine as provided by the definer.
RDB\$FLAGS	integer	Flags.
RDB\$SOURCE_LANGUAGE	integer	The RDB\$SOURCE_LANGUAGE section lists the values for this column.
RDB\$DESCRIPTION	list of byte varying	Description of the routine.
RDB\$ACCESS_CONTROL	list of byte varying	The access control list (ACL) to control access to the routine. This value can be NULL.
RDB\$SECURITY_CLASS	char(20)	Reserved for future use.

Column Name	Data Type	Summary Description
RDB\$EXTENSION_PARAMETERS	list of byte varying	Stores interface information about the routine. This includes parameter mappings, the shareable image name, and entry point name.
RDB\$TYPE_ID	integer	Reserved for future use.
RDB\$ROUTINE_OWNER	char(31)	Owner of the routine. This column is only used when the routine is standalone (when RDB\$MODULE_ID is 0) otherwise the value is NULL.
RDB\$CREATED	date vms	Set when the routine is created (the same as the parent module's creation timestamp).
RDB\$LAST_ALTERED	date vms	Set when the routine is modified by the ALTER, RENAME, GRANT, and REVOKE statements.
RDB\$ROUTINE_CREATOR	char(31)	Creator of this routine. Differentiates between AUTHORIZATION and OWNER.

H.6.24.1 RDB\$FLAGS

Represents flags for RDB\$ROUTINES system table.

Bit Position	Description
0	Routine is a function. (Call returns a result.)
1	Routine is not valid. (Invalidated by a metadata change.)
2	The function is not deterministic (that is, the routine is variant). A subsequent invocation of the routine with identical parameters may return different results.
3	Routine can change the transaction state.
4	Routine is in a secured shareable image.
5	Reserved for future use.
6	Routine is not valid. (Invalidated by a metadata change to the object upon which this routine depends. This dependency is a language semantics dependency.)
7	Reserved for future use.

Bit Position	Description
8	External function returns NULL when called with any NULL parameter.
9	Routine has been analyzed (used for trigger dependency tracking).
10	Routine inserts rows.
11	Routine modifies rows.
12	Routine deletes rows.
13	Routine selects rows.
14	Routine calls other routines.
15	Reserved for future use.
16	Routine created with USAGE IS LOCAL clause.
17	Reserved for future use.
18	Reserved for future use.
19	Routine is a SYSTEM routine.
20	Routine generated by Oracle Rdb.
21	BLR\$K_TRANSACTION can change SAVEPOINT state.
	Other bits are reserved for future use.

H.6.24.2 RDB\$SOURCE_LANGUAGE

The following table lists the values for the RDB\$SOURCE_LANGUAGE column.

Value	Language
0	Language undefined
1	Ada
2	C
3	COBOL
4	FORTRAN
5	Pascal
6	Reserved for future use.
7	BASIC
8	GENERAL
9	PL/I

Value	Language
10	SQL - default for stored functions and stored procedures

H.6.25 RDB\$SEQUENCES

The RDB\$SEQUENCES system table contains information about each sequence. The following table provides information on the columns of the RDB\$SEQUENCES system table.

Column Name	Data Type	Summary Description
RDB\$CREATED	date vms	Time sequence was created.
RDB\$LAST_ALTERED	date vms	Last time sequence was altered.
RDB\$ACCESS_CONTROL	list of byte varying	Access control list for this sequence.
RDB\$DESCRIPTION	list of byte varying	Description provided for this sequence.
RDB\$START_VALUE	bigint	Starting value for the sequence.
RDB\$MINIMUM_SEQUENCE	bigint	Minimum value for the sequence.
RDB\$MAXIMUM_SEQUENCE	bigint	Maximum value for the sequence.
RDB\$NEXT_SEQUENCE_VALUE	bigint	Next value available for use for the sequence. This column is a read only COMPUTED BY column. When the sequence is first defined this column returns NULL.
RDB\$INCREMENT_VALUE	integer	Increment value for the sequence. A positive value indicates an ascending sequence, and a negative value indicates a descending sequence.
RDB\$CACHE_SIZE	integer	Number of sequence numbers to allocate and hold in memory. If one (1), then NOCACHE was specified and the values will be allocated one at a time.
RDB\$FLAGS	integer	Flags.
RDB\$SEQUENCE_ID	integer	Unique number assigned to this sequence object. This value is for internal use only.
RDB\$SEQUENCE_NAME	char(31)	Unique name of the sequence.

Column Name	Data Type	Summary Description
RDB\$SEQUENCE_CREATOR	char(31)	Creator of this sequence.

H.6.25.1 RDB\$FLAGS

Represents flags for RDB\$SEQUENCES system table.

Bit Position	Description
0	Sequence will cycle.
1	Sequence is ordered.
2	Sequence is random.
3	Indicates that this is a system sequence and may not be dropped.
4	Indicates that there was no minimum value specified.
5	Indicates that there was no maximum value specified.
6	Indicates that this is a column IDENTITY sequence.
7	Indicates that this sequence will wait for locks.
8	Indicates that this sequence will not wait for locks.

H.6.26 RDB\$STORAGE_MAPS

The RDB\$STORAGE_MAPS system table contains information about each storage map. The following table provides information on the columns of the RDB\$STORAGE_MAPS system table.

Column Name	Data Type	Summary Description
RDB\$MAP_NAME	char(31)	The name of the storage map.
RDB\$RELATION_NAME	char(31)	The name of the table to which the storage map refers.
RDB\$INDEX_NAME	char(31)	The name of the index specified in the SQL clause, PLACEMENT VIA INDEX, of the storage map.
RDB\$FLAGS	integer	Flags.
RDB\$MAP_SOURCE	list of byte varying	The user's source text for the storage map definition.
RDB\$DESCRIPTION	list of byte varying	A user-supplied description of the storage map.

Column Name	Data Type	Summary Description
RDB\$EXTENSION_PARAMETERS	list of byte varying	Lists the column names for vertical record partitioning.
RDB\$VERTICAL_PARTITION_INDEX	integer	A counter that indicates the number of vertical record partitions. If vertical record partitioning is used, there is one RDB\$STORAGE_MAPS for each vertical partition.
RDB\$VERTICAL_PARTITION_NAME	char(31)	Name of the vertical record partition.
RDB\$SECURITY_CLASS	char(20)	Reserved for future use.

H.6.26.1 RDB\$FLAGS

Represents flags for RDB\$STORAGE_MAPS system table.

Bit Position	Description
0	If map is for a mixed format area.
1	If map enables compression.
2	Partition key cannot be updated.
3	Reserved for future use.
4	User named this partition.
5	Override used for strict partitioning - NO REORGANIZE.

H.6.27 RDB\$STORAGE_MAP_AREAS

The RDB\$STORAGE_MAP_AREAS system table contains information about each storage area to which a storage map refers. The following table provides information on the columns of the RDB\$STORAGE_MAP_AREAS system table.

Column Name	Data Type	Summary Description
RDB\$MAP_NAME	char(31)	The name of the storage map.
RDB\$AREA_NAME	char(31)	The name of the storage area referred to by the storage map.

Column Name	Data Type	Summary Description
RDB\$ROOT_DBK	char(8)	A pointer to the root of the SORTED index, if it is a SORTED index.
RDB\$ORDINAL_POSITION	integer	The order of the storage area represented by this row in the map.
RDB\$STORAGE_ID	integer	For a table, a pointer to the database logical area. For a hashed index, a pointer to the system record.
RDB\$INDEX_ID	integer	A pointer to the index logical area.
RDB\$STORAGE_BLR	list of byte varying	The BLR that represents the SQL clause, WITH LIMIT OF, in the storage map definition.
RDB\$DESCRIPTION	list of byte varying	Description of this partition.
RDB\$EXTENSION_PARAMETERS	list of byte varying	Lists table names and column names that are referenced by segmented string storage maps.
RDB\$VERTICAL_PARTITION_INDEX	integer	For LIST storage maps, the value indicates the relationship between areas of a LIST storage map area set.
RDB\$FLAGS	integer	Flags.
RDB\$SECURITY_CLASS	char(20)	Reserved for future use.
RDB\$PARTITION_NAME	char(31)	Name of the index or storage map partition.

H.6.27.1 RDB\$FLAGS

Represents flags for RDB\$STORAGE_MAP_AREAS system table.

Bit Position	Description
0	If Bit 0 is clear, the LIST storage area set is filled randomly. If Bit 0 is set, the LIST storage area set is filled sequentially.
1	User named this partition.
2	BUILD PARTITION is required.
3	Deferred build using NOLOGGING.
4	Cardinality needs to be updated on build.

H.6.28 RDB\$SYNONYMS

The RDB\$SYNONYMS system table connects the user-visible name of an object to the stored name of an object. The user-visible name of an object might be replicated in multiple schemas, whereas the stored name of an object is unique across all schemas and catalogs. This table is present only in databases that have the SQL multiscHEMA feature enabled.

Unlike rows in other system tables, the rows in the RDB\$SYNONYMS system table are compressed. The following table provides information on the columns of the RDB\$SYNONYMS system table.

Column Name	Data Type	Summary Description
RDB\$SCHEMA_ID	integer	The RDB\$CATALOG_SCHEMA_ID of the schema to which this object belongs.
RDB\$USER_VISIBLE_NAME	char(31)	The name of an object as it appears to the user.

Column Name	Data Type	Summary Description
RDB\$OBJECT_TYPE	integer	<p>A value that represents the type of an object, as follows:</p> <ul style="list-style-type: none"> • 8 A constraint. • 19 A domain (global field). • 26 An index. • 31 A system table. • 36 A view. • 60 A sequence. • 67 A storage map. • 81 A trigger. • 117 A collating sequence. • 180 An outline. • 192 A type.
RDB\$STORED_NAME	char(31)	The name of an object as is actually stored in the database.
RDB\$SECURITY_CLASS	char(20)	Reserved for future use.

H.6.29 RDB\$TRIGGERS

The RDB\$TRIGGERS system table describes the definition of a trigger. The following table provides information on the columns of the RDB\$TRIGGERS system table.

Column Name	Data Type	Summary Description
RDB\$DESCRIPTION	list of byte varying	A user-supplied text string describing the trigger.
RDB\$FLAGS	integer	Flags.
RDB\$RELATION_NAME	char(31)	The name of the table for which this trigger is defined. The trigger may be selected on an update to the named table (qualified by the columns described in the RDB\$TRIGGER_FIELD_NAME_LIST). This table is used as a subject table for all contexts that refer to it.
RDB\$TRIGGER_ACTIONS	list of byte varying	A text string containing all the sets of triggered actions defined for this trigger. The string consists of one or more sets of clumplets, one set for each triggered action.
RDB\$TRIGGER_CONTEXTS	integer	The context number used within the triggered action BLR to map the triggered action BLR to the current context of the triggering update statement.
RDB\$TRIGGER_FIELD_NAME_LIST	list of byte varying	A text string composed of a count field and one or more counted strings. The count is an unsigned word that represents the number of strings in the list. The counted strings are ASCII names that represent column names. If the trigger is of event type UPDATE, it will be evaluated if one or more of the specified columns has been modified.

Column Name	Data Type	Summary Description
RDB\$TRIGGER_NAME	char(31)	The name of a trigger. This name must be a unique trigger name within the database.
RDB\$TRIGGER_NEW_CONTEXT	integer	A context number used within the triggered action's BLR to refer to the new row values for the subject table for an UPDATE event.
RDB\$TRIGGER_OLD_CONTEXT	integer	A context number used within the triggered action's BLR to refer to the old row values of the subject table that existed before an UPDATE event.
RDB\$TRIGGER_SOURCE	list of byte varying	An optional text string for the trigger definition. The string is not used by the database system. It should reflect the full definition of the trigger. This column is used by the interfaces to display the trigger definition.
RDB\$TRIGGER_TYPE	integer	The type of trigger, as defined by the combination of the trigger action time and the trigger event. Action times are BEFORE and AFTER, and events are INSERT, DELETE, and UPDATE. The values that represent the type of trigger are shown in the RDB\$TRIGGER_TYPE Values section below.
RDB\$SECURITY_CLASS	char(20)	Reserved for future use.
RDB\$CREATED	date vms	Set when the trigger is created.
RDB\$LAST_ALTERED	date vms	Set when SQL ALTER TRIGGER statement is used.
RDB\$TRIGGER_CREATOR	char(31)	Creator of this trigger.
RDB\$EXTENSION_PARAMETERS	list of byte varying	Extension parameters.
RDB\$TRIGGER_ID	integer	Unique id for trigger with child records.

H.6.29.1 RDB\$FLAGS

Represents flags for RDB\$TRIGGERS system table.

Bit Position	Description
0	Trigger is currently disabled.
1	Invalid due to changed schema.
2	Referenced table was altered.

H.6.29.2 RDB\$TRIGGER_TYPE Values

The following table lists the values for the RDB\$TRIGGER_TYPE column of the RDB\$TRIGGERS system table and the different types of triggers they represent.

Numeric Value	Symbolic Value	Description
1	RDB\$K_BEFORE_STORE	Trigger is evaluated before an INSERT.
2	RDB\$K_BEFORE_ERASE	Trigger is evaluated before a DELETE.
3	RDB\$K_BEFORE_MODIFY	Trigger is evaluated before an UPDATE.
4	RDB\$K_AFTER_STORE	Trigger is evaluated after an INSERT.
5	RDB\$K_AFTER_ERASE	Trigger is evaluated after a DELETE.
6	RDB\$K_AFTER_MODIFY	Trigger is evaluated after an UPDATE.

H.6.30 RDB\$VIEW_RELATIONS

The RDB\$VIEW_RELATIONS system table lists all the tables that participate in a given view. There is one row for each table or view in a view definition. The following table provides information on the columns of the RDB\$VIEW_RELATIONS system table.

Column Name	Data Type	Summary Description
RDB\$VIEW_NAME	char(31)	Names a view or table that uses another table. The value of RDB\$VIEW_NAME is normally a view name, but might also be the name of a table that includes a column computed using a statistical expression.
RDB\$RELATION_NAME	char(31)	The name of a table used to form the view.
RDB\$VIEW_CONTEXT	integer	An identifier for the context variable used to identify a table in the view. The context variable appears in the BLR represented by the column RDB\$VIEW_BLR in RDB\$RELATIONS.
RDB\$SECURITY_CLASS	char(20)	Reserved for future use.

H.6.31 RDB\$TRIGGER_ACTIONS

The RDB\$TRIGGER_ACTIONS system table describes a single trigger action for a trigger.

Note

Triggers created by prior versions of Oracle Rdb are stored wholly in the RDB\$TRIGGERS table. Only new triggers created with Rdb V7.3 and later (which includes IMPORT DATABASE) are stored as multiple RDB\$TRIGGER_ACTION rows.

The following table provides information on the columns of the RDB\$TRIGGER_ACTIONS system table.

Column Name	Data Type	Summary Description
RDB\$ACTION_BLR	list of byte varying	Trigger actions BLR
RDB\$CONDITION_BLR	list of byte varying	Trigger condition (boolean) BLR
RDB\$DESCRIPTION	list of byte varying	Description text

Column Name	Data Type	Summary Description
RDB\$TRIGGER_ID	integer	Unique id for trigger
RDB\$ORDINAL_POSITION	integer	Assigned order of the trigger action within the trigger definition
RDB\$FLAGS	integer	Flags. See definitions below.
RDB\$ACTION_NAME	char(31)	Unique name of the trigger action. This value may be system assigned

H.6.31.1 RDB\$FLAGS

Represents flags for RDB\$TRIGGER_ACTIONS system table.

Bit Position	Description
0	Action is executed for EACH ROW if true, otherwise just once EACH STATEMENT
1	Action does not trigger further actions
2	TRACE statement is enabled always
	All other flags are reserved for future use

H.6.32 RDB\$WORKLOAD

The RDB\$WORKLOAD system table is an optional system table (similar to RDB\$SYNONYMS and RDB\$CATALOG_SCHEMA). It is created when the database attribute WORKLOAD COLLECTION IS ENABLED is specified on an SQL CREATE or ALTER DATABASE statement. Once created, this system table can never be dropped.

The following table provides information on the columns of the RDB\$WORKLOAD system table.

Column Name	Data Type	Summary Description
RDB\$CREATED	date vms	Time workload entry was created.
RDB\$LAST_ALTERED	date vms	Last time statistics were updated.
RDB\$DUPLICITY_FACTOR	bigint(7)	Value ranges from 1.0 to table cardinality. Number of duplicate values for an interesting column group (RDB\$FIELD_GROUP).

Column Name	Data Type	Summary Description
RDB\$NULL_FACTOR	integer(7)	Value ranges from 0.0 to 1.0. This is the proportion of table rows that have NULL in one or more columns of an interesting column group.
RDB\$RELATION_ID	integer	Base table identifier.
RDB\$FLAGS	integer	Reserved for future use.
RDB\$FIELD_GROUP	char(31)	Contains up to 15 sorted column identifiers.
RDB\$SECURITY_CLASS	char(20)	Reserved for future use.

Information Tables

I.1 Introduction to Information Tables

Information tables display internal information about storage areas, after-image journals, row caches, database users, the database root, and database character sets. Once the information tables are created, you can query them with the SQL interface.

Information tables are special read-only tables that can be created in an Oracle Rdb database and used to retrieve database attributes that are not stored in the existing relational tables. Information tables allow interesting database information, which is currently stored in an internal format, to be displayed as a relational table.

The script, INFO_TABLES.SQL, is supplied as a part of the Oracle Rdb kit. The INFO_TABLES.SQL file is in the SQL\$SAMPLE directory. Table I-1 lists the information tables that are supported in Oracle Rdb.

storage areas.)

Table I-1 Supported Information Tables

RDB\$CACHES	Displays information about the database row caches.
RDB\$CHARACTER_SETS	Displays information about the Oracle Rdb character sets.
RDB\$DATABASE_JOURNAL	Displays information about the default journal.
RDB\$DATABASE_ROOT	Displays information about the database root.
RDB\$DATABASE_USERS	Displays information about the database users.
RDB\$JOURNALS	Displays information about the database journal files.
RDB\$LOGICAL_AREAS	Displays information about the logical areas.
RDB\$NLS_CHARACTER_SETS	Displays the mapping of Oracle NLS character sets to Oracle Rdb character sets.
RDB\$SESSION_PRIVILEGES	Displays the names of the enabled privileges for this user session.
RDB\$STORAGE_AREAS	Displays information about the database storage areas.

For additional information about these information tables on OpenVMS, see the ORACLE_RDBnn topic and select the Information_Tables subtopic (where nn is the version number if using multiversion) in the Oracle Rdb command-line Help or see the next section of this chapter.

Examples

Example 1: Querying an information tables

The following example shows how to query one of the information tables created by the INFO_TABLES.SQL script.

```
SQL> SELECT * FROM RDB$LOGICAL_AREAS WHERE RDB$LOGICAL_AREA_NAME='JOBS';
RDB$LOGICAL_AREA_ID  RDB$AREA_ID  RDB$RECORD_LENGTH  RDB$THRESHOLD1_PERCENT
RDB$THRESHOLD2_PERCENT  RDB$THRESHOLD3_PERCENT  RDB$ORDERED_HASH_OFFSET
RDB$RECORD_TYPE  RDB$LOGICAL_AREA_NAME
                95                7                41                0
                0                0                0
                1  JOBS

1 row selected
```

Example 2: Queries to detect growth of storage area files

The database administrator can list storage areas where the current allocation of an area has exceeded the initial allocation. This information can be vital when managing performance in mixed areas. With mixed areas every storage area extension causes extra I/O for HASHED index access.

```

SQL> select RDB$AREA_NAME as NAME edit using 'T(15)',
cont> RDB$INITIAL_ALLOCATION as INITIAL edit using 'Z(9)',
cont> RDB$CURRENT_ALLOCATION as CURRENT edit using 'Z(9)',
cont> RDB$EXTEND_COUNT as EXTENDS edit using 'Z(9)',
cont> RDB$LAST_EXTEND as LAST_EXT_DATE
cont> from RDB$STORAGE_AREAS
cont> where (RDB$CURRENT_ALLOCATION > RDB$INITIAL_ALLOCATION + 1)
cont> and (RDB$AREA_NAME <> ' ');
NAME                INITIAL      CURRENT      EXTENDS      LAST_EXT_DATE
RDB$SYSTEM           102        3724         15      14-AUG-2003 13:53:36.81
SALARY_HISTORY       270        1270         1        6-AUG-2003 11:47:11.00
2 rows selected

```

This query shows that the system area has extended by almost 37 percent since the database was created. Why is that? Are developers creating tables without mapping the data to user defined storage areas? Area SALARY_HISTORY is a mixed area that has extended. The initial allocation is no longer adequate for this area. Was there a period in August where a lot of data was inserted into these areas? The database administrator can schedule maintenance time to resize the SALARY_HISTORY storage area.

Note

The query adds one to the initial allocation to eliminate areas where a spam page has been added and the current allocation is the initial allocation plus one block. The comparison of RDB\$AREA_NAME to a blank space eliminates snapshot areas from the query.

A similar query that shows snapshot files that have grown beyond the initial allocation is:

```

SQL> select RDB$AREA_FILE as SNAP_NAME edit using 'T(50)',
cont> RDB$INITIAL_ALLOCATION as INITIAL edit using 'Z(9)',
cont> RDB$CURRENT_ALLOCATION as CURRENT edit using 'Z(9)'
cont> from RDB$STORAGE_AREAS
cont> where (RDB$CURRENT_ALLOCATION > RDB$INITIAL_ALLOCATION)
cont> and (RDB$AREA_NAME = ' ');
SNAP_NAME                                INITIAL      CURRENT
DKD300:[SQLUSER71]MF_PERS_DEFAULT.SNP;1      50        1623
DKD300:[SQLUSER71]DEPARTMENTS.SNP;1          10        5000
2 rows selected

```

Large snapshot files are mainly caused by old active transactions, or an initial allocation size that was too small. This query gives the database administrator pointer to an area that needs investigation. Queries such as these can be executed at regular intervals to detect growth trends.

Example 3: Comparing table cardinality with cache sizes

Queries can be run periodically to check that the allocated cache sizes are adequate for the current table size. Know that a cache was too small for the table and taking corrective action can reduce cache collisions.

```
SQL> select A.RDB$CACHE_NAME, A.RDB$CACHE_SIZE, B.RDB$CARDINALITY
cont> from RDB$CACHES A, RDB$RELATIONS B
cont> where A.RDB$CACHE_NAME = B.RDB$RELATION_NAME;
A.RDB$CACHE_NAME      A.RDB$CACHE_SIZE      B.RDB$CARDINALITY
EMPLOYEES              100                   103
DEPARTMENTS           26                    26
DEGREES               20                   165
TOO_BIG               500                  10000
TOO_SMALL            1000                   100
5 rows selected
```

In this example the tables **TOO_BIG** and **DEGREES** can only cache 5 percent and 12 percent respectively of the total table. Perhaps the cache size needs to be increased? Conversely, table **TOO_SMALL** appears to have a cache far too large. Maybe this cache was incorrectly configured or the table has shrunk over time? The current cache size is probably wasting memory.

Example 4: Converting logical DBKEY areas to names

Tools such as **RMU Verify** or **RMU Show Statistics** often display a logical area **DBKEY**. For example consider this output from a stall message screen in **RMU Show Statistics**:

```
202003A5:5 16:25:18.51 waiting for record 79:155:6
```

The database administrator can use **RMU /DUMP/LAREA=RDB\$AIP** to get a list of all the logical areas and their area numbers. However, the following simple query on the **RDB\$LOGICAL_AREAS** information table can be used instead.

```
SQL> select rdb$logical_area_id, rdb$area_id, rdb$logical_area_name
cont> from rdb$logical_areas
cont> where rdb$logical_area_id=79;
RDB$LOGICAL_AREA_ID  RDB$AREA_ID  RDB$LOGICAL_AREA_NAME
79                  3      EMPLOYEES
1 row selected
```

The query shows that this stall is on table **EMPLOYEES** and it resides in physical area number 3. The database administrator can use this information to dump the database page.

```
$ RMU/DUMP/AREA=3/START=155/END=155/OUTPUT=t.t mf_personnel
```

I.1.1 Restrictions for Information Tables

The following restrictions apply to information tables:

- You cannot alter the information tables. The table names and column names must remain unchanged.
- You may not use ALTER TABLE on an Information Table. An exception will be raised.

```
SQL> alter table RDB$DATABASE_USERS add new_col int;  
%RDB-E-NO_META_UPDATE, metadata update failed  
-RDMS-E-NOMETSYSREL, operation illegal on system defined metadata
```

- Information tables are READ ONLY and in general the information is translated from other (non-table) internal data structures.
- Repeatable reads are not guaranteed for these tables. They reflect the current active database. For instance, repeated queries on RDB\$DATABASE_USERS will show the set of users as they attach and disconnect from the database.

I.2 All Information Tables

Listed below are all of the current information tables.

I.2.1 All_Information_Tables

Included with each information table is a list of the columns in that table and their description.

I.2.1.1 RDB\$STORAGE_AREAS

Displays information about the database storage areas.

Column Name	Data Type	Description
RDB\$LAST_BACKUP	date vms	Date of last backup
RDB\$LAST_INCREMENT_ RESTORE	date vms	Date of last restore
RDB\$INITIAL_ALLOCATION	integer	Allocation
RDB\$CURRENT_ALLOCATION	integer	Maximum page number
RDB\$AREA_ACCESS_MODE	integer	Access mode
RDB\$THRESHOLD1_BYTES	integer	First threshold value
RDB\$THRESHOLD2_BYTES	integer	Second threshold value

Column Name	Data Type	Description
RDB\$THRESHOLD3_BYTES	integer	Third threshold value
RDB\$PAGE_SIZE	integer	Page size
RDB\$AREA_MIN_EXTENT	integer	Minimum page extent
RDB\$AREA_MAX_EXTENT	integer	Maximum page extent
RDB\$AREA_PERCENT_EXTENT	integer	Percent growth
RDB\$EXTEND_COUNT	integer	Number of times area has been extended
RDB\$SPAM_INTERVAL	integer	Number of pages per SPAM page
RDB\$JOURNAL_SEQUENCE	integer	AIJ recovery version number
RDB\$MAX_PAGE_FREE_SPACE	integer	Maximum free space on new page
RDB\$MAX_ROW_LENGTH	integer	Largest segment that can be stored on a page
RDB\$SEQUENCE	integer	Sequence number
RDB\$AREA_ID	integer	Area ID
RDB\$LINKED_AREA_ID	integer	For live storage area, ID of snapshot area. For snapshot area, ID of live storage area
RDB\$FLAGS	integer	Flags
RDB\$THRESHOLD1_PERCENT	integer	First threshold percent value
RDB\$THRESHOLD2_PERCENT	integer	Second threshold percent value
RDB\$THRESHOLD3_PERCENT	integer	Third threshold percent value
RDB\$AREA_FILE	char(255)	Storage area file name
RDB\$CACHE_ID	integer	Row cache ID
RDB\$AREA_NAME	char(31)	Storage area name
RDB\$CLUMP_PAGE_COUNT	integer	The number of pages per logical area clump (UNIFORM area)
RDB\$PAGE_LENGTH	integer	Page length in bytes
RDB\$PAGES_PER_SPAM_PAGE_P1	integer	Number of data pages per SPAM page + 1
RDB\$SPAM_PAGE_VECTOR_LENGTH	integer	SPAM page vector length in bytes
RDB\$PAGE_PAD_LENGTH	integer	Page padding length in bytes

Column Name	Data Type	Description
RDB\$MAX_SEGMENT_LENGTH	integer	Largest length to which an existing segment can grow
RDB\$BACKUP_STATISTICS	bigint	Backup-specific statistics
RDB\$SNAPS_ENABLED_TSN	bigint	TSN of oldest snapshots. If this is a live storage area that has snapshots allowed, this field contains the TSN of the most recent transaction that enabled snapshots for or performed exclusive updates to this area
RDB\$COMMIT_TSN	bigint	TSN to which this area is consistent. If the area is marked inconsistent, this is the TSN level to which the area is currently consistent
RDB\$BACKUP_TSN	bigint	TSN of last full area backup. If this field contains zero, this area has not been backed up
RDB\$INCR_BACKUP_TSN	bigint	TSN of last incremental area backup. If this field contains zero, this area has not been incrementally backed up
RDB\$LAST_EXTEND	date vms	Date of last area extend

1.2.1.1.1 RDB\$FLAGS Represents flags for RDB\$STORAGE_AREAS information table.

Bit Position	Description
0	Page format is mixed
1	Checksum calculation is enabled
2	This is a snapshot area
3	Area has snapshots
4	Snapshots are enabled
5	Area has space management pages
6	SPAM pages are enabled
7	Allow multi-volume disk extents
8	Extent is enabled
9	Area is corrupt

Bit Position	Description
10	Area is inconsistent
11	Reserved for future use
12	Page level locking
13	Reserved for future use
14	Reserved for future use
15	Row cache is enabled
16	Area has been restructured. If TRUE, this area has been restructured and may only be readied by a restructuring process
17	Roll forward quiet-point is enabled
18	Area is corrupt due to Hot Standby. If TRUE, this storage area may have been corrupted by an aborted batch update transaction

I.2.1.2 RDB\$JOURNALS

Displays information about the database journal files.

Column Name	Data Type	Description
RDB\$LAST_BACKUP	date vms	Date of last AIJ backup
RDB\$STATUS	integer	AIJ file is inaccessible
RDB\$ALLOCATION	integer	Allocation
RDB\$EXTENT	integer	Extent
RDB\$FLAGS	integer	Flags
RDB\$JOURNAL_NAME	char(31)	Journal name
RDB\$LAST_BACKUP_SEQUENCE	integer	Last version number backed up
RDB\$LAST_ACTIVATED	date vms	Date last activated
RDB\$BACKUP_EDIT_STRING	varchar(255)	Backup editname file
RDB\$DEFAULT_FILENAME	char(255)	Default journal name
RDB\$FILENAME	char(255)	Journal name
RDB\$BACKUP_FILENAME	char(255)	Backup file name
RDB\$SEQUENCE_NUMBER	integer	Current AIJ sequence number
RDB\$STATE	char(31)	State ("Current" or "Latent")

I.2.1.2.1 RDB\$FLAGS Represents flags for RDB\$JOURNALS information table.

Bit Position	Description
0	Initialization in progress
1	Backup in progress
2	AIJ file has been modified
3	AIJ restored from existing file
4	Hard data loss resulted from fail over
5	Soft data loss resulted from fail over
6	New version of journal created
7	Journal has been overwritten
8	Backup failed prematurely
9	Journal created due to switch-over suspension
10	AIJ file block has been assigned
11	Journal created for recovery synchronization
12	Extensible AIJ truncation error on backup

I.2.1.3 RDB\$CACHES

Displays information about the database row caches.

Column Name	Data Type	Description
RDB\$ALLOCATION	integer	Allocation
RDB\$EXTENT	integer	Extent
RDB\$CACHE_ID	integer	Cache ID
RDB\$FLAGS	integer	Flags
RDB\$ROW_LENGTH	integer	Row length
RDB\$CACHE_SIZE	integer	Cache size
RDB\$WINDOW_COUNT	integer	Window count
RDB\$NUM_RESERVED_ROWS	integer	Number of reserved rows
RDB\$NUM_SWEEP_ROWS	integer	Number of sweep rows
RDB\$CACHE_NAME	char(31)	Cache name
RDB\$LOCATION	char(255)	Cache location

Column Name	Data Type	Description
RDB\$SNAP_CACHE_SIZE	integer	Number of snapshot record slots in cache
RDB\$PHYSICAL_MEMORY	bigint	Physical memory in bytes
RDB\$SWEEP_FREE_PCT	integer	Percent of slots to free during sweep
RDB\$SWEEP_BATCH_COUNT	integer	Number of records to batch write
RDB\$MAX_WS_COUNT	integer	Maximum working set count
RDB\$SWEEP_THRESH_HIGH	integer	% of marked slots to start sweep
RDB\$SWEEP_THRESH_LOW	integer	% of marked slots to stop sweep

I.2.1.3.1 RDB\$FLAGS Represents flags for RDB\$CACHES information table.

Bit Position	Description
0	Shared memory is system
1	Large memory is enabled
2	Row replacement is enabled
3	Device/directory is defined
4	Updated rows to database
5	Updated rows to backing file
6	All rows to backing file
7	Snapshots are enabled
8	Use memory-resident section for cache

I.2.1.4 RDB\$DATABASE_ROOT

Displays information about the database root.

Column Name	Data Type	Description
RDB\$CREATED	date vms	Date of database creation
RDB\$LAST_FULL_BACKUP_TIME	date vms	Date of last complete full backup
RDB\$MAJOR_VERSION	integer	Major software version
RDB\$MINOR_VERSION	integer	Minor software version

Column Name	Data Type	Description
RDB\$PRIOR_MAJOR_VERSION	integer	Major software version for database converted with /nocommit
RDB\$PRIOR_MINOR_VERSION	integer	Minor software version for database converted with /nocommit
RDB\$FLAGS	integer	Flags
RDB\$MAX_USERS	integer	Number of users
RDB\$MAX_CLUSTER_NODES	integer	Number of cluster nodes
RDB\$DEFAULT_NUM_OF_BUFFERS	integer	Number of buffers
RDB\$MAX_RECOVERY_BUFFERS	integer	Number of recover buffers
RDB\$BUFFER_SIZE	integer	Buffer size
RDB\$GLOBAL_BUFFER_COUNT	integer	Number of global buffers
RDB\$ALG_COUNT	integer	Adjustable lock granularity count
RDB\$LOCK_TIMEOUT_INTERVAL	integer	Lock timeout interval in seconds
RDB\$CHECKPOINT_INTERVAL_BLKs	integer	Checkpoint block interval
RDB\$CHECKPOINT_TIMED_SECONDS	integer	Checkpoint time interval
RDB\$TRANSACTION_INTERVAL	integer	Commit transaction interval
RDB\$CLOSE_MODE	integer	Close mode
RDB\$CLOSE_INTERVAL	integer	Close interval
RDB\$ASYNC_PREFETCH_DEPTH	integer	Async prefetch depth
RDB\$D_ASYNC_PREFETCH_DEPTH	integer	Detected async prefetch depth
RDB\$D_ASYNC_PREFETCH_THRESHOLD	integer	Detected async prefetch threshold
RDB\$CLEAN_BUFFER_COUNT	integer	Clean buffer count
RDB\$MAX_BUFFER_COUNT	integer	Maximum buffer count
RDB\$MIN_PAGE_SIZE_BLOCKS	integer	Minimum area page block count

Column Name	Data Type	Description
RDB\$MAX_PAGE_SIZE_BLOCKS	integer	Maximum area page block count
RDB\$TRANSACTION_MODE_FLAGS	integer	Transaction mode
RDB\$ALG_FACTOR_0	integer	Adjustable locking granularity factor 0
RDB\$ALG_FACTOR_1	integer	Adjustable locking granularity factor 1
RDB\$ALG_FACTOR_2	integer	Adjustable locking granularity factor 2
RDB\$ALG_FACTOR_3	integer	Adjustable locking granularity factor 3
RDB\$ALG_FACTOR_4	integer	Adjustable locking granularity factor 4
RDB\$ALG_FACTOR_5	integer	Adjustable locking granularity factor 5
RDB\$ALG_FACTOR_6	integer	Adjustable locking granularity factor 6
RDB\$ALG_FACTOR_7	integer	Adjustable locking granularity factor 7
RDB\$AUDIT_FILENAME	char(255)	Audit journal file name
RDB\$ROOT_FILENAME	char(255)	Database root file name
RDB\$RUJ_LOCATION	char(255)	Default recovery-unit journal file name
RDB\$CACHE_LOCATION	char(255)	Default device/directory specification for record cache files
RDB\$MAX_PAGES_IN_BUFFER	integer	Maximum number of pages in a buffer
RDB\$RCS_SWEEP_INTERVAL	integer	Row cache server (RCS) sweep interval (in seconds)
RDB\$RCS_CKPT_TIME	integer	Time interval to force row cache server (RCS) to checkpoint. This field contains the number of seconds that pass before RCS is forced to perform another checkpoint
RDB\$LAST_FULL_RESTORE	date vms	Date of last complete full restore
RDB\$AIJ_ACTIVATION_ID	bigint	AIJ journaling activation identifier

Column Name	Data Type	Description
RDB\$RCVR_ACTIVATION_ID	bigint	RCVR journaling activation identifier
RDB\$OPER_CLASS	integer	Operator notification classes
RDB\$PRESTART_TXN_TIMEOUT	integer	Seconds until prestarted transaction is abandoned. Zero means no abandon timer
RDB\$DB_REPLICATED	integer	AIJ log roll forward server started
RDB\$UNIQUE_VERSION	date vms	Physical database create time-and-date. This field contains the time-and-date stamp when this physical database was created / restored / copied
RDB\$AUDIT_FLAGS	integer	Audit event class flags

I.2.1.4.1 RDB\$FLAGS Represents flags for RDB\$DATABASE_ROOT information table.

Bit Position	Description
0	Single file database
1	Open mode
2	Log server mode
3	Snapshots are deferred
4	Global buffers are enabled
5	Carryover locks are enabled
6	Statistics collection is enabled
7	Fast commit is enabled
8	AIJ commit optimization is enabled
9	RUJ is corrupt
10	Database is corrupt
11	Fast incremental backup is enabled
12	Async prefetch is enabled
13	Async batch writes are enabled
14	Lock partitioning is enabled
15	Page transfer via memory

Bit Position	Description
16	Detected async prefetch is enabled
17	Shared memory is system
18	Database has been modified (TSN allocated)
19	Database conversion has been committed
20	Row cache server (RCS) checkpoints to database
21	RCS checkpoints to backing store files
22	RCS checkpoints marked and unmarked to RDC
23	Global buffers should be in VLM
24	Row cache RUJ global buffers are disabled
25	LogMiner feature is enabled
26	Prestarted transactions are enabled
27	VMS Galaxy shared memory is enabled
28	Use of P1 memory enabled for PIO code
29	A change has been made that precludes an incremental backup
30	Security audit enabled
31	Security alarm enabled flag
32	Audit/alarm on first access only
33	Audit synchronous flush flag
34	Standby database is opened read-only
35	Roll-forward quiet-point enabled
36	Database has been modified while LSS inactive
37	Continuous LogMiner feature enabled
38	AIJ buffer objects enabled
39	OBJMAN buffer objects enabled
40	Page buffer objects enabled
41	RUJ buffer objects enabled

I.2.1.5 RDB\$DATABASE_JOURNAL

Displays information about the default journal information.

Column Name	Data Type	Description
RDB\$CONDITION	integer	AIJ status
RDB\$DEFAULT_ALLOCATION	integer	Default allocation in blocks
RDB\$DEFAULT_EXTENT	integer	Default extension in blocks
RDB\$CURRENT_BACKUP_SEQUENCE	integer	Backup sequence number
RDB\$CURR_RECOVERY_SEQUENCE	integer	Recovery sequence number
RDB\$DATABASE_BACKUP_SEQUENCE	integer	Database backup sequence number
RDB\$ALLOCATION	integer	Number of allocated AIJ file blocks
RDB\$SHUTDOWN_TIME_MIN	integer	Shutdown time in minutes
RDB\$OPERATOR_CLASSES	integer	Operator class
RDB\$FLAGS	integer	Flags
RDB\$DEFAULT_BACKUP_FILENAME	char(255)	Default backup file name
RDB\$CACHE_FILENAME	char(255)	Cache file name
RDB\$STANDBY_FILENAME	char(255)	Standby database file name
RDB\$SERVER_NAME	char(31)	Server name
RDB\$BACKUP_EDIT_STRING	varchar(255)	Backup editname file
RDB\$REMOTE_NODE_NAME	char(31)	Remote node name
RDB\$CUR_ACTIVE_AIJ	integer	Current active AIJ journal index
RDB\$MASTER_FILENAME	char(255)	When database replication is active on the standby database, this field contains the file name of the master database
RDB\$LSS_NETTYPE	integer	Network transport (DECnet, TCP/IP)
RDB\$LSS_SYNCED_TAD	date vms	LSS Replication synchronized date/time

I.2.1.5.1 RDB\$FLAGS Represents flags for RDB\$DATABASE_JOURNAL information table.

Bit Position	Description
0	Journaling is enabled

Bit Position	Description
1	Overwrite is enabled
2	Backup mode
3	New journal version
4	ABS uses quiet-point AIJ backup
5	Replicated as master
6	Replicated as standby
7	Master replication database
8	Database replication online
9	Hot Standby quiet-point
10	Hot Standby is enabled
11	Database changes made when AIJ disabled
12	One or more journals overwritten
13	Hard data loss resulted from fail over
14	Full quiet-point AIJ backup required

I.2.1.6 RDB\$DATABASE_USERS

Displays information about the database users.

Column Name	Data Type	Description
RDB\$PROCESS_ID	integer	Process ID number
RDB\$STREAM_ID	integer	Stream ID number
RDB\$MONITOR_ID	integer	Monitor ID number
RDB\$ATTACH_ID	integer	Attach ID number
RDB\$FLAGS	integer	Flags

I.2.1.6.1 RDB\$FLAGS Represents flags for RDB\$DATABASE_USERS information table.

Bit Position	Description
0	Client server process
1	AIJ log server
2	Process is being recovered

Bit Position	Description
3	Database server process
4	Database utility process
5	Catch-up server
6	AIJ roll forward server
7	Row cache server
8	Log shipping server
9	Backup server
10	Continuous LogMiner Server
11	This process is binding to the database

I.2.1.7 RDB\$LOGICAL_AREAS

Displays information about the logical areas.

Column Name	Data Type	Description
RDB\$LOGICAL_AREA_ID	integer	Logical area ID
RDB\$AREA_ID	integer	Physical area ID
RDB\$FLAGS	integer	Flags
RDB\$RECORD_LENGTH	integer	Record length
RDB\$THRESHOLD1_PERCENT	integer	First threshold percent value
RDB\$THRESHOLD2_PERCENT	integer	Second threshold percent value
RDB\$THRESHOLD3_PERCENT	integer	Third threshold percent value
RDB\$ORDERED_HASH_OFFSET	integer	Ordered hash offset
RDB\$RECORD_TYPE	integer	AIP record type
RDB\$LOGICAL_AREA_NAME	char(31)	Logical area name

I.2.1.7.1 RDB\$FLAGS Represents flags for RDB\$LOGICAL_AREAS information table.

Bit Position	Description
0	Logical area uses hash ordered index
1	Logical area modified with unjournaled records

Bit Position	Description
2	Nologging is enabled
3	Use full quadword modulo for hashing (otherwise use longword modulo)
4	SPAM thresholds should be rebuilt

I.2.1.7.2 RDB\$RECORD_TYPE Represents AIP record types for RDB\$LOGICAL_ AREAS information table.

AIP Record Types	Description
0	Unknown
1	Table
2	Sorted index
3	Hashed index
4	System record
5	Large object

I.2.1.8 RDB\$CHARACTER_SETS

Displays information about the Oracle Rdb character sets.

Column Name	Data Type	Description
RDB\$LOWCASE_MAPPING	list of byte varying	Segmented string containing a 256 byte table used for lowercasing characters
RDB\$UPCASE_MAPPING	list of byte varying	Segmented string containing a 256 byte table used for uppercasing characters
RDB\$ASSOCIATED_CHARACTER_SET	integer	Identifier of the associated character set
RDB\$CHARACTER_SET_ID	integer	Character set identifier
RDB\$CHARACTER_SET_NAME	char(31)	Character set name
RDB\$CHARACTER_WILDCARD	integer	Character used as wildcard character
RDB\$FLAGS	integer	Character set flags
RDB\$FORM_OF_USE	integer	Character set form-of-use indicator

Column Name	Data Type	Description
RDB\$IDENTIFIER_CHARACTER_SET	integer	Character set ID of the identifier character set
RDB\$MAXIMUM_OCTETS	integer	Maximum number of octets per character
RDB\$MINIMUM_OCTETS	integer	Minimum number of octets per character
RDB\$REPERTOIRE	integer	Character set repertoire
RDB\$SPACE_CHARACTER	integer	Character used as space
RDB\$STRING_WILDCARD	integer	Character used as string wildcard
RDB\$VERSION	integer	Version number of character set entry

I.2.1.8.1 RDB\$REPERTOIRE Represents the repertoire values for RDB\$CHARACTER_SETS information table.

Value	Name	Description
0	OTHER	Non-specific repertoire
1	LATIN	Contains mainly Latin characters
2	JAPANESE	Contains mainly Japanese characters
3	SIMPLE_CHINESE	Contains mainly simplified Chinese characters
4	KOREAN	Contains mainly Korean characters
5	OLD_CHINESE	Contains mainly traditional Chinese characters
6	UNIVERSAL	Contains universal characters; for example, UNICODE
7	INDIAN	Contains mainly Indian characters
8	ARABIC	Contains mainly Arabic characters
9	GREEK	Contains mainly Greek characters
10	CYRILLIC	Contains mainly Cyrillic characters
11	HEBREW	Contains mainly Hebrew characters

I.2.1.8.2 RDB\$FORM_OF_USE Represents the form-of-use values for RDB\$CHARACTER_SETS information table.

Note: MIXED are odd flags, FIXED are even.

Value	Name	Description
0	FIXED_OCTET	Fixed octet
1	MIXED_OCTET	Mixed octet with DEC_KANJI style encoding
2	FIXED_NO_UP	Fixed octet, no uppercasing allowed
3	MIXED_SS2	As in MIXED_OCTET plus <SS2> as introducer to alternate single octet encoding
4	FIXED_UP_G1	Fixed octet, uppercasing only 7-bit characters
5	MIXED_SS3	As in MIXED_SS2 plus <SS3> as introducer to alternate double octet encoding
6	FIXED_OTHER	Fixed octet, other
7	MIXED_C2CB	As in MIXED_OCTET plus hex 'C2CB' as introducer to alternate double octet encoding
9	MIXED_TAG	Mixed octet with leading tag, compound string
11	MIXED_SHIFT	Mixed octet with coding table shifted; for example, SHIFT_JIS
13	MIXED_110	Mixed octet with binary pattern of high order bits specifying char size
15	MIXED_GB18030	Mixed octet with 2nd octet in range HEX(30) - HEX(39) specifying 4 octet

I.2.1.8.3 RDB\$FLAGS Represents flags for RDB\$CHARACTER_SETS information table.

Bit Position	Name	Description
0	CONTAINS_ASCII	Character set contains 7-bit ASCII characters

Bit Position	Name	Description
1	SPACE_OCTET_REPEATS	All octets of the multi-octet space character are the same value

I.2.1.9 RDB\$NLS_CHARACTER_SETS

Represents the mapping of Oracle NLS character sets to Oracle Rdb character sets.

Column Name	Data Type	Description
RDB\$CHARACTER_SET_ID	integer	Character set identifier
RDB\$NLS_ID	integer	Oracle NLS identifier of character set
RDB\$NLS_NAME	char(31)	Oracle NLS character set name

I.2.1.10 RDB\$SESSION_PRIVILEGES

Represents the mapping of the enabled privileges for this user session.

Column Name	Description
RDB\$PRIVILEGE	String describing an assigned privilege.

Index

A

ACOS function, G-2
ACOSH function, G-2
Ada language
 declaring the SQLDA, D-6
 SQLCA, C-24
ADD_MONTHS function, G-2
ANSI_AUTHORIZATION qualifier
 See also RIGHTS clause in Volumes 1 and 2
 replaced by RIGHTS clause, F-6
ANSI_DATE qualifier
 See also DEFAULT DATE FORMAT clause in
 Volumes 1 and 2
 replaced by DEFAULT DATE FORMAT clause,
 F-6
ANSI_IDENTIFIERS qualifier
 See also KEYWORD RULES clause in
 Volumes 1 and 2
 replaced by KEYWORD RULES clause, F-6
ANSI_PARAMETERS qualifier
 See also PARAMETER COLONS clause in
 Volume 2
 replaced by PARAMETER COLONS clause,
 F-6
ANSI_QUOTING qualifier
 See also QUOTING RULES clause in Volumes
 1 and 2
 replaced by QUOTING RULES clause, F-6
ASCII function, G-3
ASCII in C programs
 restriction, D-13

ASCII in dynamic SQL
 restriction, D-13
ASCIZ in C programs
 restriction, D-13
ASCIZ in dynamic SQL
 restriction, D-13
ASIN function, G-3
ASINH function, G-3
ATAN2 function, G-5
ATAN function, G-4
ATANH function, G-4

B

BASIC language
 declaring the SQLDA, D-7, D-17
 SQLCA, C-24
BITAND function, G-5
BITANDNOT function, G-6
BITNOT function, G-7
BITOR function, G-7
BITXOR function, G-7
Built-in function, G-1

C

CALL statement
 dynamic SQL and
 determining, C-12
Cascading delete, F-3
CEIL function, G-7
Character set
 logical name
 RDB\$CHARACTER_SET, E-1
 specifying, E-1

- CHR function, G-7
- C language
 - declaring the SQLDA, D-8, D-18
 - declaring the SQLDA2, D-18
 - SQLCA, C-26
- COBOL language
 - SQLCA, C-27
 - using error literals, C-10
- CONTAINING predicate
 - returning data types for parameter markers, D-4
- Conversion
 - of data types
 - in dynamic SQL, D-14
- COS function, G-7
- COSH function, G-7
- COT function, G-8

D

- Database system tables, H-1
- Data type
 - conversion
 - in dynamic SQL, D-14
 - determining for dynamic SQL, D-12
- Declaring the SQLDA
 - in Ada, D-6
 - in BASIC, D-7, D-17
 - in PL/I, D-8
- Declaring the SQLDA2
 - in C, D-18
- DELETE statement
 - number of rows deleted, C-12
- Deprecated feature
 - of command line qualifiers, F-6
 - of constraint in CREATE TABLE statement, F-7
 - of ORDER BY clause, F-3
 - SQLOPTIONS=ANSI_AUTHORIZATION, F-6
 - SQLOPTIONS=ANSI_DATE, F-6
 - SQLOPTIONS=ANSI_IDENTIFIERS, F-6
 - SQLOPTIONS=ANSI_PARAMETERS, F-6
 - SQLOPTIONS=ANSI_QUOTING, F-6
 - UNIQUE predicate, F-8

- DESCRIBE statement
 - MARKERS clause, D-2
 - SELECT LIST clause, D-2
 - SQLDA, D-9
- Dynamic SQL
 - and date-time data types, D-15
 - CALL statement
 - determining if, C-12
 - data type conversion by setting SQLTYPE field, D-14
 - declaring the SQLDA
 - for Ada, D-6
 - for BASIC, D-7, D-17
 - for C, D-8
 - for PL/I, D-8
 - declaring the SQLDA2
 - for C, D-18
 - declaring the SQLDA2 for Ada, D-16
 - declaring the SQLDA2 for BASIC, D-17
 - description of SQLDA2 fields, D-19
 - description of SQLDA fields, D-9
 - determining data types, D-12
 - distinguishing SELECT from other statements, D-4
 - EXECUTE statement, D-3
 - FETCH statement, D-3
 - INCLUDE statement, D-3
 - multiple
 - SQLDA declarations, D-3
 - OPEN statement, D-2
 - parameter markers, D-2
 - purpose of SQLDA, D-1
 - select lists, D-1
 - SELECT statement
 - determining if, C-12
 - SQLDA, D-1, D-3
 - SQLDERRD array
 - and SELECT, C-13
 - SQLERRD array, C-12
 - SQLTYPE field, D-12
 - structure of SQLDA, D-5

E

Error handling

- error messages, A-1
- flagging, A-7
- online message documentation, A-1
- RDB\$LU_STATUS, C-22
- return codes in SQLCA, C-3
- sql_get_error_text routine, C-23
- sql_signal routine, C-23
- with message vector, C-1
- with SQLCA, C-1
- with SQLSTATE, C-30

Error literals

- COBOL, C-10

Error message

- flagging of precompiler and module language, A-7
- format of, A-1
- locations of online documentation, A-3
- online documentation locations, A-3
- types of, A-1

EXECUTE statement

- parameter markers, D-3
- SQLDA, D-3, D-9

EXP function, G-8

F

FETCH statement

- current row, C-12
- SQLERRD field and, C-12
- using select lists, D-3
- using SQLDA, D-3, D-9

FLOOR function, G-8

FORTTRAN language

- SQLCA, C-27

Function

- ACOS, G-2
- ACOSH, G-2
- ADD_MONTHS, G-2
- ASCII, G-3
- ASIN, G-3
- ASINH, G-3
- ATAN, G-4

Function (cont'd)

- ATAN2, G-5
- ATANH, G-4
- BITAND, G-5
- BITANDNOT, G-6
- BITNOT, G-7
- BITOR, G-7
- BITXOR, G-7
- built-in, G-1
- CEIL, G-7
- CHR, G-7
- COS, G-7
- COSH, G-7
- COT, G-8
- EXP, G-8
- external
 - logical name for location, E-1
- FLOOR, G-8
- HEXTORAW, G-8
- INITCAP, G-8
- INSTR, G-9
- INSTRB, G-9
- LAST_DAY, G-9
- LN, G-9
- LOG, G-10
- LPAD, G-10
- LTRIM, G-10
- MOD, G-10
- MONTHS_BETWEEN, G-10
- NEW_TIME, G-11
- NEXT_DAY, G-11
- Oracle, G-1
- POWER, G-12
- RAWTOHEX, G-12
- REPLACE, G-12
- RPAD, G-13
- RTRIM, G-13
- SIN, G-13
- SINH, G-13
- SQRT, G-14
- SUBSTR, G-14
- SUBSTRB, G-14
- TAN, G-14
- TANH, G-14

H

Handling errors

- online message documentation, A-1
 - RDB\$LU_STATUS, C-22
 - sql_get_error_text routine, C-23
 - sql_signal routine, C-23
 - with message vector, C-1
 - with SQLCA, C-1
 - with SQLSTATE, C-30
- HEXTORAW function, G-8

I

INCLUDE statement

- SQLDA, D-3, D-6, D-8
 - SQLDA2, D-18
- Incompatible syntax changes, F-1
- Information Tables, I-1
- INITCAP function, G-8
- INSERT statement
- number of rows stored, C-12
- INSTRB function, G-9
- INSTR function, G-9

L

LAST_DAY function, G-9

LIKE predicate

- returning data types for parameter markers, D-4

Limits and parameters

- maximum length of SQLNAME field, D-9

List

- length of longest element, C-12
- number of elements, C-12

LN function, G-9

LOG function, G-10

Logical name, E-1

- RDB\$CHARACTER_SET, E-1
- RDB\$ROUTINES, E-1
- RDMS\$BIND_OUTLINE_MODE, E-1
- RDMS\$BIND_QG_CPU_TIMEOUT, E-1
- RDMS\$BIND_QG_REC_LIMIT, E-1
- RDMS\$BIND_QG_TIMEOUT, E-1

Logical name (cont'd)

- RDMS\$BIND_SEGMENTED_STRING_BUFFER, E-1
 - RDMS\$DEBUG_FLAGS, E-1
 - RDMS\$DIAG_FLAGS, E-2
 - RDMS\$RTX_SHRMEM_PAGE_CNT, E-2
 - RDMS\$SET_FLAGS, E-2
 - RDMS\$USE_OLD_CONCURRENCY, E-2
 - RDMS\$USE_OLD_SEGMENTED_STRING, E-2
 - RDMS\$VALIDATE_ROUTINE, E-2
 - SQL\$DATABASE, E-2
 - SQL\$DISABLE_CONTEXT, E-2
 - SQL\$EDIT, E-2
 - SQLINI, E-2
 - SYSS\$CURRENCY, E-2
 - SYSS\$DIGIT_SEP, E-2
 - SYSS\$LANGUAGE, E-2
 - SYSS\$RADIX_POINT, E-2
- LPAD function, G-10
- LTRIM function, G-10

M

MARKERS clause of DESCRIBE statement, D-2

Messages, A-1

Message vector, C-1

- in Ada, C-24
- in BASIC, C-24
- in C, C-26
- in COBOL, C-27
- in FORTRAN, C-27
- in INCLUDE statement, C-1
- in Pascal, C-28
- in PL/I, C-29
- RDB\$LU_STATUS, C-22
- sql_get_error_text routine, C-23
- sql_signal routine, C-23

Metadata

- system tables, H-1

MOD function, G-10

MONTHS_BETWEEN function, G-10

Multiple SQLDA declarations, D-3

N

NEW_TIME function, G-11
NEXT_DAY function, G-11

O

Obsolete SQL syntax, F-1
OPEN statement
 parameter markers, D-2
 SQLERRD field and, C-12
 using SQLDA, D-2, D-9
Oracle Database functions, G-1

P

Parameter
 message vector, C-22
 related to SQLDA, D-14
 SQLCA, C-2
Parameter markers
 data types returned, D-4
 determining data types, D-12
 in DESCRIBE statement, D-2
 in EXECUTE statement, D-3
 in OPEN statement, D-2
 in SELECT statement, D-2
 in SQLDA, D-2
Pascal language
 SQLCA, C-28
PL/I language
 declaring the SQLDA, D-8
 SQLCA, C-29
 SQLDA, D-1
POWER function, G-12
Predicate
 UNIQUE, F-8
PREPARE statement
 SELECT LIST clause, D-2
 SQLDA, D-9
Previously reserved words
 SQL3, F-14

Q

Query cost estimate
 SQLCA values, C-12

R

RAWTOHEX function, G-12
RDB\$CHARACTER_SET logical name, E-1
RDB\$LU_STATUS field of message vector, C-22
RDB\$MESSAGE_VECTOR structure, C-22
 in Ada, C-24
 in BASIC, C-24
 in C, C-26
 in COBOL, C-27
 in FORTRAN, C-27
 in INCLUDE statement, C-1
 in Pascal, C-28
 in PL/I, C-29
 RDB\$LU_STATUS field, C-22
 sql_get_error_text routine, C-23
 sql_signal routine, C-23
RDB\$ROUTINES logical name, E-1
RDM\$BIND_OUTLINE_MODE logical name,
 E-1
RDM\$BIND_QG_CPU_TIMEOUT logical name,
 E-1
RDM\$BIND_QG_REC_LIMIT logical name,
 E-1
RDM\$BIND_QG_TIMEOUT logical name, E-1
RDM\$BIND_SEGMENTED_STRING_BUFFER
 logical name, E-1
RDM\$DEBUG_FLAGS logical name, E-1
RDM\$DIAG_FLAGS logical name, E-2
RDM\$RTX_SHRMEM_PAGE_CNT logical
 name, E-2
RDM\$SET_FLAGS logical name, E-2
RDM\$USE_OLD_CONCURRENCY logical
 name, E-2
RDM\$USE_OLD_SEGMENTED_STRING
 logical name, E-2

RDMSS\$VALIDATE_ROUTINE logical name, E-2

REPLACE function, G-12

REPLACE statement
number of rows stored, C-13

Reserved word
ANSI89, F-10
SQL92 Standard, F-11
SQL:1999, F-13

Restriction
ASCII in C programs, D-13
ASCII in dynamic SQL, D-13
ASCIZ in C programs, D-13
ASCIZ in dynamic SQL, D-13

Routine
sql_get_error_text, C-23
sql_signal, C-23

RPAD function, G-13

RTRIM function, G-13

S

SELECT LIST clause
of DESCRIBE statement, D-2
of PREPARE statement, D-2

Select lists
DESCRIBE statement, D-2
determining data types, D-12
for SELECT statements, D-3
in dynamic SQL, D-1
PREPARE statement, D-2
used by FETCH statements, D-3

SELECT statement
dynamic SQL and
determining, C-12
number of rows in result table, C-13
parameter markers, D-2
select lists, D-3

SIN function, G-13

SINH function, G-13

SQL\$DATABASE logical name, E-2

SQL\$DISABLE_CONTEXT logical name, E-2

SQL\$EDIT logical name, E-2

SQL\$GET_ERROR_TEXT routine

See also sql_get_error_text routine, C-23

SQL\$SIGNAL routine
See sql_signal routine

SQL3 draft standard
previously reserved words, F-14

SQLABC field of SQLCA, C-3

SQLAID field of SQLCA, C-3

SQLCA, C-1
and string truncation, C-7
declaring explicitly, C-2
description of fields, C-2
error return codes, C-3
in Ada, C-24
in BASIC, C-24
in C, C-26
in COBOL, C-27
in FORTRAN, C-27
in INCLUDE statement, C-1
in Pascal, C-28
in PL/I, C-29

list information in SQLERRD array, C-12
query cost estimates in SQLERRD array, C-12

SQLABC field, C-3

SQLAID field, C-3

SQLCODE field, C-2, C-3

SQLERRD array, C-11
and counts, C-12, C-13
and dynamic SELECT, C-13
and OPEN list cursor, C-12
and OPEN table cursor, C-12
dynamic SQL and, C-12

SQLERRD field, C-2

SQLWARN fields, C-13

SQLCHRONO_SCALE field of SQLDA2
codes for date-time data types, D-26

SQLCODE field, C-3
declaring explicitly, C-2
error status code, C-3
value of return code, C-3

SQLDA, D-1
data types returned for parameter markers, D-4
declared by INCLUDE, D-3

SQLDA (cont'd)

- declaring for
 - Ada, D-6
 - BASIC, D-7, D-17
 - C, D-8
 - PL/I, D-8
- description of fields, D-9
- for date-time data types
 - See* SQLDA2
- in DESCRIBE statement, D-9
- in EXECUTE statement, D-3, D-9
- in FETCH statement, D-3, D-9
- information about select lists, D-1
- in OPEN statement, D-2, D-9
- in PREPARE statement, D-9
- in programs, D-3
- parameter markers, D-2
- purpose, D-1
- related parameters, D-14
- related SQLDAPTR declaration, D-14
- related SQLSIZE declaration, D-14
- setting SQLTYPE field to convert data types, D-14
- SQLDABC field, D-9
- SQLDAID field, D-9
- SQLDAPTR parameter, D-14
- SQLDATA field, D-9
- SQLD field, D-9
- SQLIND field, D-9
- SQLLEN field, D-9
- SQLNAME field, D-9
- SQLSIZE parameter, D-14
- SQLTYPE field, D-9
- SQLVAR field, D-9
- structure, D-5
- using multiple, D-3

SQLDA2, D-15

- codes for date-time data types, D-26
- codes for interval data types, D-25
- declaring for
 - C, D-18
- description of fields, D-19

SQLDABC field of SQLDA, D-9

SQLDAID field of SQLDA, D-9

SQLDAPTR parameter, D-14

SQLDATA field

- allocating dynamic memory for, D-19

SQLDATA field of SQLDA, D-9

SQLD field of SQLDA, D-9

SQLERRD array of SQLCA, C-11

- dynamic SELECT and, C-13
- list information, C-12
- query cost estimates, C-12

SQLIND field of SQLDA, D-9

SQLINI command file

- logical name, E-2

SQLLEN field

- of SQLDA, D-9
- of SQLDA2
 - codes for interval data types, D-25
- use in SQLDA contrasted with use in SQLDA2, D-19

SQL module processor

- command line qualifiers, F-6

SQLN

- SQLDABC field, D-9

SQLNAME field of SQLDA, D-9

SQLN field of SQLDA, D-9

SQL precompiler

- sql_get_error_text routine, C-23
- sql_signal routine, C-23

SQLSIZE parameter, D-14

SQLSTATE, C-30

SQLTYPE field of SQLDA, D-9, D-12

- setting to convert data types, D-14

SQLVAR field of SQLDA, D-9

SQLWARN fields of SQLCA, C-13

sql_get_error_text routine, C-23

sql_signal routine, C-23

SQRT function, G-14

Standards, B-1

STARTING WITH predicate

- returning data types for parameter markers, D-4

String truncation

- and SQLCA, C-7

SUBSTRB function, G-14
SUBSTR function, G-14
Syntax
 incompatible changes, F-1
SYSSCURRENCY logical name, E-2
SYSSDIGIT_SEP logical name, E-2
SYSSLANGUAGE logical name, E-2
SYSSRADIX_POINT logical name, E-2
System table, H-1
 detailed, H-1

T

Tables
 system, H-1
TAN function, G-14

TANH function, G-14
Truncating
 strings, C-7

U

UNIQUE predicate, F-8
UPDATE statement
 number of rows modified, C-13

V

Variable
 SQLDA, D-1
 SQLDA2, D-15