ODBC and SQL Reference

June, 1999

This manual details ODBC conformance and SQL language support provided by the Dharma DataLink SDK. It also describes configuration of the ODBC SDK Drivers.

Software Version: Dharma DataLink SDK Version 6.2.1
June, 1999

This draft printed: 09' 30' 99 9:31 AM

Information in this document is subject to change without notice.

Dharma Systems Inc. shall not be liable for any incidental, direct, special or consequential damages whatsoever arising out of or relating to this material, even if Dharma Systems Inc. has been advised, knew or should have known of the possibility of such damages.

The software described in this manual is furnished under a license agreement or nondisclosure agreement. The software may be used or copied only in accordance with the terms of this agreement. It is against the law to copy this software on magnetic tape, disk or any other medium for any purpose other than for backup or archival purposes.

This manual contains information protected by copyright. No part of this manual may be photocopied or reproduced in any form without prior written consent from Dharma Systems Inc.

Use, duplication, or disclosure whatsoever by the Government shall be expressly subject to restrictions as set forth in subdivision (b)(3)(ii) for restricted rights in computer software and subdivision (b)(2) for limited rights in technical data, both as set in 52.227-7013.

© 1987-1999 Dharma Systems, Inc. All rights reserved.

Dharma Systems welcomes your comments on this document and the software it describes. Send comments to:

Documentation Comments
Dharma Systems, Inc.
436 Amherst St.
Nashua, NH 03063
Phone: 603-886-1400
Fax: 603-883-6904
Electronic Mail: support@dharma.com
Web Page: http://www.dharma.com

DBstore, Dharma/ SQL, Dharma/ ODBC, Dharma ODBC SDK, Dharma DataLink, and Dharma Integrator are trademarks of Dharma Systems, Inc.

The following are third-party trademarks:

Microsoft and MS-DOS are registered trademarks, and ODBC, Windows, Windows 95 and Windows NT are trademarks of Microsoft Corporation.

Oracle is a registered trademark of Oracle Corporation.

Java, Java Development Kit, Solaris, SPARC, SunOS, and SunSoft are trademarks of Sun Microsystems, Inc.

PowerBuilder is a registered trademark of Powersoft Corporation.

All other trademarks and registered trademarks are the property of their respective holders.
# Contents

Preface ................................................................................................................................... viii

1 Overview

1.1 Introduction .......................................................................................................................... 1–1
1.2 Desktop Configuration .......................................................................................................... 1–2
1.3 Client/Server Configuration .................................................................................................. 1–3

2 Configuring ODBC Data Sources

2.1 Confirming Installation ......................................................................................................... 2–1
2.2 Adding Names for ODBC Data Sources .............................................................................. 2–2
   2.2.1 Desktop ......................................................................................................................... 2–2
   2.2.2 Client/Server .................................................................................................................. 2–3

3 ODBC Driver Conformance

3.1 General ODBC and SQL Support Levels .......................................................................... 3–1
3.2 Responses to SQLGetInfo .................................................................................................. 3–1
3.3 Supported Environment, Connection, and Statement Attributes ...................................... 3–17
3.4 Supported ODBC API Functions ....................................................................................... 3–20
3.5 Supported Data Types ......................................................................................................... 3–22

4 SQL Language Elements

4.1 SQL Features: Lite and Professional Editions ..................................................................... 4–2
4.2 SQL Identifiers ...................................................................................................................... 4–3
   4.2.1 Conventional Identifiers ............................................................................................... 4–3
   4.2.2 Delimited Identifiers ..................................................................................................... 4–4
4.3 Data Types............................................................................................................... 4–4
  4.3.1 Character Data Types ......................................................................................... 4–5
  4.3.2 Exact Numeric Data Types ................................................................................ 4–9
  4.3.3 Approximate Numeric Data Types ................................................................. 4–11
  4.3.4 Date-Time Data Types ..................................................................................... 4–11
  4.3.5 Bit String Data Types ....................................................................................... 4–12

4.4 Query Expressions ................................................................................................. 4–13
  4.4.1 Inner Joins ....................................................................................................... 4–20
  4.4.2 Outer Joins ...................................................................................................... 4–23

4.5 Search Conditions .................................................................................................. 4–24
  4.5.1 Logical Operators: OR, AND, NOT ................................................................. 4–25
  4.5.2 Relational Operators ....................................................................................... 4–25
  4.5.3 Basic Predicate ................................................................................................ 4–26
  4.5.4 Quantified Predicate ....................................................................................... 4–27
  4.5.5 BETWEEN Predicate ....................................................................................... 4–27
  4.5.6 NULL Predicate ............................................................................................... 4–28
  4.5.7 CONTAINS Predicate ..................................................................................... 4–28
  4.5.8 LIKE Predicate ............................................................................................... 4–29
  4.5.9 EXISTS Predicate ......................................................................................... 4–29
  4.5.10 IN Predicate ................................................................................................. 4–30
  4.5.11 Outer Join Predicate ..................................................................................... 4–30

4.6 Expressions ............................................................................................................. 4–30
  4.6.1 Concatenated Character Expressions .......................................................... 4–32
  4.6.2 Numeric Arithmetic Expressions .................................................................... 4–33
  4.6.3 Date Arithmetic Expressions ......................................................................... 4–33
  4.6.4 Conditional Expressions ............................................................................... 4–35

4.7 Literals ..................................................................................................................... 4–35
  4.7.1 Numeric Literals ............................................................................................. 4–36
  4.7.2 Character String Literals ............................................................................... 4–36
  4.7.3 Date-Time Literals ......................................................................................... 4–37

4.8 Date-Time Format Strings .................................................................................... 4–40
  4.8.1 Date Format Strings ...................................................................................... 4–40
  4.8.2 Time Format Strings ..................................................................................... 4–42

4.9 Functions ............................................................................................................... 4–43
  4.9.1 Aggregate Functions ...................................................................................... 4–43
5 SQL Statements

5.1 CALL .......................................................................................................................... 5–2
5.2 CREATE INDEX ............................................................................................................ 5–3
5.3 CREATE SYNONYM .................................................................................................... 5–5
5.4 CREATE TABLE .......................................................................................................... 5–6
  5.4.1 Column Constraints .......................................................................................... 5–9
  5.4.2 Table Constraints ............................................................................................. 5–11
5.5 CREATE VIEW ............................................................................................................ 5–15
5.6 DELETE ...................................................................................................................... 5–17
5.7 DROP INDEX ............................................................................................................ 5–18
5.8 DROP SYNONYM .................................................................................................... 5–19
5.9 DROP TABLE ............................................................................................................ 5–20
5.10 DROP VIEW ............................................................................................................ 5–21
5.11 GRANT ..................................................................................................................... 5–21
5.12 INSERT ..................................................................................................................... 5–22
5.13 RENAME .................................................................................................................. 5–27
5.14 REVOKE ................................................................................................................... 5–28
5.15 SELECT .................................................................................................................... 5–31
  5.15.1 ORDER BY Clause ......................................................................................... 5–32
  5.15.2 FOR UPDATE Clause ..................................................................................... 5–33
5.16 SET SCHEMA ......................................................................................................... 5–34
5.17 UPDATE ................................................................................................................... 5–36

A Server Utility Reference

Overview ......................................................................................................................... A–1

dhdaemon ..................................................................................................................... A–1
pcreg .............................................................................................................................. A–3
This manual details ODBC conformance and SQL language support provided by the Dharma DataLink SDK. It also describes configuration of the ODBC SDK Drivers.

**Audience**

This manual is intended for a variety of audiences that use Dharma DataLink SDK drivers to access data in proprietary storage systems:

- Administrators who set up access to a proprietary storage system from existing ODBC applications
- Application programmers writing ODBC applications that will access proprietary data through the ODBC Driver

**Structure**

The manual contains the following chapters and appendixes:

Chapter 1  Introduces the ODBC SDK and its components.
Chapter 2  Describes configuring data sources to use ODBC SDK drivers.
Chapter 3  Details the ODBC Driver's support for the ODBC standard, including information it returns to the SQLGetInfo call, and which ODBC API calls it supports.
Chapter 4  Describes language elements common to many SQL statements.
Chapter 5  Provides detailed reference material on each SQL statement, in alphabetic order.
Appendix A  Provides reference material on utilities used in ODBC Server administration.
Appendix B  Lists reserved words.
Appendix C  Lists error messages.
Appendix D  Summarizes the tables in the data dictionary.
Appendix E  Lists the maximum sizes for various attributes of the ODBC Server environment.
Conventions

Syntax diagrams appear in helvetica type and use the following conventions:

UPPERCASE

Uppercase type denotes reserved words. You must include reserved words in statements, but they can be upper or lower case.

lowercase

Lowercase type denotes either user-supplied elements or names of other syntax diagrams. User-supplied elements include names of tables, host-language variables, expressions, and literals. Syntax diagrams can refer to each other by name. If a diagram is named, the name appears in lowercase type above and to the left of the diagram, followed by a double-colon (for example, privilege ::). The name of that diagram appears in lowercase in diagrams that refer to it.

{} Braces denote a choice among mandatory elements. They enclose a set of options, separated by vertical bars ( | ). You must choose at least one of the options.

[] Brackets denote an optional element or a choice among optional elements.

| Vertical bars separate a set of options.

... A horizontal ellipsis denotes that the preceding element can optionally be repeated any number of times.

() ; Parentheses and other punctuation marks are required elements. Enter them as shown in syntax diagrams.

For More information

Microsoft ODBC Programmer's Reference, Version 3.0 Describes the ODBC interface, its features, and how applications use it.
1

Overview

1.1 Introduction

The Dharma DataLink SDK allows users to access proprietary storage systems through the Open Database Connectivity (ODBC) interface.

The ODBC SDK supports two different configurations for different network environments.

- The ODBC SDK Desktop configuration implements a “single-tier” ODBC architecture where the ODBC tool, the Dharma DataLink Desktop Driver, and the proprietary data all reside on the same Windows 95, Windows 98, or NT computer.

- The ODBC SDK Client/Server configuration provides network access to your proprietary data. The ODBC tool and the Dharma DataLink driver run on Windows or UNIX clients, while the ODBC SDK server library runs on the UNIX or NT server hosting the proprietary storage system.

This manual describes how to add data sources for both configurations (Chapter 2), and provides detailed reference material about the ODBC and SQL support they offer (Chapters 3, 4, and 5).

What is ODBC?

The ODBC interface enables Windows applications to access data from a variety of data sources while insulating the applications from differences between them. The ODBC standard specifies two major components:

- A library of function calls that allows applications to connect with a database system and issue requests. Chapter 3 details Dharma DataLink support.

- Syntax for Structured Query Language (SQL) statements, based on existing standards. Chapters 4 and 5 provide detailed reference material about the SQL interface to the ODBC SDK.
1.2 Desktop Configuration

With Dharma DataLink SDK Desktop, all components reside on a single computer.

**Application**
An ODBC application can be any program that calls ODBC functions and uses them to issue SQL statements. Vendors of Windows-based tools typically include ODBC capabilities with their software. Tools such as PowerBuilder™, Visual Basic™, and Web tools all can use ODBC for data access.

**ODBC Driver Manager**
The ODBC driver manager is a Microsoft-supplied program that routes calls from an application to the appropriate **ODBC driver** for a data source. To an application, the ODBC driver manager and a driver are a single entity that process requests to a particular data source.

**Dharma DataLink Desktop Driver**
The Desktop Driver processes ODBC function calls from applications that need to access the proprietary storage system. The driver translates the standard SQL statements into syntax the data source can process, retrieves the requested data from the proprietary storage system, and returns data to the application.

The Dharma DataLink Desktop Driver runs on Windows 95, Windows 98, and Windows NT. It is compatible with Version 3.0 of the Microsoft ODBC standard. It supports all Core and Level 1 API functions, and all Level 2 functions required by Windows and Web tools.

Figure 1–1 shows how these components work together to provide access to proprietary data.

**Figure 1–1 Components in the Dharma Desktop ODBC SDK**
1.3 Client/Server Configuration

With Dharma DataLink SDK Client/Server, there are components on two separate computers. The **client** is the system where the ODBC application runs. The **server** is the system that contains the proprietary storage system.

With the client/server configuration, the functions performed by the ODBC Desktop Driver are separated into two components, the ODBC Driver and the ODBC Server:

- The ODBC Driver processes ODBC function calls from applications that request data from the proprietary storage system. The driver connects to the ODBC Server, translates the standard SQL statements into syntax the data source can process, and returns data to the application.

  The Dharma DataLink Driver runs on Windows clients. It is compatible with Version 3.0 of the Microsoft ODBC standard. It supports all Core and Level 1 API functions, and all Level 2 functions required by Windows and Web tools.

- The Dharma DataLink Server runs on the server hosting the proprietary storage system. It receives requests from the ODBC Driver, processes them, and retrieves the requested data from the proprietary storage system.

Figure 1–2 shows how these components work together to provide access to proprietary data.

**Figure 1–2 Dharma DataLink SDK Client/Server**

![Diagram of Dharma DataLink SDK Client/Server](image)
This chapter describes how to add ODBC data sources that provide access to a proprietary storage system.

2.1 Confirming Installation

This chapter assumes that the Dharma DataLink SDK drivers have already been installed in your environment.

Refer to information for your proprietary storage system for details on setting up the ODBC SDK in your environment. The following steps may be completed as part of a setup program, or you may have to perform them yourself:

**Desktop**
- Install the necessary executable files, scripts and initialization file for the ODBC Desktop executable.
- Create a data dictionary and, if necessary, load it with SQL metadata that describe tables and indexes in the proprietary storage system.

**Client/Server**
- On the server system, install components for the ODBC Server.
- On the server system, create the sqlnw service name and associate it with a port number.
- On the server system, start the dhdaemon server process.
- On the server system, create a data dictionary and, if necessary, load it with SQL metadata that describe tables and indexes in the proprietary storage system.
- On the client system, install the ODBC Driver
- On the client system, create the sqlnw service name and associate it with the same port number as specified on the server.
2.2 Adding Names for ODBC Data Sources

The ODBC Administrator is a Microsoft utility to configure ODBC data sources and drivers.

All the information an application needs to connect to a particular database is called a data source in ODBC terminology. This information includes the driver name and location, network address, network software, and database name. Depending on the version of Windows, the ODBC Administrator stores data source information in text files or in the system registry.

The ODBC Administrator lets you enter the connection details for accessing a Dharma/SQL database and associate it with a data source name that users refer to when they need to access the data through an ODBC application.

2.2.1 Desktop

Use the Microsoft ODBC Administrator utility to add names of specific data sources you want to access.

1. Invoke the Microsoft ODBC Administrator from Windows (by default, from the Control Panel program group). The Administrator’s Data Sources dialog box appears.

2. Click on the System DSN tab. A list of existing system data sources appears.

3. Click on the Add... button. The Add Data Source dialog box appears.

4. In the list box, double-click on the Dharma Desktop SDK driver. The Dharma ODBC Setup dialog box appears.

5. Enter information in the following text boxes:

   - **Data Source Name:** the name of the ODBC data source for use in ODBC connect calls and by the ODBC Administrator.
   - **Description:** an optional descriptive string.
   - **Host:** this field is ignored for the Desktop configuration.
   - **Database:** the database name. Use the same name here that was specified to the mdcreate utility (see page A-3) to create the data dictionary for the proprietary storage system.
   - **User ID:** the user name for the process.
   - **Password:** the password for the process.
   - **Data Dir:** the location of the data dictionary directory. Leave this field blank unless the mdcreate command used the -d argument. (If it did, specify the same value here as that used in the -d argument.)
You must supply the name of the data source. If you omit the database name, user name, or password, the driver prompts the ODBC application user for that information when it connects to the data source.

### 2.2.2 Client/Server

Use the Microsoft ODBC Administrator utility to add the names of any Dharma DataLink server data sources the ODBC Driver will connect to:

1. Invoke the Microsoft ODBC Administrator from Windows (by default, from the Control Panel program group). The Administrator’s **Data Sources** dialog box appears.

2. Click on the **System DSN** tab. A list of existing system data sources appears.

3. Click on the **Add...** button. The **Create New Data Source** dialog box appears.

4. In the **Installed ODBC Drivers** list box, double-click on the **Dharma ODBC SDK** driver. The **Dharma ODBC Setup** dialog box appears.

5. Enter information in the following text boxes:
   - **Data Source Name**: a local name for the Dharma DataLink server data source for use in ODBC connect calls and by the ODBC Administrator.
   - **Description**: an optional descriptive string
   - **Host**: the name of the system where the Dharma DataLink server data source resides
   - **Database**: the database for the process to connect to on the host system. Use the same name here that was specified to the `mdcreate` utility (see page B-3) to create the data dictionary for the proprietary storage system.
   - **User ID**: the user name for the process
   - **Password**: the password for the process
   - **Service**: the service name used by the server. Leave this field blank unless the `dhdaemon` server process was started using the command line and the command specified the `-s` argument (see page A-2). (If it did, specify the same value here as that used in the `-s` argument.)

You must supply the name of the data source. If you omit the host name, database name, user name, or password, the driver prompts the ODBC application user for that information when it connects to the data source.

The ODBC Administrator utility updates the ODBC Driver manager registry entry with the information supplied in the dialog box.
3

ODBC Driver Conformance

3.1 General ODBC and SQL Support Levels

This chapter details the ODBC functionality the Dharma DataLink SDK supports through the ODBC Driver. ODBC specifies general conformance levels in two areas:

- ODBC application programming interface (API). Dharma supports all Core and Level 1 ODBC API functions, and most Level 2 functions.
- ODBC SQL syntax. Dharma supports Extended SQL syntax.

This chapter details the information the Dharma DataLink SDK driver returns when applications call the following functions:

- SQLGetInfo returns various details about the driver and its data source
- SQLGetEnvAttr, SQLGetConnectAttr, SQLGetStmtAttr return attributes of driver behavior
- SQLGetFunctions returns ODBC functions the driver supports
- SQLGetTypeInfo returns data types the driver supports

3.2 Responses to SQLGetInfo

Applications call the SQLGetInfo function to retrieve details about support a specific driver and data source provide for different ODBC functionality.

Applications supply the InfoType argument to SQLGetInfo to specify what information type they seek. SQLGetInfo returns the information to the InfoValuePtr output argument. Table 3-1 lists each InfoType argument the ODBC Driver recognizes along with the associated value returned to SQLGetInfo. Shaded rows indicate information types renamed or deprecated for ODBC 3.0.

For more detail about the SQLGetInfo function see the Microsoft ODBC Programmer’s Reference.
### Table 3-1 Information the ODBC Driver Returns to SQLGetInfo

<table>
<thead>
<tr>
<th>Description</th>
<th>InfoType Argument</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Guaranteed execute privileges on all procedures returned by SQLProcedures</td>
<td>SQL_ACCESSIBLE_PROCEDURES</td>
<td>Y (guaranteed)</td>
</tr>
<tr>
<td>Guaranteed read access to all table names returned by SQLTables</td>
<td>SQL_ACCESSIBLE_TABLES</td>
<td>N (not guaranteed)</td>
</tr>
<tr>
<td>Maximum number of active connections</td>
<td>SQL_ACTIVE_CONNECTIONS (Renamed for ODBC 3.0.)</td>
<td>See SQL_MAX_DRIVER_CONNECTIONS</td>
</tr>
<tr>
<td>Maximum number of active environments</td>
<td>SQL_ACTIVE_ENVIRONMENTS (ODBC 3.0)</td>
<td>0 (no limit)</td>
</tr>
<tr>
<td>Maximum number of active SQL statements</td>
<td>SQL_ACTIVE_STATEMENTS (Renamed for ODBC 3.0.)</td>
<td>See SQL_MAX_CONCURRENT_ACTIVITIES</td>
</tr>
<tr>
<td>Aggregate function support</td>
<td>SQL_AGGREGATE_FUNCTIONS (ODBC 3.0)</td>
<td>SQL_AF_ALL</td>
</tr>
<tr>
<td>Support for ALTER DOMAIN statement</td>
<td>SQL_ALTER_DOMAIN (ODBC 3.0)</td>
<td>0 (does not support)</td>
</tr>
<tr>
<td>Support for ALTER SCHEMA statement</td>
<td>SQL_ALTER_SCHEMA (ODBC 3.0)</td>
<td>0 (does not support)</td>
</tr>
<tr>
<td>Support for ALTER TABLE clauses</td>
<td>SQL_ALTER_TABLE</td>
<td>SQL_AT_ADD_COLUMN (supports adding columns)</td>
</tr>
<tr>
<td>Level of asynchronous mode support</td>
<td>SQL_ASYNC_MODE (ODBC 3.0)</td>
<td>SQL_AM_NONE (not supported)</td>
</tr>
<tr>
<td>Behavior with respect to the availability of row counts in batches</td>
<td>SQL_BATCH_ROW_COUNT (ODBC 3.0)</td>
<td>0 (does not support)</td>
</tr>
<tr>
<td>Support for batches</td>
<td>SQL_BATCH_SUPPORT (ODBC 3.0)</td>
<td>SQL_BS_ROW_COUNT_EXPLICIT (supports explicit batches with row-count generating statements)</td>
</tr>
<tr>
<td>Support for bookmarks</td>
<td>SQL_BOOKMARK_PERSISTENCE</td>
<td>0 (does not support)</td>
</tr>
<tr>
<td>Position of qualifier in a qualified table name</td>
<td>SQL_CATALOG_LOCATION (Renamed for ODBC 3.0. Was SQL_QUALIFIER_LOCATION)</td>
<td>SQL_QL_START (at the start of the name)</td>
</tr>
<tr>
<td>Support for catalog names</td>
<td>SQL_CATALOG_NAME (ODBC 3.0)</td>
<td>N (does not support)</td>
</tr>
<tr>
<td>Character used to separate table, column qualifiers</td>
<td>SQL_CATALOG_NAME_SEPARATOR (Renamed for ODBC 3.0. Was SQL_QUALIFIER_NAME_SEPARATOR)</td>
<td>. (period)</td>
</tr>
<tr>
<td>Name for a catalog</td>
<td>SQL_CATALOG_TERM (Renamed for ODBC 3.0. Was SQL_QUALIFIER_TERM)</td>
<td>&quot;&quot; (does not support)</td>
</tr>
<tr>
<td>Description</td>
<td>InfoType Argument</td>
<td>Returns</td>
</tr>
<tr>
<td>-----------------------------------------------------------------------------</td>
<td>---------------------------</td>
<td>----------------------------------------------</td>
</tr>
<tr>
<td>Statements that support catalog names</td>
<td>SQL_CATALOG_USAGE</td>
<td>0 (does not support)</td>
</tr>
<tr>
<td>(Renamed for ODBC 3.0. Was SQL_QUALIFIER_USAGE)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Default collation sequence name for the default character set</td>
<td>SQL_COLLATION_SEQ</td>
<td>&quot; &quot; (unknown)</td>
</tr>
<tr>
<td>(ODBC 3.0)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Support for column aliases</td>
<td>SQL_COLUMN_ALIAS</td>
<td>Y (supports)</td>
</tr>
<tr>
<td>Result of concatenation of NULL character column with non-NULL column</td>
<td>SQL_CONCAT_NULL_BEHAVIOR</td>
<td>SQL_CB_NULL (result is null)</td>
</tr>
<tr>
<td>Conversion from BIGINT</td>
<td>SQL_CONVERT_BIGINT</td>
<td>SQL_CVT_CHAR</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_CVT_NUMERIC</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_CVT_TINYINT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_CVT_SMALLINT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_CVT_INTEGER</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_CVT_FLOAT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_CVT_DOUBLE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_CVT_BIT</td>
</tr>
<tr>
<td>Conversion from BINARY</td>
<td>SQL_CONVERT_BINARY</td>
<td>SQL_CVT_CHAR</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_CVT_BINARY</td>
</tr>
<tr>
<td>Conversion from BIT</td>
<td>SQL_CONVERT_BIT</td>
<td>SQL_CVT_CHAR</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_CVT_NUMERIC</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_CVT_TINYINT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_CVT_SMALLINT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_CVT_BIGINT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_CVT_INTEGER</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_CVT_FLOAT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_CVT_DOUBLE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_CVT_REAL</td>
</tr>
<tr>
<td>Conversion from CHAR</td>
<td>SQL_CONVERT_CHAR</td>
<td>SQL_CVT_BIT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_CVT_BINARY</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_CVT_NUMERIC</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_CVT_TINYINT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_CVT_SMALLINT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_CVT_BIGINT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_CVT_INTEGER</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_CVT_DATE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_CVT_FLOAT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_CVT_TIME</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_CVT_TIMESTAMP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_CVT_CHAR</td>
</tr>
<tr>
<td>Conversion from DATE</td>
<td>SQL_CONVERT_DATE</td>
<td>SQL_CVT_CHAR</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_CVT_DATE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_CVT_TIMESTAMP</td>
</tr>
<tr>
<td>Conversion from DECIMAL</td>
<td>SQL_CONVERT_DECIMAL</td>
<td>0 (does not support)</td>
</tr>
<tr>
<td>Conversion from DOUBLE</td>
<td>SQL_CONVERT_DOUBLE</td>
<td>0 (does not support)</td>
</tr>
<tr>
<td>Description</td>
<td>InfoType Argument</td>
<td>Returns</td>
</tr>
<tr>
<td>-------------</td>
<td>------------------</td>
<td>---------</td>
</tr>
<tr>
<td>Conversion from FLOAT</td>
<td>SQL_CONVERT_FLOAT</td>
<td>SQL_CVT_CHAR</td>
</tr>
<tr>
<td>Support for conversion functions</td>
<td>SQL_CONVERT_FUNCTIONS</td>
<td>SQL_FN_CVT_CONVERT (supports)</td>
</tr>
<tr>
<td>Conversion from INTEGER</td>
<td>SQL_CONVERT_INTEGER</td>
<td>SQL_CVT_CHAR</td>
</tr>
<tr>
<td>Conversion from INTERVAL_DAY_TIME</td>
<td>SQL_CONVERT_INTERVAL_DAY_TIME (ODBC 3.0)</td>
<td>0 (does not support)</td>
</tr>
<tr>
<td>Conversion from INTERVAL_YEAR_MONTH</td>
<td>SQL_CONVERT_INTERVAL_YEAR_MONTH (ODBC 3.0)</td>
<td>0 (does not support)</td>
</tr>
<tr>
<td>Conversion from LONGVARBINARY</td>
<td>SQL_CONVERT_LONGVARBINARY</td>
<td>0 (does not support)</td>
</tr>
<tr>
<td>Conversion from LONGVARCHAR</td>
<td>SQL_CONVERT_LONGVARCHAR</td>
<td>0 (does not support)</td>
</tr>
<tr>
<td>Conversion from NUMERIC</td>
<td>SQL_CONVERT_NUMERIC</td>
<td>SQL_CVT_CHAR</td>
</tr>
<tr>
<td>Conversion from REAL</td>
<td>SQL_CONVERT_REAL</td>
<td>SQL_CVT_CHAR</td>
</tr>
<tr>
<td>Description</td>
<td>InfoType Argument</td>
<td>Returns</td>
</tr>
<tr>
<td>------------------------------</td>
<td>----------------------------</td>
<td>------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Conversion from SMALLINT</td>
<td>SQL_CONVERT_SMALLINT</td>
<td>SQL_CVT_CHAR</td>
</tr>
<tr>
<td>Conversion from TIME</td>
<td>SQL_CONVERT_TIME</td>
<td>SQL_CVT_CHAR</td>
</tr>
<tr>
<td>Conversion from TIMESTAMP</td>
<td>SQL_CONVERT_TIMESTAMP</td>
<td>SQL_CVT_CHAR</td>
</tr>
<tr>
<td>Conversion from TINYINT</td>
<td>SQL_CONVERT_TINYINT</td>
<td>SQL_CVT_CHAR</td>
</tr>
<tr>
<td>Conversion from VARBINARY</td>
<td>SQL_CONVERT_VARBINARY</td>
<td>0 (does not support)</td>
</tr>
<tr>
<td>Conversion from VARCHAR</td>
<td>SQL_CONVERT_VARCHAR</td>
<td>0 (does not support)</td>
</tr>
<tr>
<td>Support for table correlation names</td>
<td>SQL_CORRELATION_NAME</td>
<td>SQL_CN_DIFFERENT (supports, but correlation names must be different from table name)</td>
</tr>
<tr>
<td>Support for CREATE ASSERTION statement</td>
<td>SQL_CREATE_ASSERTION (ODBC 3.0)</td>
<td>0 (does not support)</td>
</tr>
<tr>
<td>Support for CREATE CHARACTER SET statement</td>
<td>SQL_CREATE_CHARACTER_SET (ODBC 3.0)</td>
<td>0 (does not support)</td>
</tr>
<tr>
<td>Support for CREATE COLLATION statement</td>
<td>SQL_CREATE_COLLATION (ODBC 3.0)</td>
<td>0 (does not support)</td>
</tr>
<tr>
<td>Support for CREATE DOMAIN statement</td>
<td>SQL_CREATE_DOMAIN (ODBC 3.0)</td>
<td>0 (does not support)</td>
</tr>
<tr>
<td>Support for CREATE SCHEMA statement</td>
<td>SQL_CREATE_SCHEMA (ODBC 3.0)</td>
<td>0 (does not support)</td>
</tr>
<tr>
<td>Support for CREATE TABLE statement</td>
<td>SQL_CREATE_TABLE (ODBC 3.0)</td>
<td>SQL_CT_CREATE_TABLE</td>
</tr>
<tr>
<td>Description</td>
<td>InfoType Argument</td>
<td>Returns</td>
</tr>
<tr>
<td>-------------</td>
<td>------------------</td>
<td>---------</td>
</tr>
<tr>
<td>Support for CREATE TRANSLATION statement</td>
<td>SQL_CREATE_TRANSLATION (ODBC 3.0)</td>
<td>0 (does not support)</td>
</tr>
<tr>
<td>Support for CREATE VIEW statement</td>
<td>SQL_CREATE_VIEW (ODBC 3.0)</td>
<td>SQL_CV_CREATE_VIEW</td>
</tr>
<tr>
<td>Effect of COMMIT operation on cursors and prepared statements</td>
<td>SQL_CURSOR_COMMIT_BEHAVIOR</td>
<td>SQL_CB_CLOSE (closes cursors but statements remain in prepared state)</td>
</tr>
<tr>
<td>Effect of ROLLBACK operation on cursors and prepared statements</td>
<td>SQL_CURSOR_ROLLBACK_BEHAVIOR</td>
<td>SQL_CB_CLOSE (closes cursors but statements remain in prepared state)</td>
</tr>
<tr>
<td>Support for cursor sensitivity</td>
<td>SQL_CURSOR_SENSITIVITY (ODBC 3.0)</td>
<td>SQL_INSENSITIVE (all cursors on the statement handle show the result set without reflecting any changes made to it by any other cursor within the same transaction)</td>
</tr>
<tr>
<td>Name of the data source as specified to the ODBC Administrator</td>
<td>SQL_DATA_SOURCE_NAME</td>
<td>(string containing the name)</td>
</tr>
<tr>
<td>Access limited to read-only</td>
<td>SQL_DATA_SOURCE_READ_ONLY</td>
<td>N (read-write access)</td>
</tr>
<tr>
<td>Name of the Dharma/SQL data source on the server system</td>
<td>SQL_DATABASE_NAME</td>
<td>(string containing the name)</td>
</tr>
<tr>
<td>Support for date-time literals</td>
<td>SQL_DATETIME_LITERALS (ODBC 3.0)</td>
<td>SQL_DL_SQL92_DATE</td>
</tr>
<tr>
<td>Name of the database product supporting the data source</td>
<td>SQL_DBMS_NAME</td>
<td>Dharma/SQL</td>
</tr>
<tr>
<td>Version of the database product</td>
<td>SQL_DBMS_VER</td>
<td>6.2.1</td>
</tr>
<tr>
<td>Support for creation and dropping of indexes</td>
<td>SQL_DDL_INDEX (ODBC 3.0)</td>
<td>SQL_DI_CREATE_INDEX</td>
</tr>
<tr>
<td>Default transaction isolation level</td>
<td>SQL_DEFAULT_TXN_ISOLATION</td>
<td>SQL_TXN_SERIALIZABLE</td>
</tr>
<tr>
<td>Support for describing parameters via DESCRIBE INPUT statement</td>
<td>SQL_DESCRIBE_PARAMETER (ODBC 3.0)</td>
<td>Y (supports)</td>
</tr>
<tr>
<td>Name of the dynamic link library file for the ODBC Driver</td>
<td>SQL_DRIVER_NAME</td>
<td>DHODBC.DLL</td>
</tr>
<tr>
<td>Supported ODBC version</td>
<td>SQL_DRIVER_ODBC_VER</td>
<td>03.00</td>
</tr>
<tr>
<td>Current Version of the ODBC Driver</td>
<td>SQL_DRIVER_VER</td>
<td>06.20.1000</td>
</tr>
<tr>
<td>Description</td>
<td>InfoType Argument</td>
<td>Returns</td>
</tr>
<tr>
<td>-------------------------------------------------</td>
<td>----------------------------------------</td>
<td>--------------------------------</td>
</tr>
<tr>
<td>Support for DROP ASSERTION statement</td>
<td>SQL_DROP_ASSERTION (ODBC 3.0)</td>
<td>0 (does not support)</td>
</tr>
<tr>
<td>Support for DROP CHARACTER SET statement</td>
<td>SQL_DROP_CHARACTER_SET (ODBC 3.0)</td>
<td>0 (does not support)</td>
</tr>
<tr>
<td>Support for DROP COLLATION statement</td>
<td>SQL_DROP_COLLATION (ODBC 3.0)</td>
<td>0 (does not support)</td>
</tr>
<tr>
<td>Support for DROP DOMAIN statement</td>
<td>SQL_DROP_DOMAIN (ODBC 3.0)</td>
<td>0 (does not support)</td>
</tr>
<tr>
<td>Support for DROP SCHEMA statement</td>
<td>SQL_DROP_SCHEMA (ODBC 3.0)</td>
<td>0 (does not support)</td>
</tr>
<tr>
<td>Support for DROP TABLE statement</td>
<td>SQL_DROP_TABLE (ODBC 3.0)</td>
<td>SQL_DT_DROP_TABLE (supports)</td>
</tr>
<tr>
<td>Support for DROP TRANSLATION statement</td>
<td>SQL_DROP_TRANSLATION (ODBC 3.0)</td>
<td>0 (does not support)</td>
</tr>
<tr>
<td>Support for DROP VIEW statement</td>
<td>SQL_DROP_VIEW (ODBC 3.0)</td>
<td>SQL_DT_DROP_VIEW (supports)</td>
</tr>
<tr>
<td>Supported attributes of a dynamic cursor: subset 1</td>
<td>SQL_DYNAMIC_CURSOR_ATTRIBUTES1 (ODBC 3.0)</td>
<td>0 (does not support dynamic cursors)</td>
</tr>
<tr>
<td>Supported attributes of a dynamic cursor: subset 2</td>
<td>SQL_DYNAMIC_CURSOR_ATTRIBUTES2 (ODBC 3.0)</td>
<td>0 (does not support dynamic cursors)</td>
</tr>
<tr>
<td>Support for expressions in ORDER BY clause</td>
<td>SQL_EXPRESSIONS_IN_ORDERBY</td>
<td>Y (supports)</td>
</tr>
<tr>
<td><strong>Direction that FETCH operations can support</strong></td>
<td>SQL_FETCH_DIRECTION (Deprecated in ODBC 3.0.)</td>
<td>SQL_FD_FETCH_NEXT (fetch next row only)</td>
</tr>
<tr>
<td>Single-tier driver behavior</td>
<td>SQL_FILE_USAGE</td>
<td>SQL_FILE_NOT_SUPPORTED (not a single-tier driver)</td>
</tr>
<tr>
<td>Supported attributes of a forward-only cursor: subset 1</td>
<td>SQL_FORWARDONLY_CURSOR_ATTRIBUTES1 (ODBC 3.0)</td>
<td>SQL_CA1_NEXT (supports SQL_FETCH_NEX argument to SQLFetchScroll for forward-only cursors)</td>
</tr>
<tr>
<td>Supported attributes of a forward-only cursor: subset 2</td>
<td>SQL_FORWARDONLY_CURSOR_ATTRIBUTES2 (ODBC 3.0)</td>
<td>SQL_CA2_READONLY_CONCURRENCY</td>
</tr>
<tr>
<td>Supported extensions to SQLGetData</td>
<td>SQL_GETDATA_EXTENSIONS</td>
<td>0 (does not support)</td>
</tr>
<tr>
<td>Relationship between GROUP BY clause and columns in the select list</td>
<td>SQL_GROUP_BY</td>
<td>SQL_GB_GROUP_BY_CONTAINS_SELECTION (GROUP BY clause must contain all non-aggregated columns in select list)</td>
</tr>
<tr>
<td>Case-sensitivity of user-supplied names</td>
<td>SQL_IDENTIFIER_CASE</td>
<td>SQL_IC_LOWER (case insensitive, stored in lower case)</td>
</tr>
<tr>
<td>Character used to enclose delimited identifiers</td>
<td>SQL_IDENTIFIERQUOTE_CHAR</td>
<td>&quot; (double quotation mark)</td>
</tr>
<tr>
<td>Supported views in INFORMATION_SCHEMA</td>
<td>SQL_INFO_SCHEMA_VIEWS (ODBC 3.0)</td>
<td>0 (does not support)</td>
</tr>
<tr>
<td>Description</td>
<td>InfoType Argument</td>
<td>Returns</td>
</tr>
<tr>
<td>-------------------------------------------------</td>
<td>------------------------------------</td>
<td>-----------------------------------------------------------</td>
</tr>
<tr>
<td>Support for INSERT statement</td>
<td>SQL_INSERT_STATEMENT</td>
<td>SQL_IS_INSERT_LITERALS</td>
</tr>
<tr>
<td>Referential integrity syntax support</td>
<td>SQL_INTEGRITY</td>
<td>Y (supports referential integrity syntax)</td>
</tr>
<tr>
<td>Supported attributes of a keyset cursor: subset 1</td>
<td>SQL_KEYSET_CURSOR_ATTRIBUTES1</td>
<td>0 (does not support keyset cursors)</td>
</tr>
<tr>
<td>Supported attributes of a keyset cursor: subset 2</td>
<td>SQL_KEYSET_CURSOR_ATTRIBUTES2</td>
<td>0 (does not support keyset cursors)</td>
</tr>
<tr>
<td>Data-source specific keywords</td>
<td>SQL_KEYWORDS</td>
<td>absolute, action, add, all, allocate, and, any, are, as, asc, assertion, at, authorization, alter, avg, atta, begin, between, bit_length, bigint, bitstring, binary, bit, both, by, cascade, cascaded, coll, case, cast, catalog, char, character, char_length, character_length, check, close, cluster, coalesce, collate, collation, column, colgroup_kw, colon, comma, comment, commit, compress, concat, connect, connection, constraint, constraint_kw, constraints, continue, convert, corresponding, count, create, cross, current, current_date, current_time, current_timestamp, current_user, cursor, database, datapages, date, day, deallocate, dec, decimal, declare, default, default_kw, deferrable, deferred, definition, delete, describe, delete_kw, desc, describe, descriptor, diagnostics, disconnect, distinct, div, domain, double, drop, dot, dh_f64_t, dba, end, else, end, end-exec, eqtok, escape, except, exception, exclusiv, exec, execute, exists, external, extract, false, fetch, file_kw, first, float, dh_f32_t, for, foreign, foreign_kw, fortran, found, from, full, e_tok, gt_tok, get, global, go, goto, grant, group, hash, having, hour, identified, identifier, identity, immediate, in, in_kw, include, index, indicator, inpages, initially, inner, input, insensitive, insert, int, integer, intersect, interval, into, is, isolation, join, key, key_kw, language, last, leading, left, level, le_tok, like, lin, lt_tok, local, lock, log, long_kw, lower, lvarchar, lvarchar, lpar, match, max, max_kw, min, min_kw, minu, minute, modify, mode, module, money, month, names, national, natural, nchar, ne_tok, next, no, nocompress, none, not, null, null_kw, nullif, number, numeric, nowait, octet_length, of, on,only, open, option, or, order, outer, outer_join, output, overlaps, pad, partial, pascal, pctfree, plus, position, precision, prepare, preserve, primary, primary_kw, prior, privileges, public, question, raw, read, real, realnumber, references, references_kw, relative, rename, resource, restrict, revoke, right, rollback, rows, rowid, rownum, rpar, schema, scroll, second, select, session, session_user, set, set_minus, share, size, smallint, some, space, sql, sqlca, sqlcode, sqlerror, sqlstate, sqlwarning, star, storage_manager, store_in_dharma, statistics, string, substring, sum, synonym, sysdate, system_user, systime, systimestamp, table, temporary, then, time, timestamp, timezone_hour, timezone_minute, timestamp, tinyint, to, trailing, transaction, trim, true, uid, union, unique, unknown, update, upper, user, using, value, values, varbinary, varchar, varying, view, when, whenever, where, with, work, write, year, zone, arcade, odbcce, odbcet, odbcone, content, odbces, odbcs, odbcset</td>
</tr>
<tr>
<td>Description</td>
<td>InfoType Argument</td>
<td>Returns</td>
</tr>
<tr>
<td>----------------------------------------------------------------------------</td>
<td>-----------------------------------------------------------------------------------</td>
<td>--------------------------------</td>
</tr>
<tr>
<td>Support for escape clause in LIKE predicates</td>
<td>SQL_LIKE_ESCAPE_CLAUSE</td>
<td>Y (supports)</td>
</tr>
<tr>
<td>Supported lock types</td>
<td>SQL_LOCK_TYPES (Deprecated in ODBC 3.0.)</td>
<td>0 (does not support)</td>
</tr>
<tr>
<td>Maximum number of active concurrent statements in asynchronous mode</td>
<td>SQL_MAX_ASYNC_CONCURRENT_STATEMENTS (ODBC 3.0)</td>
<td>0 (no limit)</td>
</tr>
<tr>
<td>Maximum length in hexadecimal characters of binary literals</td>
<td>SQL_MAX_BINARY_LITERAL_LEN (ODBC 3.0)</td>
<td>2000</td>
</tr>
<tr>
<td>Maximum length of a table or column qualifier</td>
<td>SQL_MAX_CATALOG_NAME_LEN (Renamed for ODBC 3.0. Was SQL_MAX_QUALIFIER_NAME_LEN)</td>
<td>0 (does not support)</td>
</tr>
<tr>
<td>Maximum length in characters of character string literals</td>
<td>SQL_MAX_CHAR_LITERAL_LEN</td>
<td>2000</td>
</tr>
<tr>
<td>Maximum length of a column name</td>
<td>SQL_MAX_COLUMN_NAME_LEN</td>
<td>32</td>
</tr>
<tr>
<td>Maximum number of columns allowed in GROUP BY clause</td>
<td>SQL_MAX_COLUMNS_IN_GROUP_BY</td>
<td>0 (no limit)</td>
</tr>
<tr>
<td>Maximum number of columns allowed in an index</td>
<td>SQL_MAX_COLUMNS_IN_INDEX</td>
<td>100</td>
</tr>
<tr>
<td>Maximum number of columns allowed in ORDER BY clause</td>
<td>SQL_MAX_COLUMNS_IN_ORDER_BY</td>
<td>0 (no limit)</td>
</tr>
<tr>
<td>Maximum number of columns allowed in a select list</td>
<td>SQL_MAX_COLUMNS_IN_SELECT</td>
<td>0 (no limit)</td>
</tr>
<tr>
<td>Maximum number of columns allowed in a table</td>
<td>SQL_MAX_COLUMNS_IN_TABLE</td>
<td>500</td>
</tr>
<tr>
<td>Maximum number of active SQL statements</td>
<td>SQL_MAX_CONCURRENT_ACTIVITIES (Renamed for ODBC 3.0. Was SQL_ACTIVE_STATEMENTS)</td>
<td>0 (no maximum)</td>
</tr>
<tr>
<td>Maximum length of a cursor name</td>
<td>SQL_MAX_CURSOR_NAME_LEN</td>
<td>32</td>
</tr>
<tr>
<td>Maximum number of active connections</td>
<td>SQL_MAX_DRIVER_CONNECTIONS (Renamed for ODBC 3.0. Was SQL_ACTIVE_CONNECTIONS)</td>
<td>10</td>
</tr>
<tr>
<td>Maximum length of user-defined names</td>
<td>SQL_MAX_IDENTIFIER_LEN (ODBC 3.0)</td>
<td>32</td>
</tr>
<tr>
<td>Maximum number of bytes allowed in the combined fields of an index</td>
<td>SQL_MAX_INDEX_SIZE</td>
<td>0 (no limit)</td>
</tr>
<tr>
<td>Maximum length of an owner name</td>
<td>SQL_MAX_OWNER_NAME_LEN (Renamed for ODBC 3.0.)</td>
<td>See SQL_MAX_SCHEMA_NAME_LEN</td>
</tr>
<tr>
<td>Maximum length of a procedure name</td>
<td>SQL_MAX_PROCEDURE_NAME_LEN</td>
<td>32</td>
</tr>
</tbody>
</table>
## Responses to SQLGetInfo

<table>
<thead>
<tr>
<th>Description</th>
<th>InfoType Argument</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum length of a table or column qualifier</td>
<td>SQL_MAX_QUALIFIER_NAME_LEN (Renamed for ODBC 3.0.)</td>
<td>See SQL_MAX_CATALOG_NAME_LEN</td>
</tr>
<tr>
<td>Maximum length in bytes of a table row</td>
<td>SQL_MAX_ROW_SIZE</td>
<td>0 (no limit)</td>
</tr>
<tr>
<td>Whether maximum row size includes LONGVARCHAR and LONGVARBINARY</td>
<td>SQL_MAX_ROW_SIZE_INCLUDES_LONG</td>
<td>Y</td>
</tr>
<tr>
<td>Maximum length of an owner name</td>
<td>SQL_MAX_SCHEMA_NAME_LEN (Renamed for ODBC 3.0. Was SQL_MAX_OWNER_NAME_LEN)</td>
<td>32</td>
</tr>
<tr>
<td>Maximum number of characters in a SQL statement</td>
<td>SQL_MAX_STATEMENT_LEN</td>
<td>25000</td>
</tr>
<tr>
<td>Maximum length of a table name</td>
<td>SQL_MAX_TABLE_NAME_LEN</td>
<td>32</td>
</tr>
<tr>
<td>Maximum number of tables allowed in FROM clause</td>
<td>SQL_MAX_TABLES_IN_SELECT</td>
<td>250</td>
</tr>
<tr>
<td>Maximum length of a user name</td>
<td>SQL_MAX_USER_NAME_LEN</td>
<td>32</td>
</tr>
<tr>
<td>Support for multiple result sets</td>
<td>SQL_MULT_RESULT_SETS</td>
<td>N (does not support)</td>
</tr>
<tr>
<td>Support for active transactions on multiple connections</td>
<td>SQL_MULTIPLE_ACTIVE_TXN</td>
<td>Y (supports)</td>
</tr>
<tr>
<td>Whether data source requires length of LONGVARCHAR and LONGVARBINARY data</td>
<td>SQL_NEED_LONG_DATA_LEN</td>
<td>N</td>
</tr>
<tr>
<td>Support for NOT NULL clause in CREATE TABLE statement</td>
<td>SQL_NON_NULLABLE_COLUMNS</td>
<td>SQL_NNC_NON_NULL (supports)</td>
</tr>
<tr>
<td>Where null values are sorted in a list</td>
<td>SQL_NULL_COLLATION</td>
<td>SQL_NC_LOW (sorted at the low end of the list)</td>
</tr>
<tr>
<td>Description</td>
<td>InfoType Argument</td>
<td>Returns</td>
</tr>
<tr>
<td>------------------------------------------</td>
<td>-------------------------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Numeric functions supported</td>
<td>SQL_NUMERIC_FUNCTIONS</td>
<td>SQL_FN_NUM_ABS</td>
</tr>
<tr>
<td>ODBC API conformance level</td>
<td>SQL_ODBC_API_CONFORMANCE (Deprecated in ODBC 3.0.)</td>
<td>SQL_OAC_LEVEL1 (supports level 1)</td>
</tr>
<tr>
<td>SQL Access Group (SAG) conformance</td>
<td>SQL_ODBC_SAG_CLI_CONFORMANCE</td>
<td>SQL_OSCC_COMPLIANT (complies with SAG CLI specification)</td>
</tr>
<tr>
<td>ODBC SQL syntax conformance</td>
<td>SQL_ODBC_SQL_CONFORMANCE (Deprecated in ODBC 3.0.)</td>
<td>SQL_OSC_EXTENDED (supports extended SQL syntax as defined by ODBC)</td>
</tr>
<tr>
<td>Referential integrity syntax support</td>
<td>SQL_ODBC_SQL_OPT_IEF (Renamed for ODBC 3.0.)</td>
<td>See SQL_INTEGRITY</td>
</tr>
<tr>
<td>ODBC version supported by driver manager</td>
<td>SQL_ODBC_VER</td>
<td>Returned by ODBC driver manager</td>
</tr>
<tr>
<td>Whether columns in ORDER BY clause must also be in select list</td>
<td>SQL_ORDER_BY_COLUMNS_IN_SELECT</td>
<td>N</td>
</tr>
<tr>
<td>Support for outer joins</td>
<td>SQL_OUTER_JOINS</td>
<td>Y (supports)</td>
</tr>
<tr>
<td>Term for entity that has owner privileges on objects</td>
<td>SQL_OWNER_TERM (Renamed for ODBC 3.0.)</td>
<td>See SQL_SCHEMA_TERM</td>
</tr>
<tr>
<td>Statements that support use of owner qualifiers</td>
<td>SQL_OWNER_USAGE (Renamed for ODBC 3.0.)</td>
<td>See SQL_SCHEMA_USAGE</td>
</tr>
<tr>
<td>Characteristics of row counts available in a parameterized execution</td>
<td>SQL_PARAM_ARRAY_ROW_COUNTS (ODBC 3.0)</td>
<td>SQL_PARC_NO_BATCH (only one row count available, which is the cumulative row count resulting from the execution of the statement for the entire array of parameters)</td>
</tr>
<tr>
<td>Characteristics of result sets available in a parameterized execution</td>
<td>SQL_PARAM_ARRAY_SELECTS (ODBC 3.0)</td>
<td>0 (does not support)</td>
</tr>
<tr>
<td>Description</td>
<td>InfoType Argument</td>
<td>Returns</td>
</tr>
<tr>
<td>---------------------------------------------------------</td>
<td>----------------------------------------</td>
<td>------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Supported operations in SQLSetPos</td>
<td>SQL_POS_OPERATION</td>
<td>0 (does not support SQLSetPos)</td>
</tr>
<tr>
<td>(Deprecated in ODBC 3.0.)</td>
<td>(Renamed for ODBC 3.0.)</td>
<td>------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Statements that support positioned operations</td>
<td>SQL_POSITIONED_STATEMENTS</td>
<td>SQL_PS_POSITIONED_DELETE</td>
</tr>
<tr>
<td>(Deprecated in ODBC 3.0.)</td>
<td>(Renamed for ODBC 3.0.)</td>
<td>------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Term for procedures</td>
<td>SQL_PROCEDURE_TERM</td>
<td>procedure</td>
</tr>
<tr>
<td>SQL procedures support</td>
<td>SQL_PROCEDURES</td>
<td>Y (supports SQL procedures)</td>
</tr>
<tr>
<td>Position of qualifier in a qualified table name</td>
<td>SQL_QUALIFIER_LOCATION</td>
<td>See SQL_CATALOG_LOCATION</td>
</tr>
<tr>
<td>(Renamed for ODBC 3.0.)</td>
<td>(Renamed for ODBC 3.0.)</td>
<td>------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Character used to separate table, column qualifiers</td>
<td>SQL_QUALIFIER_NAME_SEPARATOR</td>
<td>See SQL_CATALOG_NAME_SEPARATOR</td>
</tr>
<tr>
<td>(Renamed for ODBC 3.0.)</td>
<td>(Renamed for ODBC 3.0.)</td>
<td>------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Term for object that qualifies table names</td>
<td>SQL_QUALIFIER_TERM</td>
<td>See SQL_CATALOG_TERM</td>
</tr>
<tr>
<td>(Renamed for ODBC 3.0.)</td>
<td>(Renamed for ODBC 3.0.)</td>
<td>------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Statements that support qualifiers</td>
<td>SQL_QUALIFIER_USAGE</td>
<td>See SQL_CATALOG_USAGE</td>
</tr>
<tr>
<td>(Renamed for ODBC 3.0.)</td>
<td>(Renamed for ODBC 3.0.)</td>
<td>------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Case-sensitivity of quoted user-supplied names</td>
<td>SQL_QUOTED_IDENTIFIER_CASE</td>
<td>SQL_ICSENSITIVE (case sensitive, stored in mixed case)</td>
</tr>
<tr>
<td>Detect changes to any row in mixed-cursor operations</td>
<td>SQL_ROW_UPDATES</td>
<td>Y</td>
</tr>
<tr>
<td>Term for entity that has owner privileges on objects</td>
<td>SQL_SCHEMA_TERM</td>
<td>owner</td>
</tr>
<tr>
<td>(Renamed for ODBC 3.0. Was SQL_OWNER_TERM)</td>
<td>(Renamed for ODBC 3.0. Was SQL_OWNER_USAGE)</td>
<td>------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Statements that support use of owner qualifiers</td>
<td>SQL_SCHEMA_USAGE</td>
<td>SQL_OU_DML_STATEMENTS</td>
</tr>
<tr>
<td>(Renamed for ODBC 3.0. Was SQL_OWNER_USAGE)</td>
<td>(Renamed for ODBC 3.0. Was SQL_OWNER_USAGE)</td>
<td>------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Concurrency control options supported for scrollable</td>
<td>SQL_SCROLL_CONCURRENCY</td>
<td>0 (does not support)</td>
</tr>
<tr>
<td>cursors</td>
<td>(Deprecated in ODBC 3.0.)</td>
<td>------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Options supported for scrollable cursors</td>
<td>SQL_SCROLL_OPTIONS</td>
<td>SQL_SO_FORWARD_ONLY</td>
</tr>
<tr>
<td>Character to permit wildcard characters in search strings</td>
<td>SQL_SEARCH_PATTERN_ESCAPE</td>
<td>\ (backslash)</td>
</tr>
<tr>
<td>Name of the system where the Dharma/ SQL data source</td>
<td>SQL_SERVER_NAME</td>
<td>(string containing the name)</td>
</tr>
<tr>
<td>resides</td>
<td>(string containing the name)</td>
<td>------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Special characters allowed in user-supplied names</td>
<td>SQL_SPECIAL_CHARACTERS</td>
<td>_ (underscore)</td>
</tr>
<tr>
<td>Level of SQL-92 support</td>
<td>SQL_SQL_CONFORMANCE</td>
<td>SQL_SC_SQL92_ENTRY (entry level SQL-92 compliant)</td>
</tr>
<tr>
<td>(ODBC 3.0)</td>
<td>(ODBC 3.0)</td>
<td>------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Datetime scalar functions supported</td>
<td>SQL_SQL92_DATETIME_FUNCTIONS</td>
<td>SQL_SDF_CURRENT_DATE</td>
</tr>
<tr>
<td>(ODBC 3.0)</td>
<td>(ODBC 3.0)</td>
<td>------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>

3–12 ODBC Driver Conformance
<table>
<thead>
<tr>
<th>Description</th>
<th>InfoType Argument</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Behavior of DELETE statement that refers to a foreign key</td>
<td>SQL_SQL92_FOREIGN_KEY_DELETE_RULE</td>
<td>SQL_SFKU_NO_ACTION</td>
</tr>
<tr>
<td>(ODBC 3.0)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Behavior of UPDATE statement that refers to a foreign key</td>
<td>SQL_SQL92_FOREIGN_KEY_UPDATE_RULE</td>
<td>SQL_SFKU_NO_ACTION</td>
</tr>
<tr>
<td>(ODBC 3.0)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GRANT statement clauses supported</td>
<td>SQL_SQL92_GRANT</td>
<td>SQL_SG_DELETE_TABLE</td>
</tr>
<tr>
<td>(ODBC 3.0)</td>
<td></td>
<td>SQL_SG_INSERT_TABLE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_SG_INSERT_COLUMN</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_SG_REFERENCES_TABLE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_SG_REFERENCES_COLUMN</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_SG_SELECT_TABLE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_SG_UPDATE_TABLE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_SG_UPDATE_COLUMN</td>
</tr>
<tr>
<td>Numeric scalar functions supported</td>
<td>SQL_SQL92_NUMERIC_VALUE_FUNCTIONS</td>
<td>SQL_SNVF_CHAR_LENGTH</td>
</tr>
<tr>
<td>(ODBC 3.0)</td>
<td></td>
<td>SQL_SNVF_CHARACTER_LENGTH</td>
</tr>
<tr>
<td>Predicates supported</td>
<td>SQL_SQL92_PREDICATES</td>
<td>SP_EXISTS</td>
</tr>
<tr>
<td>(ODBC 3.0)</td>
<td></td>
<td>SQL_SP_ISNOTNULL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_SP_ISSNULL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_SP_UNIQUE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_SP_LIKE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_SP_IN</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_SP_BETWEEN</td>
</tr>
<tr>
<td>Relational join operators supported</td>
<td>SQL_SQL92_RELATIONAL_JOIN_OPERATORS</td>
<td>SQL_SRJO_INNER_JOIN</td>
</tr>
<tr>
<td>(ODBC 3.0)</td>
<td></td>
<td>SQL_SRJO_LEFT_OUTER_JOIN</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_SRJO_NATURAL_JOIN</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_SRJO_RIGHT_OUTER_JOIN</td>
</tr>
<tr>
<td>REVOKE statement clauses supported</td>
<td>SQL_SQL92_REVOKE</td>
<td>SQL_SR_GRANT_OPTION_FOR</td>
</tr>
<tr>
<td>(ODBC 3.0)</td>
<td></td>
<td>SQL_SR.Cascade</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_SR_RESTRICT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_SR_DELETE_TABLE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_SR_INSERT_TABLE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_SR_INSERT_COLUMN</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_SR_REFERENCES_TABLE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_SR_REFERENCES_COLUMN</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_SR_SELECT_TABLE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_SR_UPDATE_TABLE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_SR_UPDATE_COLUMN</td>
</tr>
<tr>
<td>Row value constructor expressions supported</td>
<td>SQL_SQL92_ROW_VALUE_CONSTRUCTOR</td>
<td>0 (does not support)</td>
</tr>
<tr>
<td>(ODBC 3.0)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>String scalar functions supported</td>
<td>SQL_SQL92_STRING_FUNCTIONS</td>
<td>SQL_SSF_CONVERT</td>
</tr>
<tr>
<td>(ODBC 3.0)</td>
<td></td>
<td>SQL_SSF_LOWER</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_SSF_UPPER</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_SSF_SUBSTRING</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_SSF_TRANSLATE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_SSF_TRIM_LEADING</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_SSF_TRIM_TRAILING</td>
</tr>
<tr>
<td>Value expressions supported</td>
<td>SQL_SQL92_VALUE_EXPRESSIONS</td>
<td>SQL_SVE_CASE</td>
</tr>
<tr>
<td>(ODBC 3.0)</td>
<td></td>
<td>SQL_SVE_CAST</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_SVE_COALESCE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_SVE_NULLIF</td>
</tr>
<tr>
<td>CLI standards to which the driver conforms</td>
<td>SQL_STANDARD_CLI_CONFORMANCE</td>
<td>SQL_SCC_XOPEN_CLI_VERSION1</td>
</tr>
<tr>
<td>(ODBC 3.0)</td>
<td></td>
<td>(conforms to X/Open CLI version 1)</td>
</tr>
<tr>
<td>Description</td>
<td>InfoType Argument</td>
<td>Returns</td>
</tr>
<tr>
<td>----------------------------------------------------------------------------</td>
<td>-------------------------------------------------------------</td>
<td>------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Supported attributes of a static cursor: subset 1</td>
<td>SQL_STATIC_CURSOR_ATTRIBUTES1</td>
<td>SQL_CA1_NEXT (supports SQL_FETCH_NEXT argument to SQLFetchScroll for static cursors)</td>
</tr>
<tr>
<td>Supported attributes of a static cursor: subset 2</td>
<td>SQL_STATIC_CURSOR_ATTRIBUTES2</td>
<td>0 (does not support any attributes in the subset))</td>
</tr>
<tr>
<td>Whether static cursor changes are detectable</td>
<td>SQL_STATIC_SENSITIVITY</td>
<td>0 (does not support static cursors)</td>
</tr>
<tr>
<td>String functions supported</td>
<td>SQL_STRING_FUNCTIONS</td>
<td>SQL_FN_STR_CONCAT</td>
</tr>
<tr>
<td>Predicates that support subqueries</td>
<td>SQL_SUBQUERIES</td>
<td>SQL_SQ_CORRELATED_SUBQUERIES</td>
</tr>
<tr>
<td>System functions supported</td>
<td>SQL_SYSTEM_FUNCTIONS</td>
<td>SQL_FN_SYS_DBNAME</td>
</tr>
<tr>
<td>Term for tables</td>
<td>SQL_TABLE_TERM</td>
<td>table</td>
</tr>
<tr>
<td>Timestamp intervals supported for TIMESTAMPDIFF function</td>
<td>SQL_TIMEDATE_DIFF_INTERVALS</td>
<td>SQL_TSI_FRAC_SECOND</td>
</tr>
<tr>
<td>Timestamp intervals supported for TIMESTAMPADD function</td>
<td>SQL_TIMEDATE_ADD_INTERVALS</td>
<td>SQL_TSI_FRAC_SECOND</td>
</tr>
</tbody>
</table>
### 3.3 Supported Environment, Connection, and Statement Attributes

Table 3–2 details the driver attributes that the Dharma/ SQL ODBC Driver supports. Applications can set and retrieve supported driver attributes through the following routines:

- `SQLGetEnvAttr` and `SQLSetEnvAttr` for environment attributes
- `SQLGetConnectAttr` and `SQLSetConnectAttr` for connection attributes
- `SQLGetStmtAttr` and `SQLSetStmtAttr` for statement attributes

#### Table 3–2   Supported Environment, Connection, and Statement Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Supported?</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ODBC Environment Attributes</strong></td>
<td></td>
</tr>
<tr>
<td>SQL_ATTR_CONNECTION_POOLING</td>
<td>No</td>
</tr>
<tr>
<td>SQL_ATTR_CP_MATCH</td>
<td>No</td>
</tr>
<tr>
<td>SQL_ATTR_ODBC_VER</td>
<td>Yes</td>
</tr>
<tr>
<td>Attribute</td>
<td>Supported?</td>
</tr>
<tr>
<td>---------------------------------------</td>
<td>------------</td>
</tr>
<tr>
<td>SQL_ATTR_OUTPUT_NTS</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>ODBC Connection Attributes</strong></td>
<td></td>
</tr>
<tr>
<td>SQL_ATTR_ACCESS_MODE</td>
<td>Yes (supports both read_only and read_write connection modes)</td>
</tr>
<tr>
<td>SQL_ATTR_ASYNC_ENABLE</td>
<td>Yes (supports only SQL_ASYNC_ENABLE_OFF, for other values returns warning “Option value changed” and sets to default)</td>
</tr>
<tr>
<td>SQL_ATTR_AUTO_IPD</td>
<td>Yes</td>
</tr>
<tr>
<td>SQL_ATTR_AUTOCOMMIT</td>
<td>Yes</td>
</tr>
<tr>
<td>SQL_ATTR_CONNECTION_TIMEOUT</td>
<td>Yes (supports only 0, for other values returns warning “Option value changed” and sets to default)</td>
</tr>
<tr>
<td>SQL_ATTR_CURRENT_CATALOG</td>
<td>No</td>
</tr>
<tr>
<td>SQL_ATTR_LOGIN_TIMEOUT</td>
<td>Yes (supports only 0, for other values returns warning “Option value changed” and sets to default)</td>
</tr>
<tr>
<td>SQL_ATTR_ODBC_CURSORS</td>
<td>Yes</td>
</tr>
<tr>
<td>SQL_ATTR_PACKET_SIZE</td>
<td>Yes (supports only default, for other values returns warning “Option value changed” and sets to default)</td>
</tr>
<tr>
<td>SQL_ATTR_QUIET_MODE</td>
<td>Yes (supports only default, for other values returns warning “Option value changed” and sets to default)</td>
</tr>
<tr>
<td>SQL_ATTR_TRACE</td>
<td>No</td>
</tr>
<tr>
<td>SQL_ATTR_TRACEFILE</td>
<td>No</td>
</tr>
<tr>
<td>SQL_ATTR_TRANSLATE_DLL</td>
<td>No</td>
</tr>
<tr>
<td>SQL_ATTR_TRANSLATE_LIB</td>
<td>No</td>
</tr>
<tr>
<td>SQL_ATTR_TRANSLATE_OPTION</td>
<td>No</td>
</tr>
<tr>
<td>SQL_ATTR_TXN_ISOLATION</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>ODBC Statement Attributes</strong></td>
<td></td>
</tr>
<tr>
<td>SQL_ATTR_APP_PARAM_DESC</td>
<td>Yes</td>
</tr>
<tr>
<td>SQL_ATTR_APP_ROW_DESC</td>
<td>Yes</td>
</tr>
<tr>
<td>SQL_ATTR_ASYNC_ENABLE</td>
<td>Yes (supports only SQL_ASYNC_ENABLE_OFF, for other values returns warning “Option value changed” and sets to default)</td>
</tr>
<tr>
<td>Attribute</td>
<td>Supported?</td>
</tr>
<tr>
<td>------------------------------------------------</td>
<td>---------------------------------------------------------------------------</td>
</tr>
<tr>
<td>SQL_ATTR_CONCURRENCY</td>
<td>Yes (supports only SQL_CONCUR_READ_ONLY, for other values returns warning “Option value changed” and sets to default)</td>
</tr>
<tr>
<td>SQL_ATTR_CURSOR_SCROLLABLE</td>
<td>No</td>
</tr>
<tr>
<td>SQL_ATTR_CURSOR_TYPE</td>
<td>Yes (supports only SQL_FORWARD_ONLY, for other values returns warning “Option value changed” and sets to default)</td>
</tr>
<tr>
<td>SQL_ATTR_CURSOR_SENSITIVITY</td>
<td>Yes (supports only SQL_INSENSITIVE, for other values returns warning “Option value changed” and sets to default)</td>
</tr>
<tr>
<td>SQL_ATTR_ENABLE_AUTO_IPD</td>
<td>No</td>
</tr>
<tr>
<td>SQL_ATTR_FETCH_BOOKMARK_PTR</td>
<td>No</td>
</tr>
<tr>
<td>SQL_ATTR_IMP_PARAM_DESC</td>
<td>Yes</td>
</tr>
<tr>
<td>SQL_ATTR_IMP_ROW_DESC</td>
<td>Yes</td>
</tr>
<tr>
<td>SQL_ATTR_KEYSET_SIZE</td>
<td>No</td>
</tr>
<tr>
<td>SQL_ATTR_MAX_LENGTH</td>
<td>No</td>
</tr>
<tr>
<td>SQL_ATTR_MAX_ROWS</td>
<td>Yes</td>
</tr>
<tr>
<td>SQL_ATTR_MAX_ROWS</td>
<td>Yes</td>
</tr>
<tr>
<td>SQL_ATTR_METADATA_ID</td>
<td>Yes</td>
</tr>
<tr>
<td>SQL_ATTR_NOSCAN</td>
<td>No</td>
</tr>
<tr>
<td>SQL_ATTR_PARAM_BIND_OFFSET_PTR</td>
<td>Yes</td>
</tr>
<tr>
<td>SQL_ATTR_PARAM_BIND_TYPE</td>
<td>Yes</td>
</tr>
<tr>
<td>SQL_ATTR_PARAM_OPERATION_PTR</td>
<td>Yes</td>
</tr>
<tr>
<td>SQL_ATTR_PARAM_STATUS_PTR</td>
<td>Yes</td>
</tr>
<tr>
<td>SQL_ATTR_PARAMS_PROCESSED_PTR</td>
<td>Yes</td>
</tr>
<tr>
<td>SQL_ATTR_PARAMSET_SIZE</td>
<td>Yes</td>
</tr>
<tr>
<td>SQL_ATTR_QUERY_TIMEOUT</td>
<td>Yes (supports only 0, for other values returns warning “Option value changed” and sets to default)</td>
</tr>
<tr>
<td>SQL_ATTR_RETRIEVE_DATA</td>
<td>Yes</td>
</tr>
<tr>
<td>SQL_ATTR_ROW_ARRAY_SIZE</td>
<td>Yes</td>
</tr>
<tr>
<td>SQL_ATTR_ROW_BIND_OFFSET_PTR</td>
<td>Yes</td>
</tr>
<tr>
<td>SQL_ATTR_ROW_BIND_TYPE</td>
<td>Yes</td>
</tr>
</tbody>
</table>
3.4 Supported ODBC API Functions

The ODBC Driver supports all Core and Level 1 API functions, and most Level 2 functions. Here is a complete list of the functions the driver supports. Applications can request this same information through the SQLGetFunctions function with a FunctionId of SQL_API_ODBC3_ALL_FUNCTIONS.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Supported?</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQL_ATTR_ROW_NUMBER</td>
<td>No</td>
</tr>
<tr>
<td>SQL_ATTR_ROW_OPERATION_PTR</td>
<td>Yes</td>
</tr>
<tr>
<td>SQL_ATTR_ROW_STATUS_PTR</td>
<td>Yes</td>
</tr>
<tr>
<td>SQL_ATTR_ROWS_FETCHED_PTR</td>
<td>Yes</td>
</tr>
<tr>
<td>SQL_ATTR_SIMULATE_CURSOR</td>
<td>No</td>
</tr>
<tr>
<td>SQL_ATTR_USE_BOOKMARKS</td>
<td>No</td>
</tr>
</tbody>
</table>

SQLAllocConnect
(Deprecated in ODBC 3.0)

SQLAllocEnv
(Deprecated in ODBC 3.0)

SQLAllocHandle

SQLAllocStmt
(Deprecated in ODBC 3.0)

SQLBindCol

SQLBindParam
(Deprecated in ODBC 3.0)

SQLBindParameter

SQLBrowseConnect

SQLCancel

SQLCloseCursor

SQLColAttribute

SQLColAttributes
(Deprecated in ODBC 3.0)

SQLColumnPrivileges

SQLColumns

SQLConnect

SQLCopyDesc
3.5 Supported Data Types

The Dharma DataLink Driver supports data types that it maps to corresponding ODBC SQL data types. Table 3-3 lists the Dharma data types and the corresponding ODBC data types.
### Table 3–3 Dharma and Corresponding ODBC Data Types

<table>
<thead>
<tr>
<th>Dharma Data Type</th>
<th>ODBC Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIGINT</td>
<td>SQL_BIGINT</td>
</tr>
<tr>
<td>BINARY</td>
<td>SQL_BINARY</td>
</tr>
<tr>
<td>BIT</td>
<td>SQL_BIT</td>
</tr>
<tr>
<td>CHARACTER</td>
<td>SQL_CHAR</td>
</tr>
<tr>
<td>DATE</td>
<td>SQL_DATE</td>
</tr>
<tr>
<td>DECIMAL</td>
<td>SQL_DECIMAL</td>
</tr>
<tr>
<td>DOUBLE PRECISION</td>
<td>SQL_DOUBLE</td>
</tr>
<tr>
<td>FLOAT</td>
<td>SQL_DOUBLE</td>
</tr>
<tr>
<td>FLOAT</td>
<td>SQL_FLOAT</td>
</tr>
<tr>
<td>INTEGER</td>
<td>SQL_INTEGER</td>
</tr>
<tr>
<td>LVARBINARY</td>
<td>SQL_LONGVARBINARY</td>
</tr>
<tr>
<td>LVARCHAR</td>
<td>SQL_LONGVARCHAR</td>
</tr>
<tr>
<td>NUMERIC</td>
<td>SQL_NUMERIC</td>
</tr>
<tr>
<td>REAL</td>
<td>SQL_REAL</td>
</tr>
<tr>
<td>SMALLINT</td>
<td>SQL_SMALLINT</td>
</tr>
<tr>
<td>TIME</td>
<td>SQL_TIME</td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td>SQL_TIMESTAMP</td>
</tr>
<tr>
<td>TINYINT</td>
<td>SQL_TINYINT</td>
</tr>
<tr>
<td>VARBINARY</td>
<td>SQL_VARBINARY</td>
</tr>
<tr>
<td>VARCHAR</td>
<td>SQL_VARCHAR</td>
</tr>
</tbody>
</table>
This chapter describes language elements that are common to many SQL statements. Syntax diagrams in other chapters often refer to these language elements without detailed explanation. The major syntax elements described in the following sections are:

**Identifiers** (page 4-3) are user-supplied names for elements such as tables, views, cursors, and columns. SQL statements use those names to refer to the elements.

**Data types** (page 4-4) control how SQL stores column values. CREATE TABLE statements specify data types for columns.

**Query expressions** (page 4-13) retrieve values from tables. Query expressions form the basis of other SQL statements and syntax elements.

**Search conditions** (page 4-24) specify a condition that is true or false about a given row or group of rows. Query expressions and UPDATE statements specify search conditions to restrict the number of rows in the result table.

**Expressions** (page 4-30) are a symbol or string of symbols used to represent or calculate a single value in an SQL statement. When SQL encounters an expression, it retrieves or calculates the value represented by the expression and uses that value when it executes the statement.

**Literals** (page 4-35) are a type of SQL expression that specify a constant value. Some SQL constructs allow literals but prohibit other forms of expressions.

**Date-time format strings** (page 4-40) control the output of date and time values. Dharma/SQL interprets format strings and replaces them with formatted values.

**Functions** (page 4-43) are a type of SQL expression that return a value based on the argument they are supplied. Aggregate functions calculate a single value for a collection of rows in a result table. Scalar functions calculate a value based on another single value.
4.1 SQL Features: Lite and Professional Editions

The Lite and Professional editions of the Dharma DataLink SDK support different levels of SQL functionality. Table 4–1 details which SQL features are supported by the Lite and Professional editions.

<table>
<thead>
<tr>
<th>SQL Support</th>
<th>Lite</th>
<th>Pro</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECT/ INSERT/ UPDATE/ DELETE</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>INSERT...SELECT</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CREATE/ DROP TABLE</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CREATE/ DROP INDEX</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CREATE/ DROP SYNONYM</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GROUP BY clause</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>AVG, COUNT, MAX, MIN, SUM</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>20 SQL data types (including LONG, DATE)</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Interactive SQL utility</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Transaction support</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>User authentication</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Set operators (UNION/ INTERSECT/ MINUS)</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>The UNION, INTERSECT, and MINUS set operators manipulate how SQL returns result sets from multiple query expressions (Section 4.4).</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Library of over 100 scalar functions (Section 4.9.2)</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>HAVING clause: The HAVING clause in query expressions (Section 4.4) specifies a search condition for the row groupings specified in the GROUP BY clause.</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Table/ column privileges: GRANT (Section 5.11) and REVOKE (Section 5.14) statements.</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Outer Joins (Section 4.4.2)</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>CASE expressions (Sections 4.6.4 and 4.9.2.8)</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Derived tables: Derived tables are specified through query expressions in the FROM clause of another query expression (Section 4.4).</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Subqueries: Subqueries are query expressions used within a search condition or as an expression. Subqueries can occur in basic predicates (Section 4.5.3), quantified predicates (Section 4.5.4), IN predicates (Section 4.5.10), and expressions (Section 4.6)</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>View support: CREATE (Section 5.5) and DROP VIEW (Section 5.10) statements.</td>
<td></td>
<td>✓</td>
</tr>
</tbody>
</table>
### SQL Support

<table>
<thead>
<tr>
<th>SQL Support</th>
<th>Lite</th>
<th>Professional</th>
</tr>
</thead>
<tbody>
<tr>
<td>SET SCHEMA statement: (Section 5.16) changes default qualifier for database objects</td>
<td>✓</td>
<td></td>
</tr>
</tbody>
</table>

The following SQL features are not supported in either the Lite or Professional editions:

- Multiple character sets and collation sequences (page 4-6)
- ALTER TABLE statement (referred to in the discussion of data types (Section 4.3) and in the discussion of DROP TABLE (Section 5.9))
- CREATE TABLE statement (Section 5.4): TABLESPACE, PCTFREE, or STORAGE_MANAGER clauses
- CREATE INDEX statement (Section 5.2): PCTFREE clause

### 4.2 SQL Identifiers

SQL syntax requires users to supply names for elements such as tables, views, cursors, and columns when they define them. SQL statements must use those names to refer to the table, view, or other element. In syntax diagrams, SQL identifiers are shown in lowercase type.

The maximum length for SQL identifiers is 32 characters.

There are two types of SQL identifiers:

- Conventional identifiers
- Delimited identifiers enclosed in double quotation marks

#### 4.2.1 Conventional Identifiers

Unless they are delimited identifiers (see "Delimited Identifiers"), SQL identifiers must:

- Begin with an uppercase or lowercase letter
- Contain only letters, digits, or the underscore character (_)
- Not be reserved words

Except for delimited identifiers, SQL does not distinguish between uppercase and lowercase letters in SQL identifiers. It converts all names to lower case, but statements can refer to the names in mixed case. The following examples show some of the characteristics of conventional identifiers:
-- Names are case-insensitive:
CREATE TABLE Test (Column1 CHAR);
INSERT INTO TEST (COLUMN1) VALUES('1');
1 record inserted.
SELECT * FROM TEST;
COL
---
1
1 record selected
TABLE TEST;

<table>
<thead>
<tr>
<th>COLNAME</th>
<th>NULL ?</th>
<th>TYPE</th>
<th>LENGTH</th>
</tr>
</thead>
<tbody>
<tr>
<td>column1</td>
<td>------</td>
<td>------</td>
<td>--------</td>
</tr>
</tbody>
</table>

-- Cannot use reserved words:
CREATE TABLE TABLE (COL1 CHAR);
CREATE TABLE TABLE (COL1 CHAR);
* error(-20003): Syntax error

### 4.2.2 Delimited Identifiers

**Delimited identifiers** are SQL identifiers enclosed in double quotation marks (".

Enclosing a name in double quotation marks preserves the case of the name and allows it to be a reserved word and special characters. Special characters are any characters other than letters, digits, or the underscore character.) Subsequent references to a delimited identifier must also use enclosing double quotation marks. To include a double-quotatation-mark character in a delimited identifier, precede it with another double-quotatation mark.

The following SQL example shows some ways to create and refer to delimited identifiers:

```
CREATE TABLE "delimited ids"
    ( "***" CHAR(10),
      "_uscore" CHAR(10),
      "**quote" CHAR(10),
      " space" CHAR(10) );
INSERT INTO "delimited ids" ("***") VALUES('text string');
1 record inserted.
SELECT * FROM "delimited ids";
"            _USCORE      "QUOTE        SPACE
-            -------      ------       ------
text strin
1 record selected
```

### 4.3 Data Types

The SQL statements CREATE TABLE and ALTER TABLE specify data types for each column in the tables they define. This section describes the data types SQL supports for table columns.

There are several categories of SQL data types:

- Character
- Exact numeric
• Approximate numeric
• Date-time
• Bit String

All the data types can store null values. A null value indicates that the value is not
known and is distinct from all non-null values.

Syntax

data_type ::
    char_data_type
    | exact_numeric_data_type
    | approx_numeric_data_type
    | date_time_data_type
    | bit_string_data_type

4.3.1 Character Data Types

See "Character String Literals" on page 4-36 for details on specifying values to be stored
in character columns.

Syntax

char_data_type ::
    { CHARACTER | CHAR } [(length)] [ CHARACTER SET charset-name ]
    | { CHARACTER VARYING | CHAR VARYING | VARCHAR } [(length)]
    | LVARCHAR | LONG VARCHAR
    | { NATIONAL CHARACTER | NATIONAL CHAR | NCHAR } [(length)]
    | { NATIONAL CHARACTER VARYING | NATIONAL VARCHAR } [(length)]

Arguments

{ CHARACTER | CHAR } [(length)] [ CHARACTER SET charset-name ]
Type CHARACTER (abbreviated as CHAR) corresponds to a null terminated character
string with the maximum length specified. The default length is 1. The maximum
length is 2000.

The optional CHARACTER SET clause specifies an alternative character set supported
by the underlying storage system. "Specifying the Character Set for Character Data
Types" on page 4-6 describes general considerations for using this clause. See the
documentation for the underlying storage system for details on valid values for charset-
name, if any.

{ NATIONAL CHARACTER | NATIONAL CHAR | NCHAR } [(length)]
Type NATIONAL CHARACTER is equivalent to type CHARACTER with a
CHARACTER SET clause specifying the character set designated as NATIONAL
CHARACTER by the underlying storage system. See "Specifying the Character Set for
Character Data Types" on page 4-6.

{ CHARACTER VARYING | CHAR VARYING | VARCHAR } [(length)]
    [ CHARACTER SET charset-name ]
Type CHARACTER VARYING corresponds to a variable-length character string with
the maximum length specified.
The optional CHARACTER SET clause specifies an alternative character set supported by the underlying storage system. "Specifying the Character Set for Character Data Types" on page 4-6 describes general considerations for using this clause. See the documentation for the underlying storage system for details on valid values for CHARSET-name, if any.

The default length for columns defined as CHARACTER VARYING is 1. The maximum length depends on whether the data type specification includes the CHARACTER SET clause:

- If it does not specify CHARACTER SET, the maximum length is 2000.
- If it does specify CHARACTER SET, the maximum length is 32752.

\{ NATIONAL CHARACTER VARYING | NATIONAL VARCHAR \} \[(length)\]

Type NATIONAL CHARACTER VARYING is equivalent to type CHARACTER VARYING with a CHARACTER SET clause specifying the character set designated as NATIONAL CHARACTER by the underlying storage system. See "Specifying the Character Set for Character Data Types" on page 4-6.

LVARCHAR | LONG VARCHAR

Type LONG VARCHAR corresponds to an arbitrarily-long character string with a maximum length limited by the specific storage system.

The arbitrary size and unstructured nature of LONG data types restrict where they can be used.

- LONG columns are allowed in select lists of query expressions and in INSERT statements.
- INSERT statements can store data from columns of any type into a LONG VARCHAR column, but LONG VARCHAR data cannot be stored in any other type.
- CONTAINS predicates are the only predicates that allow LONG columns (and then only if the underlying storage system explicitly supports CONTAINS predicates).
- Conditional expressions, arithmetic expressions, and functions cannot specify LONG columns.
- UPDATE statements cannot specify LONG columns.

Specifying the Character Set for Character Data Types

SQL allows column definitions of type CHARACTER and CHARACTER VARYING to specify an alternate character set. If you omit the CHARACTER SET clause in a column definition, the default character set is the standard 7-bit ASCII character set, shown in Table 4–2.

The character set associated with a table column defines which set of characters can be stored in that column, how those characters are represented in the underlying storage system, and how character strings using the character set compare with each other:

- The set of characters allowed in a character set is called the repertoire of the character set. The default ASCII character set has a repertoire of 128 characters, shown in Table 4–2. Other character sets, such as Unicode, specify much larger repertoires and include characters for many languages other than English.
The storage representation for a character set is called the **form of use** of the character set. The form of use for the default ASCII character set is a single byte (or octet) containing a number designating a particular ASCII character, also shown in Table 4-2. Other character sets, such as Unicode, use two or more bytes (or a varying number of bytes, depending on the character) for each character.

The rules used to control how character strings compare with each other is called the **collation** of a character set. Each character set specifies a collating sequence that defines relative values of each character for comparing, merging and sorting character strings. Character sets may also define additional collations that override the default for a character set. SQL statements specify such collations with the COLLATE clause in character column definitions, basic predicates, the GROUP BY clause of query expressions, and the ORDER BY clause of SELECT statements.

Table 4-2 shows the characters in the default ASCII character set and the decimal values that designate each character. (This is the default representation on UNIX; other operating systems may have slight differences in their definitions of the default ASCII character set.) The values also define the collating sequence for the character set. For instance, this collating sequence specifies that a lowercase letter is always a larger value than an uppercase letter.

<table>
<thead>
<tr>
<th>Val</th>
<th>Char</th>
<th>Val</th>
<th>Char</th>
<th>Val</th>
<th>Char</th>
<th>Val</th>
<th>Char</th>
<th>Val</th>
<th>Char</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>NUL</td>
<td>1</td>
<td>SOH</td>
<td>2</td>
<td>STX</td>
<td>3</td>
<td>ETX</td>
<td>4</td>
<td>EOT</td>
</tr>
<tr>
<td>5</td>
<td>ENQ</td>
<td>6</td>
<td>ACK</td>
<td>7</td>
<td>BEL</td>
<td>8</td>
<td>BS</td>
<td>9</td>
<td>HT</td>
</tr>
<tr>
<td>10</td>
<td>NL</td>
<td>11</td>
<td>VT</td>
<td>12</td>
<td>NP</td>
<td>13</td>
<td>CR</td>
<td>14</td>
<td>SO</td>
</tr>
<tr>
<td>15</td>
<td>SI</td>
<td>16</td>
<td>DLE</td>
<td>17</td>
<td>DC1</td>
<td>18</td>
<td>DC2</td>
<td>19</td>
<td>DC3</td>
</tr>
<tr>
<td>20</td>
<td>DC4</td>
<td>21</td>
<td>NAK</td>
<td>22</td>
<td>SYN</td>
<td>23</td>
<td>ETB</td>
<td>24</td>
<td>CAN</td>
</tr>
<tr>
<td>25</td>
<td>EM</td>
<td>26</td>
<td>SUB</td>
<td>27</td>
<td>ESC</td>
<td>28</td>
<td>FS</td>
<td>29</td>
<td>GS</td>
</tr>
<tr>
<td>30</td>
<td>RS</td>
<td>31</td>
<td>US</td>
<td>32</td>
<td>SP</td>
<td>33</td>
<td>!</td>
<td>34</td>
<td>&quot;</td>
</tr>
<tr>
<td>35</td>
<td>#</td>
<td>36</td>
<td>$</td>
<td>37</td>
<td>%</td>
<td>38</td>
<td>&amp;</td>
<td>39</td>
<td>'</td>
</tr>
<tr>
<td>40</td>
<td>(</td>
<td>41</td>
<td>)</td>
<td>42</td>
<td>*</td>
<td>43</td>
<td>+</td>
<td>44</td>
<td>,</td>
</tr>
<tr>
<td>45</td>
<td>-</td>
<td>46</td>
<td>.</td>
<td>47</td>
<td>/</td>
<td>48-57</td>
<td>0-9</td>
<td>58</td>
<td>:</td>
</tr>
<tr>
<td>59</td>
<td>;</td>
<td>60</td>
<td>&lt;</td>
<td>61</td>
<td>=</td>
<td>62</td>
<td>&gt;</td>
<td>63</td>
<td>?</td>
</tr>
<tr>
<td>64</td>
<td>@</td>
<td>91</td>
<td>[</td>
<td>92</td>
<td>\</td>
<td>93</td>
<td>]</td>
<td>94</td>
<td>^</td>
</tr>
<tr>
<td>95</td>
<td>_</td>
<td>96</td>
<td>`</td>
<td>65-90</td>
<td>A-Z</td>
<td>97-122</td>
<td>a-z</td>
<td>123</td>
<td>{</td>
</tr>
<tr>
<td>124</td>
<td>]</td>
<td>125</td>
<td>-</td>
<td>126</td>
<td>~</td>
<td>127</td>
<td>DEL</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Dharma/SQL supports the ASCII_SET character-set name and a collation sequence named CASE_INSENSITIVE. The ASCII_SET character set is the same as the default and is provided to test and illustrate the CHARACTER SET syntax. The CASE_INSENSITIVE collation sequence overrides the default ASCII collation and specifies the same comparison values for a lowercase letter as its uppercase counterpart.

The following example uses the ISQL TABLE command to show two tables. bigwigs is defined with the CASE_INSENSITIVE collating sequence for its column, and bigwigs2 is not. Both tables contain the same data. SELECT statements show the difference in collation:
Support for character sets other than ASCII_SET and collations other than CASE_INSENSITIVE depends on the underlying storage system. When statements refer to a character set or collation name that is not supported by the underlying storage system, SQL generates an error:

```
ISQL> create table badset (c1 char(10) character set bad_set);
create table badset (c1 char(10) character set bad_set);
* error(-20239): Invalid character set name specified
ISQL> create table badseq (c1 char(10) collate bad_seq);
create table badseq (c1 char(10) collate bad_seq);
* error(-20240): Invalid collation name specified
```

The NATIONAL CHARACTER reserved words in SQL are shorthand notation for specifying a particular character set supported by the underlying storage system. If the underlying storage system designates a supported character set as the national character set, column definitions can use the NATIONAL CHARACTER (or NATIONAL CHARACTER VARYING) data type instead of explicitly specifying the character set name in the CHARACTER SET clause of the CHARACTER (or CHARACTER VARYING) data type. If the underlying storage system does not associate another character set with the NATIONAL CHARACTER clause, the default national character set is the ASCII_SET character set.

### 4.3.2 Exact Numeric Data Types

See "Numeric Literals" for details on specifying values to be stored in numeric columns.
Syntax

```sql
exact_numeric_data_type ::
  TINYINT
  | SMALLINT
  | INTEGER
  | BIGINT
  | NUMERIC | NUMBER [ ( precision [ , scale ] ) ]
  | DECIMAL [(precision, scale)]
  | MONEY [(precision)]
```

Arguments

**TINYINT**
Type TINYINT corresponds to an integer value stored in one byte. The range of TINYINT is -128 to 127.

**SMALLINT**
Type SMALLINT corresponds to an integer value of length 2 bytes.

The range of SMALLINT is -32768 to +32767.

**INTEGER**
Type INTEGER corresponds to an integer of length 4 bytes.

The range of values for INTEGER columns is -2 ** 31 to 2 ** 31 -1.

**BIGINT**
Type BIGINT corresponds to an integer of length 8 bytes. The range of values for BIGINT columns is -2 ** 63 to 2 ** 63 -1.

**NUMERIC | NUMBER [ ( precision [ , scale ] ) ]**
Type NUMERIC corresponds to a number with the given precision (maximum number of digits) and scale (the number of digits to the right of the decimal point). By default, NUMERIC columns have a precision of 32 and scale of 0. If NUMERIC columns omit the scale, the default scale is 0.

The range of values for a NUMERIC type column is -n to +n where n is the largest number that can be represented with the specified precision and scale. If a value exceeds the precision of a NUMERIC column, SQL generates an overflow error. If a value exceeds the scale of a NUMERIC column, SQL rounds the value.

NUMERIC type columns cannot specify a negative scale or specify a scale larger than the precision.

The following example shows what values will fit in a column created with a precision of 3 and scale of 2:
insert into t4 values(33.33);
error(-20052): Overflow error
insert into t4 values(33.9);
error(-20052): Overflow error
insert into t4 values(3.3);
1 record inserted.
insert into t4 values(33);
error(-20052): Overflow error
insert into t4 values(33.33);
1 record inserted.
insert into t4 values(3.33333);
1 record inserted.
insert into t4 values(3.3555);
1 record inserted.
select * from t4;
   C1
---
  3.30
  3.33
  3.33
  3.36
4 records selected

DECIMAL [(precision, scale)]
Type DECIMAL is equivalent to type NUMERIC.

MONEY [(precision)]
Type MONEY is equivalent to type NUMERIC with a fixed scale of 2.

4.3.3 Approximate Numeric Data Types

See "Numeric Literals" for details on specifying values to be stored in numeric columns.

Syntax

approx_numeric_data_type ::=
    REAL
    | DOUBLE PRECISION
    | FLOAT [ (precision) ]

Arguments

REAL
Type REAL corresponds to a single precision floating point number equivalent to the C language float type.

DOUBLE PRECISION
Type DOUBLE PRECISION corresponds to a double precision floating point number equivalent to the C language double type.

FLOAT [ (precision) ]
Type FLOAT corresponds to a double precision floating point number of the given precision. By default, FLOAT columns have a precision of 8.
4.3.4 Date-Time Data Types

See "Date-Time Literals" for details on specifying values to be stored in date-time columns. See "Date-Time Format Strings" for details on using format strings to specify the output format of date-time columns.

Syntax

date_time_data_type ::
   DATE
   | TIME
   | TIMESTAMP

Arguments

DATE
Type DATE stores a date value as three parts: year, month, and day. The range for the parts is:
   • Year: 1 to 9999
   • Month: 1 to 12
   • Day: Lower limit is 1; the upper limit depends on the month and the year

TIME
Type TIME stores a time value as four parts: hours, minutes, seconds, and milliseconds. The range for the parts is:
   • Hours: 0 to 23
   • Minutes: 0 to 59
   • Seconds: 0 to 59
   • Milliseconds: 0 to 999

TIMESTAMP
Type TIMESTAMP combines the parts of DATE and TIME.

4.3.5 Bit String Data Types

Syntax

bit_string_data_type ::
   BIT
   | BINARY [(length)]
   | VARBINARY [(length)]
   | LVARBINARY | LONG VARBINARY

Arguments

BIT
Type BIT corresponds to a single bit value of 0 or 1.

SQL statements can assign and compare values in BIT columns to and from columns of types CHAR, VARCHAR, BINARY, VARBINARY, TINYINT, SMALLINT, and
INTEGER. However, in assignments from BINARY, VARBINARY, and LONG
VARBINARY, the value of the first four bits must be 0001 or 0000.

No arithmetic operations are allowed on BIT columns.

**BINARY \([\text{[length]}]\)**

Type BINARY corresponds to a bit field of the specified length of bytes. The default
length is 1 byte. The maximum length is 2000 bytes.

In interactive SQL, INSERT statements must use a special format to store values in
BINARY columns. They can specify the binary values as a bit string, hexadecimal
string, or character string. INSERT statements must enclose binary values in single-
quote marks, preceded by `b` for a bit string and `x` for a hexadecimal string:

<table>
<thead>
<tr>
<th>Prefix</th>
<th>Suffix</th>
<th>Example (for same 2 byte data)</th>
</tr>
</thead>
<tbody>
<tr>
<td>bit string</td>
<td><code>b</code></td>
<td><code>b'1010110100010000'</code></td>
</tr>
<tr>
<td>hex string</td>
<td><code>x</code></td>
<td><code>x'ad10'</code></td>
</tr>
<tr>
<td>char string</td>
<td></td>
<td><code>'ad10'</code></td>
</tr>
</tbody>
</table>

SQL interprets a character string as the character representation of a hexadecimal string.

If the data inserted into a BINARY column is less than the length specified, SQL pads it
with zeroes.

BINARY data can be assigned and compared to and from columns of type BIT, CHAR,
and VARBINARY types. No arithmetic operations are allowed.

**VARBINARY \([\text{[length]}]\)**

Type VARBINARY corresponds to a variable-length bit field with the maximum length
specified. The default length is 1 and the maximum length is 32752. Otherwise,
VARBINARY columns have the same characteristics as BINARY.

**LVARBINARY | LONG VARBINARY**

Type LONG VARBINARY corresponds to an arbitrarily-long bit field with the maximum
length defined by the underlying storage system.

The arbitrary size and unstructured nature of LONG data types restrict where they can
be used.

- LONG columns are allowed in select lists of query expressions and in INSERT
  statements.
- INSERT statements can store data from columns of any type into a LONG
  VARCHAR column, but LONG VARCHAR data cannot be stored in any other type.
- CONTAINS predicates are the only predicates that allow LONG columns (and then
  only if the underlying storage system explicitly supports CONTAINS predicates).
- Conditional expressions, arithmetic expressions, and functions cannot specify
  LONG columns.
- UPDATE statements cannot specify LONG columns.
4.4 Query Expressions

Description

A query expression selects the specified column values from one or more rows contained in one or more tables specified in the FROM clause. The selection of rows is restricted by a search condition in the WHERE clause. The temporary table derived through the clauses of a select statement is called a **result table**.

Query expressions form the basis of other SQL statements and syntax elements:

- SELECT statements are query expressions with optional ORDER BY and FOR UPDATE clauses.
- CREATE VIEW statements specify their result table as a query expression.
- INSERT statements can specify a query expression to add the rows of the result table to a table.
- UPDATE statements can specify a query expression that returns a single row to modify columns of a row.
- Some search conditions can specify query expressions. Basic predicates can specify query expressions, but the result table can contain only a single value. Quantified and IN predicates can specify query expressions, but the result table can contain only a single column.
- The FROM clause of a query expression can itself specify a query expression, called a **derived table**.

Syntax

```
query_expression ::
  query_specification
|   query_expression set_operator query_expression
|   ( query_expression )

set_operator ::
  { UNION [ ALL ] | INTERSECT | MINUS }

query_specification ::
  SELECT [ALL | DISTINCT]
  |
  *
|   | { table_name | alias } . * [, { table_name | alias } . * ] ... |
FROM table_ref [ { dharma ORDERED } ] [ , table_ref [ { dharma ORDERED } ] ] ... 
| [ WHERE search_condition ]
| [ GROUP BY [table.]column_name [ COLLATE collation-name ] ]
| [ , [table.]column_name [ COLLATE collation-name ] ] ... 
| [ HAVING search_condition ]
```
Overview

table_ref ::
  table_name [ AS ] [ alias [ ( column_alias [ , ... ] ) ] ]
| ( query_expression ) [ AS ] alias [ ( column_alias [ , ... ] ) ]
| [ ( ] joined_table [ ) ]

joined_table ::
  table_ref CROSS JOIN table_ref
| table_ref [ INNER | LEFT OUTER ] JOIN table_ref ON search_condition

Arguments

SELECT [ ALL | DISTINCT ]
DISTINCT specifies that the result table omits duplicate rows. ALL is the default, and specifies that the result table includes all rows.

SELECT * | { table_name | alias } . *
Specifies that the result table includes all columns from all tables named in the FROM clause. For instance, the following examples both specify all the columns in the customers table:

SELECT * FROM customers;
SELECT customers.* FROM customers;

The tablename.* syntax is useful when the select list refers to columns in multiple tables, and you want to specify all the columns in one of those tables:

SELECT CUSTOMERS.CUSTOMER_ID, CUSTOMERS.CUSTOMER_NAME, ORDERS.*
FROM CUSTOMERS, ORDERS;

SELECT expr [[ AS ]] [ ' ] column_title [ ' ] ]
Specifies a list of expressions, called a select list, whose results will form columns of the result table. Typically, the expression is a column name from a table named in the FROM clause. The expression can also be any supported mathematical expression, scalar function, or aggregate function that returns a value.

The optional 'column_title' argument specifies a new heading for the associated column in the result table. Enclose the new title in single or double quotation marks if it contains spaces or other special characters:

SELECT order_value, order_value * .2 AS 'order "markup"' FROM orders;

| ORDER_VALUE | ORDER "MARKUP"
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>5000000.00</td>
<td>1000000.00</td>
</tr>
<tr>
<td>110000.00</td>
<td>22000.00</td>
</tr>
<tr>
<td>3300000.00</td>
<td>660000.00</td>
</tr>
</tbody>
</table>

You can qualify column names with the name of the table they belong to:

SELECT CUSTOMER.CUSTOMER_ID FROM CUSTOMERS

You must qualify a column name if it occurs in more than one table specified in the FROM clause:

SELECT CUSTOMERS.CUSTOMER_ID
FROM CUSTOMERS, ORDERS

Qualified column names are always allowed even when they are not required.
FROM table_ref ...
The FROM clause specifies one or more table references. Each table reference resolves to one table (either a table stored in the database or a virtual table resulting from processing the table reference) whose rows the query expression uses to create the result table.

There are three forms of table references:

- A direct reference to a table, view or synonym
- A **derived table** specified by a query expression in the FROM clause
- A joined table that combines rows and columns from multiple tables

The usage notes specific to each form of table reference follow.

If there are multiple table references, SQL joins the tables to form an intermediate result table that is used as the basis for evaluating all other clauses in the query expression. That intermediate result table is the **Cartesian product** of rows in the tables in the FROM clause, formed by concatenating every row of every table with all other rows in all tables.

```
FROM table_name [ AS ] [ alias [ ( column_alias [ , ... ] ) ] ]
```

Explicitly names a table. The name listed in the FROM clause can be a table name, a view name, or a synonym.

The **alias** is a name you use to qualify column names in other parts of the query expression. Aliases are also called **correlation names**.

If you specify an alias, you must use it, and not the table name, to qualify column names that refer to the table. Query expressions that join a table with itself must use aliases to distinguish between references to column names.

For example, the following query expression joins the table customer with itself. It uses the aliases `x` and `y` and returns information on customers in the same city as customer SMITH:

```
SELECT y.cust_no, y.name
FROM customer x, customer y
WHERE x.name = 'SMITH'
    AND y.city = x.city ;
```

Similar to table aliases, the **column_alias** provides an alternative name to use in column references elsewhere in the query expression. If you specify column aliases, you must specify them for all the columns in table_name. Also, if you specify column aliases in the FROM clause, you must use them—not the column names—in references to the columns.

```
FROM ( query_expression ) [ AS ] alias [ ( column_alias [ , ... ] ) ]
```

Specifies a derived table through a query expression. With derived tables, you must specify an alias to identify the derived table.

Derived tables can also specify column aliases. Column aliases provides an alternative name to use in column references elsewhere in the query expression. If you specify column aliases, you must specify them for all the columns in the result table of the query expression. Also, if you specify column aliases in the FROM clause, you must use them, and not the column names, in references to the columns.
FROM ( () joined_table () )
Combines data from two table references by specifying a join condition. The syntax currently allowed in the FROM clause supports only a subset of possible join conditions:

- CROSS JOIN specifies a Cartesian product of rows in the two tables
- INNER JOIN specifies an inner join using the supplied search condition
- LEFT OUTER JOIN specifies a left outer join using the supplied search condition

You can also specify these and other join conditions in the WHERE clause of a query expression. See "Inner Joins" and "Outer Joins" for more detail on both ways of specifying joins.

{ dharma ORDERED }
Directs the SQL engine optimizer to join the tables in the order specified. Use this clause when you want to override the SQL engine's join-order optimization. This is useful for special cases when you know that a particular join order will result in the best performance from the underlying storage system. Since this clause bypasses join-order optimization, carefully test queries that use it to make sure the specified join order is faster than relying on the optimizer.

Note that the braces ({ and }) are part of the required syntax and not syntax conventions.

```sql
SELECT sc.tbl 'Table', sc.col 'Column',
       sc.coltype 'Data Type', sc.width 'Size'
FROM systpe.syscolumns sc, systpe.systables st
   { dharma ORDERED }
WHERE sc.tbl = st.tbl AND st.tbltype = 'S'
ORDER BY sc.tbl, sc.col;
```

WHERE search_condition
The WHERE clause specifies a search_condition that applies conditions to restrict the number of rows in the result table. If the query expression does not specify a WHERE clause, the result table includes all the rows of the specified table reference in the FROM clause.

The search_condition is applied to each row of the result table set of the FROM clause. Only rows that satisfy the conditions become part of the result table. If the result of the search_condition is NULL for a row, the row is not selected.

Search conditions can specify different conditions for joining two or more tables. See "Inner Joins" on page 4-20 and "Outer Joins" on page 4-23 for more details.

See “Search Conditions” on page 4-24 for details on the different kinds of search conditions.

```sql
SELECT *
FROM customer
WHERE city = 'BURLINGTON' AND state = 'MA' ;

SELECT *
FROM customer
WHERE city IN (  
    SELECT city  
    FROM customer  
    WHERE name = 'SMITH' ) ;
```
GROUP BY column_name ...
Specifies grouping of rows in the result table:

- For the first column specified in the GROUP BY clause, SQL arranges rows of the result table into groups whose rows all have the same values for the specified column.

- If a second GROUP BY column is specified, SQL then groups rows in each main group by values of the second column.

- SQL groups rows for values in additional GROUP BY columns in a similar fashion.

All columns named in the GROUP BY clause must also be in the select list of the query expression. Conversely, columns in the select list must also be in the GROUP BY clause or be part of an aggregate function.

If column_name refers to a character column, the column reference can include an optional COLLATE clause. The COLLATE clause specifies a collation sequence supported by the underlying storage system. (See "Specifying the Character Set for Character Data Types" on page 4-6 for notes on character sets and collations. See the documentation for your underlying storage system for details on any supported collations.)

HAVING search_condition
The HAVING clause allows conditions to be set on the groups returned by the GROUP BY clause. If the HAVING clause is used without the GROUP BY clause, the implicit group against which the search condition is evaluated is all the rows returned by the WHERE clause.

A condition of the HAVING clause can compare one aggregate function value with another aggregate function value or a constant.

-- select customer number and number of orders for all customers who had more than 10 orders prior to March 31st, 1991.

SELECT  cust_no, count(*)
FROM  orders
WHERE order_date < to_date ('3/31/1991')
GROUP BY cust_no
HAVING count (*) > 10 ;

UNION [ALL]
Appends the result table from one query expression to the result table from another.

The two query expressions must have the same number of columns in their result table, and those columns must have the same or compatible data types.

The final result table contains the rows from the second query expression appended to the rows from the first. By default, the result table does not contain any duplicate rows from the second query expression. Specify UNION ALL to include duplicate rows in the result table.

-- Get a merged list of customers and suppliers.
SQL Language Elements

Overview

SELECT name, street, state, zip
FROM customer
UNION
SELECT name, street, state, zip
FROM supplier;

-- Get a list of customers and suppliers
-- with duplicate entries for those customers who are
-- also suppliers.
SELECT name, street, state, zip
FROM customer
UNION ALL
SELECT name, street, state, zip
FROM supplier;

INTERSECT
Limits rows in the final result table to those that exist in the result tables from both query expressions.

The two query expressions must have the same number of columns in their result table, and those columns must have the same or compatible data types.

-- Get a list of customers who are also suppliers.
SELECT name, street, state, zip
FROM customer
INTERSECT
SELECT name, street, state, zip
FROM supplier;

MINUS
Limits rows in the final result table to those that exist in the result table from the first query expression minus those that exist in the second. In other words, the MINUS operator returns rows that exist in the result table from the first query expression but that do not exist in the second.

The two query expressions must have the same number of columns in their result table, and those columns must have the same or compatible data types.

-- Get a list of suppliers who are not customers.
SELECT name, street, state, zip
FROM supplier
MINUS
SELECT name, street, state, zip
FROM customer;

Authorization

The user executing a query expression must have any of the following privileges:

- DBA privilege
- SELECT permission on all the tables/views referred to in the query_expression.
4.4.1 Inner Joins

Description

Inner joins specify how the rows from one table reference are to be joined with the rows of another table reference. Inner joins usually specify a search condition that limits the number of rows from each table reference that become part of the result table generated by the inner join operation.

If an inner join does not specify a search condition, the result table from the join operation is the Cartesian product of rows in the tables, formed by concatenating every row of one table with every row of the other table. Cartesian products (also called cross products or cross joins) are not practically useful, but SQL logically processes all join operations by first forming the Cartesian products of rows from tables participating in the join.

If specified, the search condition is applied to the Cartesian product of rows from the two tables. Only rows that satisfy the search condition become part of the result table generated by the join.

A query expression can specify inner joins in either its FROM clause or in its WHERE clause. For each formulation in the FROM clause, there is an equivalent syntax formulation in the WHERE clause. Currently, not all syntax specified by the SQL-92 standard is allowed in the FROM clause.

Syntax

from_clause_inner_join ::
    | FROM table_ref CROSS JOIN table_ref
    | FROM table_ref [ INNER ] JOIN table_ref ON search_condition

where_clause_inner_join ::
    FROM table_ref, table_ref WHERE search_condition

Arguments

FROM table_ref CROSS JOIN table_ref

Explicitly specifies that the join generates the Cartesian product of rows in the two table references. This syntax is equivalent to omitting the WHERE clause and a search condition. The following queries illustrate the results of a simple CROSS JOIN operation and an equivalent formulation that does not use the CROSS JOIN syntax:
SELECT * FROM T1; -- Contents of T1
C1    C2
--    --
10    15
20    25
2 records selected

SELECT * FROM T2; -- Contents of T2
C3    C4
--    --
10    BB
15    DD
2 records selected

SELECT * FROM T1 CROSS JOIN T2; -- Cartesian product
C1    C2    C3    C4
--    --    --    --
10    15    10    BB
10    15    15    DD
20    25    10    BB
20    25    15    DD
4 records selected

SELECT * FROM T1, T2; -- Different formulation, same results
C1    C2    C3    C4
--    --    --    --
10    15    10    BB
10    15    15    DD
20    25    10    BB
20    25    15    DD
4 records selected

FROM table_ref [ INNER ] JOIN table_ref ON search_condition
FROM table_ref, table_ref WHERE search_condition
These two equivalent syntax constructions both specify search_condition for restricting rows that will be in the result table generated by the join. In the first format, INNER is optional and has no effect. There is no difference between the WHERE form of inner joins and the JOIN ON form.

Equi-joins

An equi-join specifies that values in one table equal some corresponding column's values in the other:

-- For customers with orders, get their name and order info, :
SELECT customer.cust_no, customer.name, orders.order_no, orders.order_date
FROM customers INNER JOIN orders
    ON customer.cust_no = orders.cust_no;

-- Different formulation, same results:
SELECT customer.cust_no, customer.name, orders.order_no, orders.order_date
FROM customers, orders
    WHERE customer.cust_no = orders.cust_no;

Self joins

A self join, or auto join, joins a table with itself. If a WHERE clause specifies a self join, the FROM clause must use aliases to have two different references to the same table:
-- Get all the customers who are from the same city as customer SMITH:
SELECT y.cust_no, y.name
FROM customer AS x INNER JOIN customer AS y
ON x.name = 'SMITH' AND y.city = x.city;

-- Different formulation, same results:
SELECT y.cust_no, y.name
FROM customer x, customer y
WHERE x.name = 'SMITH' AND y.city = x.city;

4.4.2 Outer Joins

Description

An outer join between two tables returns more information than a corresponding inner join. An outer join returns a result table that contains all the rows from one of the tables even if there is no row in the other table that satisfies the join condition.

In a left outer join, the information from the table on the left is preserved: the result table contains all rows from the left table even if some rows do not have matching rows in the right table. Where there are no matching rows in the left table, SQL generates null values.

In a right outer join, the information from the table on the right is preserved: the result table contains all rows from the right table even if some rows do not have matching rows in the left table. Where there are no matching rows in the right table, SQL generates null values.

SQL supports two forms of syntax to support outer joins:

- In the WHERE clause of a query expression, specify the outer join operator (+) after the column name of the table for which rows will not be preserved in the result table. Both sides of an outer-join search condition in a WHERE clause must be simple column references. This syntax is similar to Oracle's SQL syntax, and allows both left and right outer joins.

- For left outer joins only, in the FROM clause, specify the LEFT OUTER JOIN clause between two table names, followed by a search condition. The search condition can contain only the join condition between the specified tables.

Dharma's SQL implementation does not support full (two-sided) outer joins.

Syntax

from_clause_inner_join ::
FROM table_ref LEFT OUTER JOIN table_ref ON search_condition

where_clause_inner_join ::
WHERE [table_name.]column (+) = [table_name.]column
| WHERE [table_name.]column = [table_name.]column (+)

Examples

The following example shows a left outer join. It displays all the customers with their orders. Even if there is not a corresponding row in the orders table for each row in the customer table, NULL values are displayed for the orders.order_no and orders.order_date columns.
SELECT customer.cust_no, customer.name, orders.order_no, orders.order_date
FROM customers, orders
WHERE customer.cust_no = orders.cust_no (+);

The following series of examples illustrates the outer join syntax:

SELECT * FROM T1; -- Contents of T1
C1  C2
--  --
10  15
20  25
2 records selected

SELECT * FROM T2; -- Contents of T2
C3  C4
--  --
10  BB
15  DD
2 records selected

-- Left outer join
SELECT * FROM T1 LEFT OUTER JOIN T2 ON T1.C1 = T2.C3;
C1  C2  C3  C4
--  --  --  --
10  15  10  BB
20  25
2 records selected

-- Left outer join: different formulation, same results
SELECT * FROM T1, T2 WHERE T1.C1 = T2.C3 (+);
C1  C2  C3  C4
--  --  --  --
10  15  10  BB
20  25
2 records selected

-- Right outer join
SELECT * FROM T1, T2 WHERE T1.C1 (+) = T2.C3;
C1  C2  C3  C4
--  --  --  --
10  15  10  BB
15  DD
2 records selected

4.5 Search Conditions

Description

A search condition specifies a condition that is true or false about a given row or group of rows. Query expressions and UPDATE statements can specify a search condition. The search condition restricts the number of rows in the result table for the query expression or UPDATE statement.

Search conditions contain one or more predicates. The predicates that can be part of a search condition are described in the following subsections.
**Syntax**

```
search_condition ::
    [NOT] predicate
    [ { AND | OR } { predicate | ( search_condition ) } ]
predicate ::
    basic_predicate
    | quantified_predicate
    | between_predicate
    | null_predicate
    | like_predicate
    | contains_predicate
    | exists_predicate
    | in_predicate
    | outer_join_predicate
```

**4.5.1 Logical Operators: OR, AND, NOT**

Logical operators combine multiple search conditions. SQL evaluates multiple search conditions in this order:

1. Search conditions enclosed in parentheses. If there are nested search conditions in parentheses, SQL evaluates the innermost search condition first.
2. Search conditions preceded by NOT
3. Search conditions combined by AND
4. Search conditions combined by OR

**Examples**

```
SELECT *
FROM customer
WHERE name = 'LEVIEN' OR name = 'SMITH';
SELECT *
FROM customer
WHERE city = 'PRINCETON' AND state = 'NJ';
SELECT *
FROM customer
WHERE NOT (name = 'LEVIEN' OR name = 'SMITH');
```

**4.5.2 Relational Operators**

Relational operators specify how SQL compares expressions in basic and quantified predicates.

**Syntax**

```
relop ::=
    =
    | <>
    | !=
    | ^=
    | <
    | <=
    | >
    | >=
```
Relational Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Predicate is:</th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>True if the two expressions are equal.</td>
</tr>
<tr>
<td>&lt;&gt;</td>
<td>!=</td>
</tr>
<tr>
<td>&lt;</td>
<td>True if the first expression is less than the second expression.</td>
</tr>
<tr>
<td>&lt;=</td>
<td>True if the first expression is less than or equal to the second expression.</td>
</tr>
<tr>
<td>&gt;</td>
<td>True if the first expression is greater than the second expression.</td>
</tr>
<tr>
<td>&gt;=</td>
<td>True if the first expression is greater than or equal to the second expression.</td>
</tr>
</tbody>
</table>

See "Basic Predicate" and "Quantified Predicate" for more information.

4.5.3 Basic Predicate

Description

A basic predicate compares two values using a relational operator (see "Relational Operators"). If a basic predicate specifies a query expression, then the query expression must return a single value. Basic predicates often specify an inner join. See "Inner Joins" for more detail.

If the value of any expression is null or the query_expression does not return any value, then the result of the predicate is set to false.

Basic predicates that compare two character expressions can include an optional COLLATE clause. The COLLATE clause specifies a collation sequence supported by the underlying storage system. (See "Specifying the Character Set for Character Data Types" on page 4-6 for notes on character sets and collations. See the documentation for your underlying storage system for details on any supported collations.)

Syntax

```
basic_predicate ::
    expr relop { expr | (query_expression) } [ COLLATE collation_name ]
```
When ALL is specified the predicate evaluates to true if the query_expression returns no values or the specified relationship is true for all the values returned.

When SOME or ANY is specified the predicate evaluates to true if the specified relationship is true for at least one value returned by the query_expression. There is no difference between the SOME and ANY keywords. The predicate evaluates to false if the query_expression returns no values or the specified relationship is false for all the values returned.

**Syntax**

\[
\text{quantified Predicate ::} \\
\text{expr relop} \ { \text{ALL | ANY | SOME} } \ (\text{query_expression})
\]

**Example**

\[
10 < \text{ANY} \ (\text{SELECT COUNT(\ast)} \\
\text{FROM order_tbl} \\
\text{GROUP BY custid})
\]

### 4.5.5 BETWEEN Predicate

**Description**

The BETWEEN predicate can be used to determine if a value is within a specified value range or not. The first expression specifies the lower bound of the range and the second expression specifies the upper bound of the range.

The predicate evaluates to true if the value is greater than or equal to the lower bound of the range, or less than or equal to the upper bound of the range.

**Syntax**

\[
\text{between Predicate ::} \\
\text{expr [ NOT ] BETWEEN expr AND expr}
\]

**Example**

\[
salary \text{ BETWEEN 2000.00 AND 10000.00}
\]
4.5.7 CONTAINS Predicate

Description

The SQL CONTAINS predicate is an extension to the SQL standard that allows storage systems to provide search capabilities on character and binary data. See the documentation for the underlying storage system for details on support, if any, for the CONTAINS predicate.

Syntax

column_name [ NOT ] CONTAINS 'string'

Notes

- column_name must be one of the following data types: CHARACTER, VARCHAR, LONG VARCHAR, BINARY, VARBINARY, or LONG VARBINARY.
- There must be an index defined for column_name, and the CREATE INDEX statement for the column must include the TYPE clause, and specify an index type that indicates to the underlying storage system that this index supports CONTAINS predicates. See the documentation for your storage system for the correct TYPE clause for indexes that support CONTAINS predicates.
- The format of the quoted string argument and the semantics of the CONTAINS predicate are defined by the underlying storage system.

4.5.8 LIKE Predicate

Description

The LIKE predicate searches for strings that have a certain pattern. The pattern is specified after the LIKE keyword in a string constant. The pattern can be specified by a string in which the underscore ( _ ) and percent sign ( % ) characters have special semantics.

The ESCAPE clause can be used to disable the special semantics given to characters ' _ ' and ' % '. The escape character specified must precede the special characters in order to disable their special semantics.

Syntax

like_predicate ::
   column_name [ NOT ] LIKE string_constant
   [ ESCAPE escape-character ]

Notes

- The column name specified in the LIKE predicate must refer to a character string column.
- A percent sign in the pattern matches zero or more characters of the column string.
- A underscore sign in the pattern matches any single character of the column string.

Examples

cust_name LIKE '%Computer%''
cust_name LIKE '___'
item_name LIKE '%\_%' ESCAPE '\'

In the first example, for all strings with the substring 'Computer', the predicate will evaluate to true. In the second example, for all strings which are exactly three characters long, the predicate will evaluate to true. In the third example, the backslash character '\_' has been specified as the escape character, which means that the special interpretation given to the character ' _ ' is disabled. The pattern will evaluate to TRUE if the column item_name has embedded underscore characters.

### 4.5.9 EXISTS Predicate

**Description**

The EXISTS predicate can be used to check for the existence of specific rows. The query_expression returns rows rather than values. The predicate evaluates to true if the number of rows returned by the query_expression is non zero.

**Syntax**

exists_predicate ::

```
EXISTS ( query_expression )
```

**Example**

```
EXISTS (SELECT * FROM order_tbl
         WHERE order_tbl.custid = :custid)
```

In this example, the predicate will evaluate to true if the specified customer has any orders.

### 4.5.10 IN Predicate

**Description**

The IN predicate can be used to compare a value with a set of values. If an IN predicate specifies a query expression, then the result table it returns can contain only a single column.

**Syntax**

```
in_predicate ::

  expr [ NOT ] IN { ( query_expression ) |
                   ( constant , constant [ , ... ] ) }
```

**Example**

```
address.state IN ('MA', 'NH')
```
4.5.11 Outer Join Predicate

Description

An outer join predicate specifies two tables and returns a result table that contains all of
the rows from one of the tables, even if there is no matching row in the other table. See
"Outer Joins" for more information.

Syntax

outer_join_predicate ::
    [ table_name. ] column = [ table_name. ] column (+)
    | [table_name. ] column (+) = [ table_name. ] column

4.6 Expressions

Description

An expression is a symbol or string of symbols used to represent or calculate a single
value in an SQL statement. When you specify an expression in a statement, SQL
retrieves or calculates the value represented by the expression and uses that value when
it executes the statement.

Expressions are also called scalar expressions or value expressions.

Syntax

expr ::
    [ { table_name | alias } . ] column-name
    | character-literal
    | numeric-literal
    | date-time-literal
    | aggregate-function
    | scalar-function
    | concatenated-char-expr
    | numeric-arith-expr
    | date-arith-expr
    | conditional-expr
    | ( expr )

Arguments

[ { table_name | alias } . ] column-name
A column in a table.

You can qualify column names with the name of the table they belong to:

SELECT CUSTOMER.CUSTOMER_ID FROM CUSTOMERS

You must qualify a column name if it occurs in more than one table specified in the
FROM clause:

SELECT CUSTOMER.CUSTOMER_ID
FROM CUSTOMERS, ORDERS

Qualified column names are always allowed even when they are not required.
You can also qualify column names with an alias. Aliases are also called correlation names.

The FROM clause of a query expression can specify an optional alias after the table name (see "Query Expressions" for details on query expressions). If you specify an alias, you must use it—not the table name—to qualify column names that refer to the table. Query expressions that join a table with itself must use aliases to distinguish between references to column names.

The following example shows a query expression that joins the table customer with itself. It uses the aliases x and y and returns information on customers in the same city as customer SMITH:

```
SELECT y.cust_no, y.name
FROM customer x, customer y
WHERE x.name = 'SMITH'
   AND y.city = x.city;
```

### 4.6.1 Concatenated Character Expressions

**Description**

The || concatenation operator (two vertical bars) concatenates the two character expressions it separates.

The concatenation operator is similar to the CONCAT scalar function (see page 4-56). However, the concatenation operator allows easy concatenation of more than two character expressions, while the CONCAT scalar function requires nesting.
Overview

Syntax

concatenated-char-expr ::
   { character-literal | character-expr } || { character-literal | character-expr }
   [ { character-literal | character-expr } || { character-literal | character-expr } ] [ ... ]

Arguments

character-literal
A character literal. See "Character String Literals" on page 4-36 for details on specifying character literals.

character-expr
Any expression that evaluates to a character string (see "Data Types" for details of character data types), including column names and scalar functions that return a character string.

Examples

ISQL> SELECT 'Today''s date is ' || TO_CHAR(SYSDATE) FROM SYSCALCTABLE;
TODAY'S DATE IS 08/17/1998
--------------------------
Today's date is 08/17/1998
1 record selected

4.6.2 Numeric Arithmetic Expressions

Description

Numeric arithmetic expressions compute a value using addition, subtraction, multiplication, and division operations on numeric literals and expressions that evaluate to any numeric data type.

Syntax

numeric-arith-expr ::
   [ + | - ] { numeric-literal | numeric-expr } [ { + | - | * | / } numeric-arith-expr ]

Arguments

[ + | - ]
Unary plus or minus operator

numeric-literal
A numeric literal. See "Numeric Literals" for details on specifying numeric literals.

numeric-expr
Any expression that evaluates to a numeric data type (see "Data Types" for details of numeric data types), including:

- Column names
- Subqueries that return a single value
- Aggregate functions
- CAST or CONVERT operations to numeric data types
- Other scalar functions that return a numeric data type
Expressions

\{ + | - | * | \}

Addition, subtraction, multiplication, or subtraction operator. SQL evaluates numeric arithmetic expressions in the following order:

- Unary plus or minus
- Expressions in parentheses
- Multiplication and division, from left to right
- Addition and subtraction, from left to right

4.6.3 Date Arithmetic Expressions

Description

Date arithmetic expressions compute the difference between date-time expressions in terms of days or milliseconds. SQL supports these forms of date arithmetic:

- Addition and subtraction of integers to and from date-time expressions
- Subtraction of a date-time expression from another

Syntax

date_arith_expr ::
    date_time_expr \{ + | - \} int_expr
|    date_time_expr - date_time_expr

Arguments

date_time_expr
An expression that returns a value of type DATE or TIME or TIMESTAMP. A single date-time expression can not mix data types. All elements of the expression must be the same data type.

Date-time expressions can contain date-time literals, but they must be converted to DATE or TIME using the CAST, CONVERT, or TO_DATE functions (see the following examples, and "CAST" and "CONVERT function (extension)").

int_expr
An expression that returns an integer value. SQL interprets the integer differently depending on the data type of the date-time expression:

- For DATE expressions, integers represent days
- For TIME expressions, integers represent milliseconds
- For TIMESTAMP expressions, integers represent milliseconds

Examples

The following example manipulates DATE values using date arithmetic. SQL interprets integers as days and returns date differences in units of days:

```
SELECT C1, C2, C1-C2 FROM DTEST
```

```
c1   c2   c1-c2
```
The following example manipulates TIME values using date arithmetic. SQL interprets integers as milliseconds and returns time differences in milliseconds:

```sql
select systime,
systime - 3000,
systime - cast ('15:28:01' as time)
from dtest;
```

```
systime systime-3000 systime-convert(time,15:28:01)
15:28:09 15:28:06 8000
```

### 4.6.4 Conditional Expressions

Conditional expressions are a subset of scalar functions that generate different results depending on the value of their arguments. They provide some of the flexibility of traditional programming constructs to allow expressions to return alternate results depending on the value of their arguments.

The following scalar functions provide support for conditional expressions. See the discussion for each function in "Scalar Functions" for details.

**CASE**

CASE is the most general conditional expression. It specifies a series of search conditions and associated expressions. SQL returns the value specified by the first expression whose associated search condition evaluates as true. If none of the expressions evaluate as true, the CASE expression returns a null value (or the value of some other default expression if the CASE expression includes the ELSE clause).

All the other conditional expressions can also be expressed as CASE expressions.

**DECODE**

DECODE provides a subset of the functionality of CASE that is compatible with Oracle SQL syntax. DECODE is not SQL-92 compatible.

**NULLIF**

NULLIF substitutes a null value for an expression if it is equal to a second expression.

**COALESCE**

COALESCE specifies a series of expressions. SQL returns the first expression whose value is not null. If all the expressions evaluate as null, COALESCE returns a null value.

**IFNULL**

IFNULL substitutes a specified value if an expression evaluates as null. If the expression is not null, IFNULL returns the value of the expression.
4.7 Literals

**Literals** are a type of expression that specify a constant value (they are also called **constants**). You can specify literals wherever SQL syntax allows expressions. Some SQL constructs allow literals but prohibit other forms of expressions.

There are three types of literals:

- **Numeric**
- **Character string**
- **Date-time**

The following sections discuss each type of literal.

4.7.1 Numeric Literals

A numeric literal is a string of digits that SQL interprets as a decimal number. SQL allows the string to be in a variety of formats, including scientific notation.

**Syntax**

```
[+|-][0-9][0-9]...[0-9][0-9]...[E|e][+|-][0-9][0-9][0-9][0-9]
```

**Examples**

The following are all valid numeric strings:

- 123
- 123.456
- -123.456
- 12.34E-04

4.7.2 Character String Literals

A character string literal is a string of characters enclosed in single quotation marks (').

To include a single quotation mark in a character-string literal, precede it with an additional single quotation mark. The following SQL examples show embedding quotation marks in character-string literals:

```sql
insert into quote values('unquoted literal');
insert into quote values('''single-quoted literal''');
insert into quote values("double-quoted literal");
insert into quote values('O''Hare');
select * from quote;
```

```
c1
unknown literal
'single-quoted literal'
"double-quoted literal"
O'Hare
```

To insert a character-string literal that spans multiple lines, enclose each line in single quotation marks. The following SQL examples shows this syntax, as well as embedding quotation marks in one of the lines:

```sql
insert into quote values('O''Hare
hyphenated literal
"double-quoted literal"

select * from quote;
```
Overview

```sql
insert into quote2 values ('Here''s a very long character string ' 'literal that will not fit on a single line.');
1 record inserted.
select * from quote2;
C1
--
Here's a very long character string literal that will not fit on a single line.
```

4.7.3 Date-Time Literals

SQL supports special formats for literals to be used in conjunction with date-time data types. Basic predicates and the VALUES clause of INSERT statements can specify date literals directly for comparison and insertion into tables. In other cases, you need to convert date literals to the appropriate date-time data type with the CAST, CONVERT, or TO_DATE scalar functions.

Enclose date-time literals in single quotation marks.

4.7.3.1 Date Literals

Date literals specify a day, month, and year. By default, SQL supports any of the following formats, enclosed in single quotation marks ('). Check with your administrator to see if the set of supported formats has been changed by setting the TPE_DFLT_DATE runtime variable.

Syntax

```sql
date-literal ::
   {d 'yyyy-mm-dd'}
   | mm-dd-yyyy
   | mm/dd/yyyy
   | yyyy-mm-dd
   | yyyy/mm/dd
   | dd-mon-yyyy
   | dd/mo YYYY
```

Arguments

```sql
{d 'yyyy-mm-dd'}
A date literal enclosed in an escape clause compatible with ODBC. Precede the literal string with an open brace ( { } ) and a lowercase d. End the literal with a close brace. For example:

```sql
INSERT INTO DTEST VALUES ({d '1994-05-07'})
```

If you use the ODBC escape clause, you must specify the date using the format yyyy-mm-dd.

```sql
dd
The day of month as a 1- or 2-digit number (in the range 01-31).
```

```sql
mm
The month value as a 1- or 2-digit number (in the range 01-12).
```
mon
The first 3 characters of the name of the month (in the range 'JAN' to 'DEC').

yyy
The year as 4-digit number. By default, SQL generates an invalid date string error if the year is specified as anything but 4 digits. Check with your administrator to see if this default behavior has been changed by setting the DH_Y2K_CUTOFF runtime variable.

Examples
The following SQL examples show some of the supported formats for date literals:

```sql
CREATE TABLE T2 (C1 DATE, C2 TIME);
INSERT INTO T2 (C1) VALUES('5/7/56');
INSERT INTO T2 (C1) VALUES('7/MAY/1956');
INSERT INTO T2 (C1) VALUES('1956/05/07');
INSERT INTO T2 (C1) VALUES({d '1956-05-07'});
INSERT INTO T2 (C1) VALUES('29-sep-1952');
SELECT C1 FROM T2;
```

c1
1956-05-07
1956-05-07
1956-05-07
1956-05-07
1952-09-29

4.7.3.2 Time Literals
Time literals specify an hour, minute, second, and millisecond, using the following format, enclosed in single quotation marks ('):

Syntax
```
time-literal ::
  {t 'hh:mi:ss'}
  hh:mi:ss[.mls]
```

Arguments
```
{t 'hh:mi:ss'}
A time literal enclosed in an escape clause compatible with ODBC. Precede the literal string with an open brace ( { } ) and a lowercase t. End the literal with a close brace. For example:

INSERT INTO TTEST VALUES ({t '23:22:12'})
```

If you use the ODBC escape clause, you must specify the time using the format hh:mi:ss.

hh
The hour value as a 1- or 2-digit number (in the range 00 to 23).

mi
The minute value as a 1- or 2-digit number (in the range 00 to 59).
ss
The seconds value as a 1- or 2-digit number (in the range 00 to 59).

mls
The milliseconds value as a 1- to 3-digit number (in the range 000 to 999).

Examples

The following SQL examples show some of the formats SQL will and will not accept for time literals:

```sql
INSERT INTO T2 (C2) VALUES('3');
error(-20234): Invalid time string
INSERT INTO T2 (C2) VALUES('8:30');
error(-20234): Invalid time string
INSERT INTO T2 (C2) VALUES('8:30:1');
INSERT INTO T2 (C2) VALUES('8:30:');
error(-20234): Invalid time string
INSERT INTO T2 (C2) VALUES('8:30:00');
INSERT INTO T2 (C2) VALUES('8:30:1:1');
INSERT INTO T2 (C2) VALUES('8:30:1:1'));
SELECT C2 FROM T2;
```

c2
08:30:01
08:30:00
08:30:01
08:30:01

4.7.3.3 Timestamp Literals

Timestamp literals specify a date and a time separated by a space, enclosed in single quotation marks ('):

Syntax

```
{ts 'yyyy-mm-dd hh:mi:ss'}

| date-literal time-literal |
```

Arguments

```
{ts 'yyyy-mm-dd hh:mi:ss'}
A timestamp literal enclosed in an escape clause compatible with ODBC. Precede the literal string with an open brace ( { ) and a lowercase ts. End the literal with a close brace. For example:

```sql
INSERT INTO DTES...
VALUES ((ts '1956-05-07 10:41:37'))
```

If you use the ODBC escape clause, you must specify the timestamp using the format yyyy-mm-dd hh:mi:ss.

date-literal
A date literal.

time-literal
A time literal.
4.8 Date-Time Format Strings

The TO_CHAR scalar function supports a variety of format strings to control the output of date and time values. The format strings consist of keywords that SQL interprets and replaces with formatted values.

The format strings are case sensitive. For instance, SQL replaces 'DAY' with all uppercase letters, but follows the case of 'Day'.

Supply the format strings, enclosed in single quotation marks, as the second argument to the TO_CHAR function. For example:

```
SELECT C1 FROM T2;
C1
--
09/29/1952
1 record selected
SELECT TO_CHAR(C1, 'Day, Month ddth'),
     TO_CHAR(C2, 'HH12 a.m.') FROM T2;
TO_CHAR(C1,Day, Month DDTH)  TO_CHAR(C2,HH12 A.M.)
----------------------------  ---------------------
Monday   , September 29th    02 p.m.
1 record selected
```

For details of the TO_CHAR function, see "TO_CHAR" on page 4-96.

4.8.1 Date Format Strings

A date format string can contain any of the following format keywords along with other characters. The format keywords in the format string are replaced by corresponding values to get the result. The other characters are displayed as literals.

- **CC**: The century as a 2-digit number.
- **YYYY**: The year as a 4-digit number.
- **YYY**: The last 3 digits of the year.
- **YY**: The last 2 digits of the year.
- **Y**: The last digit of the year.
- **Y,YYY**: The year as a 4-digit number with a comma after the first digit.
- **Q**: The quarter of the year as 1-digit number (with values 1, 2, 3, or 4).
- **MM**: The month value as 2-digit number (in the range 01-12).
MONTH The name of the month as a string of 9 characters ('JANUARY' to 'DECEMBER').

MON The first 3 characters of the name of the month (in the range 'JAN' to 'DEC').

WW The week of year as a 2-digit number (in the range 01-52).

W The week of month as a 1-digit number (in the range 1-5).

DDD The day of year as a 3-digit number (in the range 001-365).

DD The day of month as a 2-digit number (in the range 01-31).

D The day of week as a 1-digit number (in the range 1-7, 1 for Sunday and 7 for Saturday).

DAY The day of week as a 9 character string (in the range 'SUNDAY' to 'SATURDAY').

DY The day of week as a 3 character string (in the range 'SUN' to 'SAT').

J The Julian day (number of days since DEC 31, 1899) as an 8 digit number.

TH When added to a format keyword that results in a number, this format keyword ('TH') is replaced by the string 'ST', 'ND', 'RD' or 'TH' depending on the last digit of the number.

Example:

```
SELECT C1 FROM T2;
C1
--
09/29/1952
1 record selected
SELECT TO_CHAR(C1, 'Day, Month ddth'),
     TO_CHAR(C2, 'HH12 a.m.') FROM T2;
TO_CHAR(C1, DAY, MONTH DDTH)  TO_CHAR(C2, HH12 A.M.)
---------------------------  ---------------------
Monday   , September 29th    02 p.m.
1 record selected
```

4.8.2 Time Format Strings

A time format string can contain any of the following format keywords along with other characters. The format keywords in the format string are replaced by corresponding values to get the result. The other characters are displayed as literals.

AM The string 'AM' or 'PM' depending on whether time...
PM corresponds to forenoon or afternoon.
A.M. The string 'A.M.' or 'P.M.' depending on whether time
  corresponds to forenoon or afternoon.
P.M.
HH12 The hour value as a 2-digit number (in the range 00 to
  11).
HH HH24 The hour value as a 2-digit number (in the range 00 to
  23).
MI The minute value as a 2-digit number (in the range 00 to
  59).
SS The seconds value as a 2-digit number (in the range 00 to
  59).
SSSSS The seconds from midnight as a 5-digit number (in the
  range 00000 to 86399).
MLS The milliseconds value as a 3-digit number (in the range
  000 to 999).

Example:

```sql
SELECT C1 FROM T2;
C1
--
09/29/1952
1 record selected
SELECT TO_CHAR(C1, 'Day, Month ddth'),
  TO_CHAR(C2, 'HH12 a.m.') FROM T2;
TO_CHAR(C1,DAY, MONTH DDTH)  TO_CHAR(C2,HH12 A.M.)
---------------------------  ---------------------
Monday   , September 29th    02 p.m.
1 record selected
```

4.9 Functions

Functions are a type of SQL expression that return a value based on the argument they
are supplied. SQL supports two types of functions:

- **Aggregate functions** calculate a single value for a collection of rows in a result table
  (if the function is in a statement with a GROUP BY clause, it returns a value for each
  group in the result table). Aggregate functions are also called set or statistical
  functions. Aggregate functions cannot be nested.

- **Scalar functions** calculate a value based on another single value. Scalar functions
  are also called value functions. Scalar functions can be nested.
4.9.1 Aggregate Functions

4.9.1.1 AVG

Syntax

```
AVG ( { [ALL] expression } | { DISTINCT column_ref } )
```

Description

The aggregate function AVG computes the average of a collection of values. The keyword DISTINCT specifies that the duplicate values are to be eliminated before computing the average.

- Null values are eliminated before the average value is computed. If all the values are null, the result is null.
- The argument to the function must be of type SMALLINT, INTEGER, NUMERIC, REAL or FLOAT.
- The result is of type NUMERIC.

Example

```
SELECT AVG (salary)
FROM employee
WHERE deptno = 20 ;
```

4.9.1.2 COUNT

Syntax

```
COUNT ( { [ALL] expression } | { DISTINCT column_ref } | * )
```

Description

The aggregate function COUNT computes either the number of rows in a group of rows or the number of non-null values in a group of values.

- The keyword DISTINCT specifies that the duplicate values are to be eliminated before computing the count.
- If the argument to COUNT function is '*', then the function computes the count of the number of rows in group.
- If the argument to COUNT function is not '*', then null values are eliminated before the number of rows is computed.
- The argument column_ref or expression can be of any type.
- The result of the function is of INTEGER type. The result is never null.

Example

```
SELECT COUNT (*)
FROM orders
WHERE order_date = SYSDATE ;
```
4.9.1.3 MAX

Syntax

```sql
MAX ( { [ALL] expression } | { DISTINCT column_ref } )
```

Description

The aggregate function MAX returns the maximum value in a group of values.

- The specification of DISTINCT has no effect on the result.
- The argument column_ref or expression can be of any type.
- The result of the function is of the same data type as that of the argument.
- The result is null if the result set is empty or contains only null values.

Example

```sql
SELECT order_date, product, MAX (qty)
FROM orders
GROUP BY order_date, product ;
```

4.9.1.4 MIN

Syntax

```sql
MIN ( { [ALL] expression } | { DISTINCT column_ref } )
```

Description

The aggregate function MIN returns the minimum value in a group of values.

- The specification of DISTINCT has no effect on the result.
- The argument column_ref or expression can be of any type.
- The result of the function is of the same data type as that of the argument.
- The result is null if the result set is empty or contains only null values.

Example

```sql
SELECT MIN (salary)
FROM employee
WHERE deptno = 20 ;
```

4.9.1.5 SUM

Syntax

```sql
SUM ( { [ALL] expression } | { DISTINCT column_ref } )
```

Description

The aggregate function SUM returns the sum of the values in a group. The keyword DISTINCT specifies that the duplicate values are to be eliminated before computing the sum.

- The argument column_ref or expression can be of any type.
Overview

- The result of the function is of the same data type as that of the argument except that the result is of type INTEGER when the argument is of type SMALLINT or TINYINT.
- The result can have a null value.

Example

```sql
SELECT SUM (amount)
FROM orders
WHERE order_date = SYSDATE ;
```

4.9.2 Scalar Functions

4.9.2.1 ABS function (ODBC compatible)

Syntax

```sql
ABS ( expression )
```

Description

The scalar function ABS computes the absolute value of expression.

Example

```sql
SELECT  ABS (MONTHS_BETWEEN (SYSDATE, order_date))
FROM  orders
WHERE   ABS (MONTHS_BETWEEN (SYSDATE, order_date)) > 3 ;
```

Notes

- The argument to the function must be of type TINYINT, SMALLINT, INTEGER, NUMERIC, REAL or FLOAT.
- The result is of type NUMERIC.
- If the argument expression evaluates to null, the result is null.

4.9.2.2 ACOS function (ODBC compatible)

Syntax

```sql
ACOS ( expression )
```

Description

The scalar function ACOS returns the arccosine of expression.
**Example**

```sql
select acos (.5) 'Arccosine in radians' from syscalctable;
```

```
ARCCOSINE IN RAD
-------------
1.047197551196598
1 record selected
```

```sql
select acos (.5) * (180/ pi()) 'Arccosine in degrees' from syscalctable;
```

```
ARCCOSINE IN DEG
-------------
59.999999999999993
1 record selected
```

**Notes**

ACOS takes the ratio (expression) of two sides of a right triangle and returns the corresponding angle. The ratio is the length of the side adjacent to the angle divided by the length of the hypotenuse.

The result is expressed in radians and is in the range -Pi/ 2 to Pi/ 2 radians. To convert degrees to radians, multiply degrees by Pi/ 180. To convert radians to degrees, multiply radians by 180/ Pi.

- expression must be in the range -1 to 1.
- expression must evaluate to an approximate numeric data type.

**4.9.2.3 ADD_MONTHS function (extension)**

**Syntax**

```sql
ADD_MONTHS ( date_expression, integer_expression )
```

**Description**

The scalar function ADD_MONTHS adds to the date value specified by the date_expression, the given number of months specified by integer_expression and returns the resultant date value.

**Example**

```sql
SELECT *
FROM   customer
WHERE  ADD_MONTHS (start_date, 6) > SYSDATE ;
```

**Notes**

- The first argument must be of DATE type.
- The second argument to the function must be of numeric type.
- The result is of type DATE.
- If any of the arguments evaluate to null, the result is null.
4.9.2.4 ASCII function (ODBC compatible)

Syntax

```
ASCII ( char_expression )
```

Description

The scalar function ASCII returns the ASCII value of the first character of the given character expression.

Example

```
SELECT ASCII ( zip )
FROM customer ;
```

Notes

- The argument to the function must be of type character.
- The result is of type INTEGER.
- If the argument char_expression evaluates to null, the result is null.

4.9.2.5 ASIN function (ODBC compatible)

Syntax

```
ASIN ( expression )
```

Description

The scalar function ASIN returns the arcsine of expression.

Example

```
select asin (1) * (180/ pi()) 'Arcsine in degrees' from syscalctable;
```

```
ARCSINE IN DEGREES
----------------
90.000000000000000
1 record selected
```

```
select asin (1) 'Arcsine in radians' from syscalctable;
```

```
ARCSINE IN RADIANS
----------------
1.570796326794897
1 record selected
```

Notes

ASIN takes the ratio (expression) of two sides of a right triangle and returns the corresponding angle. The ratio is the length of the side opposite the angle divided by the length of the hypotenuse.

The result is expressed in radians and is in the range -Pi/ 2 to Pi/ 2 radians. To convert degrees to radians, multiply degrees by Pi/ 180. To convert radians to degrees, multiply radians by 180/ Pi.

- expression must be in the range -1 to 1.
- expression must evaluate to an approximate numeric data type.
4.9.2.6 ATAN function (ODBC compatible)

Syntax

```
ATAN ( expression )
```

Description

The scalar function ATAN returns the arctangent of expression.

Example

```
select atan (1) * (180/ pi()) 'Arctangent in degrees' from syscalctable;
ARCTANGENT IN DE
----------------
45.000000000000000
1 record selected
select atan (1) 'Arctangent in radians' from syscalctable;
ARCTANGENT IN RA
----------------
0.785398163397448
1 record selected
```

Notes

ATAN takes the ratio (expression) of two sides of a right triangle and returns the corresponding angle. The ratio is the length of the side opposite the angle divided by the length of the side adjacent to the angle.

The result is expressed in radians and is in the range -Pi/2 to Pi/2 radians. To convert degrees to radians, multiply degrees by Pi/180. To convert radians to degrees, multiply radians by 180/Pi.

- expression must be in the range -1 to 1.
- expression must evaluate to an approximate numeric data type.

4.9.2.7 ATAN2 function (ODBC compatible)

Syntax

```
ATAN2 ( expression1 , expression2 )
```

Description

The scalar function ATAN2 returns the arctangent of the x and y coordinates specified by expression1 and expression2.

Example

```
select atan2 (1,1) * (180/ pi()) 'Arctangent in degrees' from syscalctable;
ARCTANGENT IN DE
----------------
45.000000000000000
1 record selected
select atan2 (1,1) 'Arctangent in radians' from syscalctable;
ARCTANGENT IN RA
----------------
0.785398163397448
1 record selected
```
ATAN2 takes the ratio of two sides of a right triangle and returns the corresponding angle. The ratio is the length of the side opposite the angle divided by the length of the side adjacent to the angle.

`expression1` and `expression2` specify the x and y coordinates of the end of the hypotenuse opposite the angle.

The result is expressed in radians and is in the range -Pi/2 to Pi/2 radians. To convert degrees to radians, multiply degrees by Pi/180. To convert radians to degrees, multiply radians by 180/Pi.

Both `expression1` and `expression2` must evaluate to approximate numeric data types.

### 4.9.2.8 CASE (SQL-92 Compatible)

**Syntax**

```sql
case-expr::
  searched-case-expr | simple-case-expr

searched-case-expr::
  CASE
    WHEN search_condition THEN { result-expr | NULL }
    [ ... ]
    [ ELSE expr | NULL ]
  END

simple-case-expr::
  CASE primary-expr
    WHEN expr THEN { result-expr | NULL }
    [ ... ]
    [ ELSE expr | NULL ]
  END
```

**Description**

The CASE scalar function is a type of conditional expression. (See the topic on Conditional Expressions for a summary of all the conditional expressions.)

The general form of the CASE scalar function specifies a series of search conditions and associated result expressions. It is called a searched case expression. SQL returns the value specified by the first result expression whose associated search condition evaluates as true. If none of the search conditions evaluate as true, the CASE expression returns a null value (or the value of some other default expression if the CASE expression includes the ELSE clause).

CASE also supports syntax for a shorthand notation, called a simple case expression, for evaluating whether one expression is equal to a series of other expressions.

**Notes**

- This function is not allowed in a GROUP BY clause
- Arguments to this function cannot be query expressions
Arguments

**CASE**
The `CASE` keyword alone, not followed by `primary-expr`, specifies a searched case expression. It must be followed by one or more `WHEN-THEN` clauses each that specify a search condition and corresponding expression.

```
WHEN search_condition THEN { result-expr | NULL }
```

*WHEN* clause for searched case expressions. SQL evaluates *search_condition*. If *search_condition* evaluates as true, `CASE` returns the value specified by *result-expr* (or null, if the clause specifies `THEN NULL`).

If *search_condition* evaluates as false, SQL evaluates the next `WHEN-THEN` clause, if any, or the `ELSE` clause, if it is specified.

**CASE primary-expr**
The `CASE` keyword followed by an expression specifies a simple case expression. In a simple case expression, one or more `WHEN-THEN` clauses specify two expressions.

A simple case expression can always be expressed as a searched case expression. Consider the following general simple case expression:

```
CASE primary-expr
  WHEN expr1 THEN result-expr1
  WHEN expr2 THEN result-expr2
  ELSE expr3
END
```

The preceding simple case expression is equivalent to the following searched case expression:

```
CASE
  WHEN primary-expr = expr1 THEN result-expr1
  WHEN primary-expr = expr2 THEN result-expr2
  ELSE expr3
END
```

```
WHEN expr THEN { result-expr | NULL }
```

*WHEN* clause for simple case expressions. SQL evaluates *expr* and compares it with *primary-expr* specified in the `CASE` clause. If they are equal, `CASE` returns the value specified by *result-expr* (or null, if the clause specifies `THEN NULL`).

If *expr* is not equal to *primary-expr*, SQL evaluates the next `WHEN-THEN` clause, if any, or the `ELSE` clause, if it is specified.

```
[ ELSE { expr | NULL } ]
```

In both searched case expressions and simple case expressions, the `ELSE` clause specifies an optional expression whose value SQL returns if none of the conditions specified in `WHEN-THEN` clauses were satisfied. If the `CASE` expression omits the `ELSE` clause, it is the same as specifying `ELSE NULL`.

**Examples**

The following example shows a searched case expression that assigns a label denoting tables as system tables if they begin with the letters `sys`. Note that this example can not be reformulated as a simple case expression, since it specifies a relational operator other than =.
SELECT tbl,
    CASE
        WHEN tbl like 'sys%' THEN 'System Table'
        ELSE 'Not System table'
    END
FROM systables;

<table>
<thead>
<tr>
<th>TBL</th>
<th>SEARCHED_CASE(TBLSYS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>systblspaces</td>
<td>System Table</td>
</tr>
<tr>
<td>systables</td>
<td>System Table</td>
</tr>
<tr>
<td>syscolumns</td>
<td>System Table</td>
</tr>
<tr>
<td>sysindexes</td>
<td>System Table</td>
</tr>
<tr>
<td>sysdbauth</td>
<td>System Table</td>
</tr>
<tr>
<td>systabauth</td>
<td>System Table</td>
</tr>
<tr>
<td>syscolauth</td>
<td>System Table</td>
</tr>
<tr>
<td>sysviews</td>
<td>System Table</td>
</tr>
<tr>
<td>syssynonymms</td>
<td>System Table</td>
</tr>
<tr>
<td>syddblinks</td>
<td>System Table</td>
</tr>
<tr>
<td>sys_keycol_usage</td>
<td>System Table</td>
</tr>
<tr>
<td>sys_ref_constrs</td>
<td>System Table</td>
</tr>
<tr>
<td>sys_chk_constrs</td>
<td>System Table</td>
</tr>
<tr>
<td>sys_tbl_constrs</td>
<td>System Table</td>
</tr>
<tr>
<td>sys_chkcol_usage</td>
<td>System Table</td>
</tr>
<tr>
<td>sysdatatypes</td>
<td>System Table</td>
</tr>
<tr>
<td>syscalctable</td>
<td>System Table</td>
</tr>
<tr>
<td>systblstat</td>
<td>System Table</td>
</tr>
</tbody>
</table>

The following example shows a searched case expression and an equivalent simple case expression.

- Searched case expression:
  SELECT tbl,
    CASE
        WHEN tbltype = 'S' THEN 'System Table'
        ELSE 'Not System table'
    END
  FROM systables;

- Equivalent simple case expression:
  SELECT tbl,
    CASE tbltype
        WHEN  'S' THEN 'System Table'
        ELSE 'Not System table'
    END
  FROM systables;

4.9.2.9 CAST function (SQL-92 compatible)

Syntax

CAST ( { expression | NULL } AS data_type [(length)] )

Description

The scalar function CAST converts an expression to another data type. The first argument is the expression to be converted. The second argument is the target data type.

The length option for the data_type argument specifies the length for conversions to CHAR and VARCHAR data types. If omitted, the default is 30 bytes.
If the expression evaluates to null, the result of the function is null. Specifying NULL with the CAST function is useful for set operations such as UNION that require two tables to have the same structure. CAST NULL allows you to specify a column of the correct data type so a table with a similar structure to another, but with fewer columns, can be in a union operation with the other table.

The CAST function provides a data-type-conversion mechanism compatible with the SQL-92 standard.

Use the CONVERT function, enclosed in the ODBC escape clause \{fn \}, to specify ODBC-compliant syntax for data type conversion. See the CONVERT (ODBC compatible) function for more information.

Example

The following SQL example uses CAST to convert an integer field from a catalog table to a character data type:

```sql
SELECT CAST(fld AS CHAR(25)), fld FROM systpe.syscalctable;
```

<table>
<thead>
<tr>
<th>CONVERT(CHARACTER(25),FLD)</th>
<th>FLD</th>
</tr>
</thead>
<tbody>
<tr>
<td>--------------------------</td>
<td>-----</td>
</tr>
<tr>
<td>100</td>
<td>100</td>
</tr>
</tbody>
</table>

1 record selected

4.9.2.10 CEILING function (ODBC compatible)

Syntax

`CEILING ( expression )`

Description

The scalar function CEILING returns the smallest integer greater than or equal to expression.

Example

```sql
SELECT CEILING (32.5) 'Ceiling' FROM SYSTPE.SYSCALCTABLE;
```

Notes

- `expression` must evaluate to a numeric data type.

4.9.2.11 CHAR function (ODBC compatible)

Syntax

`CHAR ( integer_expression )`

Description

The scalar function CHAR returns a character string with the first character having an ASCII value equal to the argument expression. CHAR is identical to CHR but provides ODBC-compatible syntax.
Example

```sql
SELECT *
FROM customer
WHERE SUBSTR(zip, 1, 1) = CHAR(53) ;
```

Notes

- The argument to the function must be of type INTEGER, TINYINT, or SMALLINT.
- The result is of type character.
- If the argument `integer_expression` evaluates to null, the result is null.

4.9.2.12 CHARTOROWID (extension)

Syntax

```
CHARTOROWID (char_expression)
```

Description

The scalar function CHARTOROWID returns a ROWID contained in the input argument in character form. The representation of a row identifier depends on the storage manager. The format of the `char_expression` argument to this function varies between storage managers.

Example

The following example shows the character-string format for a row identifier supplied as an argument to CHARTOROWID. In this example, the format for a row identifier is an integer (delimited as a character string by single quotes).

```sql
SELECT ROWID, FLD FROM SYSCALCTABLE;
ROWID   FLD
-----   ---
0       100
1 record selected
-- CHARTOROWID requires single quotes around its argument
SELECT * FROM SYSCALCTABLE WHERE ROWID = CHARTOROWID ('0');
    FLD
    ---
     100
1 record selected
```

Notes

- The argument to the function must be of type character.
- The result is of internal ROWID type as defined by the storage manager.
- If the argument `char_expression` evaluates to null, the result is null.
- The SQL statement execution returns error if the result of the input character expression does not contain a character string in the proper format for a row identifier, as defined by the storage manager.
4.9.2.13 CHR function (extension)

**Syntax**

```sql
CHR ( integer_expression )
```

**Description**

The scalar function CHR returns a character string with the first character having an ASCII value equal to the argument expression.

**Example**

```sql
SELECT *
FROM   customer
WHERE  SUBSTR (zip, 1, 1) = CHR (53) ;
```

**Notes**

- The argument to the function must be of type INTEGER, TINYINT, or SMALLINT.
- The result is of type character.
- If the argument `integer_expression` evaluates to null, the result is null.

4.9.2.14 COALESCE (SQL-92 compatible)

**Syntax**

```sql
COALESCE ( expression1, expression2 [ , ... ] )
```

**Description**

The COALESCE scalar function is a type of conditional expression. (See the topic on Conditional Expressions for a summary of all the conditional expressions.)

COALESCE specifies a series of expressions, and returns the first expression whose value is not null. If all the expressions evaluate as null, COALESCE returns a null value.

The COALESCE syntax is shorthand notation for a common case that can also be represented in a CASE expression. The following two formulations are equivalent:

```sql
COALESCE ( expression1 , expression2 , expression3 )
```

```sql
CASE
  WHEN expression1 IS NOT NULL THEN expression1
  WHEN expression2 IS NOT NULL THEN expression2
  ELSE expression3
```

**Example**

```sql
SELECT COALESCE(end_date, start_date) from job_hist;
```

**Notes**

- This function is not allowed in a GROUP BY clause
- Arguments to this function cannot be query expressions
4.9.2.15 CONCAT function (ODBC compatible)

Syntax

```
CONCAT ( char_expression , char_expression )
```

Description

The scalar function CONCAT returns a concatenated character string formed by concatenating argument one with argument two.

The CONCAT scalar function is similar to the concatenation operator (see page 4-32). However, the concatenation operator allows easy concatenation of more than two character expressions, while the CONCAT function requires nesting.

Example

```
SELECT name, empno, salary
FROM customer
WHERE project = CONCAT('US', proj_nam);
```

Notes

- Both the arguments must be of type CHARACTER or VARCHAR.
- The result is of type VARCHAR.
- If any of the argument expressions evaluates to null, the result is null.
- The trailing blanks for the first argument are removed.

4.9.2.17 CONVERT function (extension)

Syntax

```
CONVERT ( 'data_type[(length)]', expression )
```

Description

The scalar function CONVERT converts an expression to another data type. The first argument is the target data type. The second argument is the expression to be converted to that type.

The length option for the data_type argument specifies the length for conversions to CHAR and VARCHAR data types. If omitted, the default is 30 bytes.

If the expression evaluates to null, the result of the function is null.

The CONVERT function syntax is similar to but not compatible with the ODBC CONVERT function. Enclose the function in the ODBC escape clause `{fn }`, to specify ODBC-compliant syntax. See the CONVERT function (ODBC compatible) topic for more information.

Example

The following SQL example converts an integer field from a catalog table to a character string:
### 4.9.2.18 CONVERT function (ODBC compatible)

**Syntax**

```
{fn CONVERT (expression , data_type ) }
```

<table>
<thead>
<tr>
<th>data_type:</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQL_BIGINT</td>
</tr>
<tr>
<td>SQL_BINARY</td>
</tr>
<tr>
<td>SQL_BIT</td>
</tr>
<tr>
<td>SQL_CHAR</td>
</tr>
<tr>
<td>SQL_DATE</td>
</tr>
<tr>
<td>SQL_DECIMAL</td>
</tr>
<tr>
<td>SQL_DOUBLE</td>
</tr>
<tr>
<td>SQL_FLOAT</td>
</tr>
<tr>
<td>SQL_INTEGER</td>
</tr>
<tr>
<td>SQL_LONGVARBINARY</td>
</tr>
<tr>
<td>SQL_LONGVARCHAR</td>
</tr>
<tr>
<td>SQL_REAL</td>
</tr>
<tr>
<td>SQL_SMALLINT</td>
</tr>
<tr>
<td>SQL_TIME</td>
</tr>
<tr>
<td>SQL_TIMESTAMP</td>
</tr>
<tr>
<td>SQL_TINYINT</td>
</tr>
<tr>
<td>SQL_VARBINARY</td>
</tr>
<tr>
<td>SQL_VARCHAR</td>
</tr>
</tbody>
</table>

**Description**

The ODBC scalar function CONVERT converts an expression to another data type. The first argument is the expression to be converted. The second argument is the target data type.

If the expression evaluates to null, the result of the function is null.

The ODBC CONVERT function provides ODBC-compliant syntax for data type conversion. You must enclose the function with the ODBC escape clause `{fn }` to use ODBC-compliant syntax.

### 4.9.2.19 COS function (ODBC compatible)

**Syntax**

```
COS ( expression )
```
Overview

Description

The scalar function COS returns the cosine of expression.

Example

```
select cos(45 * pi()/180) 'Cosine of 45 degrees' from syscalctable;
```

<table>
<thead>
<tr>
<th>COSINE OF 45 DEG</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.707106781186548</td>
</tr>
</tbody>
</table>

1 record selected

Notes

COS takes an angle (expression) and returns the ratio of two sides of a right triangle. The ratio is the length of the side adjacent to the angle divided by the length of the hypotenuse.

- expression specifies an angle in radians
- expression must evaluate to an approximate numeric data type.

To convert degrees to radians, multiply degrees by Pi/ 180. To convert radians to degrees, multiply radians by 180/ Pi.

4.9.2.20 CURDATE function (ODBC compatible)

Syntax

```
CURDATE ()
```

Description

CURDATE returns the current date as a DATE value. This function takes no arguments.

SQL statements can refer to CURDATE anywhere they can refer to a DATE expression.

For example,

```
INSERT INTO objects (object_owner, object_id, create_date)
VALUES (USER, 1001, CURDATE()) ;
```

4.9.2.21 CURTIME function (ODBC compatible)

Syntax

```
CURTIME ()
```

Description

CURTIME returns the current time as a TIME value. This function takes no arguments.

SQL statements can refer to CURTIME anywhere they can refer to a TIME expression.

For example,

```
INSERT INTO objects (object_owner, object_id, create_time)
VALUES (USER, 1001, CURTIME()) ;
```
4.9.2.22 DATABASE (ODBC compatible)

Syntax

DATABASE [ ( ) ]

Description

The scalar function DATABASE returns the name of the database corresponding to the current connection name. This function takes no arguments, and the trailing parentheses are optional.

Example

```sql
select database() from t2;
```

DATABASE
--------
steel
1 record selected

4.9.2.23 DAYNAME function (ODBC compatible)

Syntax

DAYNAME ( date_expression )

Description

Returns a character string containing the name of the day (for example, Sunday, through Saturday) for the day portion of date_expression. The argument date_expression can be the name of a column, the result of another scalar function, or a date or timestamp literal.

Example

```sql
SELECT *
FROM orders
WHERE order_no = 342 and DAYNAME(order_date)='SATURDAY';
```

<table>
<thead>
<tr>
<th>ORDER_NO</th>
<th>ORDER_DATE</th>
<th>REFERENCE</th>
<th>CUST_NO</th>
</tr>
</thead>
<tbody>
<tr>
<td>342</td>
<td>08/10/1991</td>
<td>tdfg/101</td>
<td>10001</td>
</tr>
</tbody>
</table>

1 record selected

4.9.2.24 DAYOFMONTH function (ODBC compatible)

Syntax

DAYOFMONTH ( date_expression )

Description

The scalar function DAYOFMONTH returns the day of the month in the argument as a short integer value in the range of 1 - 31.

Example

```sql
SELECT *
FROM orders
WHERE DAYOFMONTH (order_date) = 14 ;
```
Notes

- The argument to the function must be of type DATE.
- The argument must be specified in the format MM/DD/YYYY.
- The result is of type SHORT.
- If the argument expression evaluates to null, the result is null.

4.9.2.25 DAYOFWEEK function (ODBC compatible)

Syntax

DAYOFWEEK ( date_expression )

Description

The scalar function DAYOFWEEK returns the day of the week in the argument as a short integer value in the range of 1 - 7.

Example

```
SELECT *
FROM orders
WHERE DAYOFWEEK (order_date) = 2;
```

Notes

- The argument to the function must be of type DATE.
- The argument must be specified in the format MM/DD/YYYY.
- The result is of type SHORT.
- If the argument expression evaluates to null, the result is null.

4.9.2.26 DAYOFYEAR function (ODBC compatible)

Syntax

DAYOFYEAR ( date_expression )

Description

The scalar function DAYOFYEAR returns the day of the year in the argument as a short integer value in the range of 1 - 366.

Example

```
SELECT *
FROM orders
WHERE DAYOFYEAR (order_date) = 300;
```

Notes

- The argument to the function must be of type DATE.
- The argument must be specified in the format MM/DD/YYYY.
• The result is of type SHORT.
• If the argument expression evaluates to null, the result is null.

4.9.2.27 DB_NAME (extension)

Syntax

\[
\text{DB\_NAME ( )}
\]

Description

The scalar function DB\_NAME returns the name of the database corresponding to the current connection name. It provides compatibility with the Sybase SQL Server function db\_name.

Example

```
SELECT DB\_NAME() FROM T2;
```
```
DB\_NAME
-------
dharmav4
1 record selected
```

4.9.2.28 DECODE function (extension)

Syntax

\[
\text{DECODE ( expression, search\_expression, match\_expression}
\]
\[
[ , search\_expression, match\_expression ...]
\]
\[
[ , default\_expression ]
\]

Description

The DECODE scalar function is a type of conditional expression. (See the topic on Conditional Expressions for a summary of all the conditional expressions.)

The scalar function DECODE compares the value of the first argument expression with each search\_expression and if a match is found, returns the corresponding match\_expression. If no match is found, then the function returns default\_expression. If default\_expression is not specified and no match is found, the function returns a null value.

DECODE provides a subset of the functionality of CASE that is compatible with Oracle SQL syntax. Use a simple case expression for SQL-compatible syntax (see CASE (SQL-92 Compatible)).

Example

```
SELECT ename, DECODE (deptno,
  10, 'ACCOUNTS    ',
  20, 'RESEARCH    ',
  30, 'SALES       ',
  40, 'SUPPORT     ',
  'NOT ASSIGNED'
) FROM employee ;
```
Notes

- The first argument expression can be of any type. The types of all search_expressions must be compatible with the type of the first argument.

- The match_expressions can be of any type. The types of all match_expressions must be compatible with the type of the first match_expression.

- The type of the default_expression must be compatible with the type of the first match_expression.

- The type of the result is the same as that of the first match_expression.

- If the first argument expression is null then the value of the default_expression is returned, if it is specified. Otherwise null is returned.

### 4.9.2.29 DEGREES function (ODBC compatible)

**Syntax**

DEGREES ( expression )

**Description**

The scalar function DEGREES returns the number of degrees in an angle specified in radians by expression.

**Example**

```sql
SELECT DEGREES(3.14159265359) 'Degrees in pi Radians'
FROM SYSTPE.SYSCALCTABLE;
```

**Notes**

- expression specifies an angle in radians
- expression must evaluate to a numeric data type.

### 4.9.2.30 DIFFERENCE function (ODBC compatible)

**Syntax**

DIFFERENCE ( string_exp1,string_exp2 )

**Description**

The scalar function DIFFERENCE returns an integer value that indicates the difference between the values returned by the SOUNDEX function for string_exp1 and string_exp2.

**Example**

```sql
SELECT DIFFERENCE(name,'Robets')
FROM customer
WHERE name = 'Roberts';
```

DIFFEREN

2
1 record selected
Notes

- The arguments of the function can be of the type fixed length or variable length CHARACTER.

- The result is INTEGER.

- If the argument expression evaluates to null, the result is null.

### 4.9.2.31 EXP function (ODBC compatible)

**Syntax**

```
EXP ( expression )
```

**Description**

The scalar function EXP returns the exponential value of expression (e raised to the power of expression).

**Example**

```
SELECT EXP( 4 ) 'e to the 4th power'
FROM SYSTPE.SYSCALCTABLE;
```

**Notes**

- expression must evaluate to an approximate numeric data type.

### 4.9.2.32 FLOOR function (ODBC compatible)

**Syntax**

```
FLOOR ( expression )
```

**Description**

The scalar function FLOOR returns the largest integer less than or equal to expression.

**Example**

```
SELECT FLOOR (32.5) 'Floor'
FROM SYSTPE.SYSCALCTABLE;
```

**Notes**

- expression must evaluate to a numeric data type.

### 4.9.2.33 GREATEST function (extension)

**Syntax**

```
GREATEST ( expression, expression, ... )
```

**Description**

The scalar function GREATEST returns the greatest value among the values of the given expressions.
Example

```sql
SELECT cust_no, name,
       GREATEST (ADD_MONTHS (start_date, 10), SYSDATE)
FROM customer;
```

Notes

- The first argument to the function can be of any type. The types of the subsequent arguments must be compatible with that of the first argument.
- The type of the result is the same as that of the first argument.
- If any of the argument expressions evaluates to null, the result is null.

4.9.2.34 HOUR function (ODBC compatible)

Syntax

```
HOUR (time_expression)
```

Description

The scalar function HOUR returns the hour in the argument as a short integer value in the range of 0 - 23.

Example

```sql
SELECT *
FROM arrivals
WHERE HOUR (in_time) < 12;
```

Notes

- The argument to the function must be of type TIME.
- The argument must be specified in the format hh:mm:ss.
- The result is of type SHORT.
- If the argument expression evaluates to null, the result is null.

4.9.2.35 IFNULL function (ODBC compatible)

Syntax

```
IFNULL(expr, value)
```

Description

The scalar function IFNULL returns value if expr is null. If expr is not null, IFNULL returns expr.

Example

```sql
select c1, ifnull(c1, 9999) from temp order by c1;
```
4.9.2.36 INITCAP function (extension)

Syntax

\[
\text{INITCAP} \ (\text{char\_expression})
\]

Description

The scalar function INITCAP returns the result of the argument character expression after converting the first character to upper case and the subsequent characters to lower case.

Example

\[
\text{SELECT INITCAP (name) FROM customer ;}
\]

Notes

- The argument to the function must be of type CHARACTER.
- The result is of type CHARACTER.
- If the argument expression evaluates to null, the result is null.

4.9.2.37 INSERT function (ODBC compatible)

Syntax

\[
\text{INSERT} (\text{string\_exp1}, \text{start}, \text{length}, \text{string\_exp2})
\]

Description

The scalar function INSERT returns a character string where length characters have been deleted from string\_exp1 beginning at start and string\_exp2 has been inserted into string\_exp1, beginning at start.

Example

\[
\text{SELECT INSERT(name,2,4,'xx') FROM customer}
\]

\[
\text{WHERE name = 'Goldman';}
\]

\[
\text{INSERT(NAME,2,4,XX)}
\]

\[
\text{Gxxan}
\]

1 record selected
Example

- The string_exp can be of the type fixed length or variable length CHARACTER.
- The start and length can be of the type INTEGER, SMALLINT, TINYINT or BIGINT.
- The result string is of the type string_exp1.
- If any of the argument expression evaluates to a null, the result would be a null.
- If start is negative or zero, the result string evaluates to a null.
- If length is negative, the result evaluates to a null.

4.9.2.38 INSTR function (extension)

Syntax

```
INSTR ( char_expression, char_expression [, start_position [, occurrence]])
```

Description

The scalar function INSTR searches for the character string corresponding to the second argument in the character string corresponding to the first argument starting at start_position. If occurrence is specified, then INSTR searches for the n_th occurrence where n is the value of the fourth argument. The position (with respect to the start of string corresponding to the first argument) is returned if a search is successful. Zero is returned if no match can be found.

Example

```
SELECT cust_no, name
FROM customer
WHERE INSTR (LOWER (addr), 'heritage') > 0 ;
```

Notes

- The first and second arguments must be of type CHARACTER.
- The third and fourth arguments, if specified, must be of type INTEGER.
- The values for specifying position in a character string starts from 1. That is, the very first character in a string is at position 1, the second character is at position 2 and so on.
- If the third argument is not specified, a default value of 1 is assumed.
- If the fourth argument is not specified, a default value of 1 is assumed.
- The result is of type INTEGER.
- If any of the argument expressions evaluates to null, the result is null.
4.9.2.39 LAST_DAY function (extension)

Syntax

LAST_DAY (date_expression)

Description

The scalar function LAST_DAY returns the date corresponding to the last day of the month containing the argument date.

Example

```sql
SELECT *
FROM orders
WHERE LAST_DAY (order_date) + 1 = '08/01/1991';
```

Notes

- The argument to the function must be of type DATE.
- The result is of type DATE.
- If the argument expression evaluates to null, the result is null.

4.9.2.40 LCASE function (ODBC compatible)

Syntax

LCASE (char_expression)

Description

The scalar function LCASE returns the result of the argument character expression after converting all the characters to lower case. LCASE is the same as LOWER but provides ODBC-compatible syntax.

Example

```sql
SELECT *
FROM customer
WHERE LCASE (name) = 'smith';
```

Notes

- The argument to the function must be of type CHARACTER.
- The result is of type CHARACTER.
- If the argument expression evaluates to null, the result is null.

4.9.2.41 LEAST function (extension)

Syntax

LEAST (expression, expression, ...)

Functions

SQL Language Elements 4–63
Overview

Description

The scalar function LEAST returns the lowest value among the values of the given expressions.

Example

```sql
SELECT cust_no, name, 
    LEAST (ADD_MONTHS (start_date, 10), SYSDATE) 
FROM  customer ;
```

Notes

- The first argument to the function can be of any type. The types of the subsequent arguments must be compatible with that of the first argument.
- The type of the result is the same as that of the first argument.
- If any of the argument expressions evaluates to null, the result is null.

4.9.2.42  LEFT function (ODBC compatible)

Syntax

```sql
LEFT ( string_exp, count )
```

Description

The scalar function LEFT returns the leftmost count of characters of string_exp.

Example

```sql
SELECT LEFT(name,4) 
FROM   customer  
WHERE  name = 'Goldman';
```

Notes

- The string_exp can be of the type fixed or variable length CHARACTER.
- The count can be of the type INTEGER, SMALLINT, BIGINT, or TINYINT.
- If any of the arguments of the expression evaluates to a null, the result would be null.
- If the count is negative, the result evaluates to a null.

4.9.2.43  LENGTH function (ODBC compatible)

Syntax

```sql
LENGTH ( char_expression )
```
**Description**

The scalar function `LENGTH` returns the string length of the value of the given character expression.

**Example**

```sql
SELECT name 'LONG NAME'
FROM customer
WHERE LENGTH (name) > 5 ;
```

**Notes**

- The argument to the function must be of type `CHARACTER` or `VARCHAR`.
- The result is of type `INTEGER`.
- If the argument expression evaluates to null, the result is null.

### 4.9.2.44 LOCATE function (ODBC compatible)

**Syntax**

`LOCATE( char-expr1 , char-expr2, [start-position] )`

**Description**

The scalar function `LOCATE` returns the location of the first occurrence of `char-expr1` in `char-expr2`. If the function includes the optional integer argument `start-position`, `LOCATE` begins searching `char-expr2` at that position. If the function omits the `start-position` argument, `LOCATE` begins its search at the beginning of `char-expr2`.

LOCATE denotes the first character position of a character expression as 1. If the search fails, LOCATE returns 0. If either character expression is null, LOCATE returns a null value.

**Example**

The following example uses two string literals as character expressions. LOCATE returns a value of 6:

```sql
SELECT  LOCATE('this', 'test this test', 1) FROM TEST;
LOCATE(THIS,
------------
6
1 record selected
```

### 4.9.2.45 LOG10 function (ODBC compatible)

**Syntax**

`LOG10 ( expression )`

**Description**

The scalar function `LOG10` returns the base 10 logarithm of `expression`. 
Overview

Example

```
SELECT LOG10 (100) 'Log base 10 of 100'
    FROM SYSTPE.SYSCALCTABLE;
```

Notes

- expression must evaluate to an approximate numeric data type.

4.9.2.46 LOWER function (SQL-92 compatible)

Syntax

```
LOWER ( char_expression )
```

Description

The scalar function LOWER returns the result of the argument character expression after converting all the characters to lower case.

Example

```
SELECT *
    FROM customer
    WHERE LOWER (name) = 'smith' ;
```

Notes

- The argument to the function must be of type CHARACTER.
- The result is of type CHARACTER.
- If the argument expression evaluates to null, the result is null.

4.9.2.47 LPAD function (extension)

Syntax

```
LPAD ( char_expression, length [ , pad_expression ] )
```

Description

The scalar function LPAD pads the character string corresponding to the first argument on the left with the character string corresponding to the third argument so that after the padding, the length of the result is length.

Example

```
SELECT LPAD (name, 30)
    FROM customer ;
SELECT LPAD (name, 30, '.')
    FROM customer ;
```

Notes

- The first argument to the function must be of type CHARACTER.
- The second argument to the function must be of type INTEGER.
- The third argument, if specified, must be of type CHARACTER.
If the third argument is not specified, the default value is a string of length 1 containing one blank.

If $L_1$ is the length of the first argument and $L_2$ is the value of the second argument, then:

- If $L_1$ is less than $L_2$, the number of characters padded is equal to $L_2 - L_1$.
- If $L_1$ is equal to $L_2$, no characters are padded and the result string is the same as the first argument.
- If $L_1$ is greater than $L_2$, the result string is equal to the first argument truncated to first $L_2$ characters.

The result is of type CHARACTER.

If the argument expression evaluates to null, the result is null.

### 4.9.2.48 LTRIM function (ODBC compatible)

**Syntax**

```
LTRIM ( char_expression [ , char_set ] )
```

**Description**

The scalar function LTRIM removes all the leading characters in char_expression, that are present in char_set and returns the resultant string. Thus, the first character in the result is guaranteed to be not in char_set. If the char_set argument is omitted, the function removes the leading and trailing blanks from char_expression.

**Example**

```
SELECT name, LTRIM (addr, ' ') FROM customer ;
```

**Notes**

- The first argument to the function must be of type CHARACTER.
- The second argument to the function must be of type CHARACTER.
- The result is of type CHARACTER.
- If the argument expression evaluates to null, the result is null.

### 4.9.2.49 MINUTE function (ODBC compatible)

**Syntax**

```
MINUTE ( time_expression )
```

**Description**

The scalar function MINUTE returns the minute value in the argument as a short integer in the range of 0 - 59.
Example

```
SELECT *
FROM   arrivals
WHERE  MINUTE (in_time) > 10 ;
```

Notes

- The argument to the function must be of type TIME.
- The argument must be specified in the format HH:MI:SS.
- The result is of type SHORT.
- If the argument expression evaluates to null, the result is null.

4.9.2.50 MOD function (ODBC compatible)

Syntax

```
MOD ( expression1, expression2 )
```

Description

The scalar function MOD returns the remainder of expression1 divided by expression2.

Example

```
SELECT MOD (11, 4) 'Modulus'
FROM SYSTPE.SYSCALCTABLE;
```

Notes

- Both expression1 and expression2 must evaluate to exact numeric data types.
- If expression2 evaluates to zero, MOD returns zero.

4.9.2.51 MONTHNAME function (ODBC compatible)

Syntax

```
MONTHNAME ( date_expression )
```

Description

Returns a character string containing the name of the month (for example, January, through December ) for the month portion of date_expression. Argument date_expression can be name of a column, the result of another scalar function, or a date or timestamp literal.

Example

```
SELECT * 
FROM   orders
WHERE  order_no =346 and MONTHNAME(order_date)='JUNE';
```

```
ORDER_NO  ORDER_DATE   REFERENCE              CUST_NO
----------  -----------   ---------------------  -------
 346    06/01/1991        87/rd                           10002
1 record selected
```
4.9.2.52 MONTH function (ODBC compatible)

Syntax

MONTH ( date_expression )

Description

The scalar function MONTH returns the month in the year specified by the argument as a short integer value in the range of 1 - 12.

Example

```
SELECT *
FROM orders
WHERE MONTH (order_date) = 6;
```

Notes

- The argument to the function must be of type DATE.
- The argument must be specified in the format MM/DD/YYYY.
- The result is of type SHORT.
- If the argument expression evaluates to null, the result is null.

4.9.2.53 MONTHS_BETWEEN function (extension)

Syntax

MONTHS_BETWEEN ( date_expression, date_expression )

Description

The scalar function MONTHS_BETWEEN computes the number of months between two date values corresponding to the first and second arguments.

Example

```
SELECT MONTHS_BETWEEN (SYSDATE, order_date)
FROM orders
WHERE order_no = 1002;
```

Notes

- The first and the second arguments to the function must be of type DATE.
- The result is of type INTEGER.
- The result is negative if the date corresponding to the second argument is greater than that corresponding to the first argument.
- If any of the arguments expression evaluates to null, the result is null.

4.9.2.54 NEXT_DAY function (extension)

Syntax

NEXT_DAY ( date_expression, day_of_week )
Overview

Description
The scalar function NEXT_DAY returns the minimum date that is greater than the date corresponding to the first argument for which the day of the week is same as that specified by the second argument.

Example
SELECT NEXT_DAY (order_date, 'MONDAY')
FROM orders;

Notes
- The first argument to the function must be of type DATE.
- The second argument to the function must be of type CHARACTER. The result of the second argument must be a valid day of week ('SUNDAY', 'MONDAY' etc.)
- The result is of type DATE.
- If any of the argument expressions evaluates to null, the result is null.

4.9.2.55 NOW function (ODBC compatible)
Syntax
NOW ( )
Description
NOW returns the current date and time as a TIMESTAMP value. This function takes no arguments.

4.9.2.56 NULLIF (SQL-92 compatible)
Syntax
NULLIF ( expression1, expression2 )
Description
The NULLIF scalar function is a type of conditional expression. (See the topic on Conditional Expressions for a summary of all the conditional expressions.)

The NULLIF scalar function returns a null value for expression1 if it is equal to expression2. It's useful for converting values to null from applications that use some other representation for missing or unknown data.

Notes
- This function is not allowed in a GROUP BY clause.
- Arguments to this function cannot be query expressions.
- The NULLIF expression is shorthand notation for a common case that can also be represented in a CASE expression, as follows:
CASE
  WHEN expression1 = expression2 THEN NULL
  ELSE expression1

Example
This example uses the NULLIF scalar function to insert a null value into an address column if the host-language variable contains a single space character.

  INSERT INTO employee (add1) VALUES (NULLIF (:address1, ' '));

4.9.2.57 NVL function (extension)

Syntax
NVL ( expression, expression )

Description
The scalar function NVL returns the value of the first expression if the first expression value is not null. If the first expression value is null, the value of the second expression is returned.

The NVL function is not ODBC compatible. Use the IFNULL function for ODBC-compatible syntax.

Example

  SELECT salary + NVL (comm, 0) 'TOTAL SALARY'
  FROM employee ;

Notes
- The first argument to the function can be of any type.
- The type of the second argument must be compatible with that of the first argument.
- The type of the result is the same as the first argument.

4.9.2.58 OBJECT_ID function (extension)

Syntax
OBJECT_ID ('table_name')

Description
The scalar function OBJECT_ID returns the value of the id column in the systpe.systables, plus one. This function provides compatibility with the Sybase SQL Server function object_id.

Arguments
- table_name
  The name of the table for which OBJECT_ID returns an identification value.
Example

```sql
select id, object_id(tbl), tbl from systpe.systables
1  where owner = 'systpe';
```

<table>
<thead>
<tr>
<th>ID</th>
<th>OBJECT_ID(TBL)</th>
<th>TBL</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>systblspaces</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>systables</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>syscolumns</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>sysindexes</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>systsfiles</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>syslogfiles</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
<td>syddbbackup</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>syslogbackup</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
<td>sysdbsyncpt</td>
</tr>
<tr>
<td>9</td>
<td>10</td>
<td>sysdbuuid</td>
</tr>
<tr>
<td>10</td>
<td>11</td>
<td>syssyssvr</td>
</tr>
<tr>
<td>11</td>
<td>12</td>
<td>syusuarsvr</td>
</tr>
</tbody>
</table>

4.9.2.59 PI function (ODBC compatible)

Syntax

```sql
PI ()
```

Description

The scalar function PI returns the constant value of pi as a floating point value.

Example

```sql
SELECT PI ()
FROM SYSTPE.SYSCALCTABLE;
```

4.9.2.60 POWER function (ODBC compatible)

Syntax

```sql
POWER ( expression1 , expression2 )
```

Description

The scalar function POWER returns `expression1` raised to the power of `expression2`.

Example

```sql
SELECT POWER (3, 2) '3 raised to the 2nd power'
FROM SYSTPE.SYSCALCTABLE;
```

Notes

- `expression1` must evaluate to a numeric data type.
- `expression2` must evaluate to an exact numeric data type.
4.9.2.61  PREFIX function (extension)

Syntax

PREFIX(char_expression, start_position, char_expression)

Description

The scalar function PREFIX returns the substring of a character string starting from the position specified by start position, and ending before the specified character.

Arguments

char_expression
An expression that evaluates to a character string, typically a character-string literal or column name. If the expression evaluates to null, PREFIX returns null.

start_position
An expression that evaluates to an integer value. PREFIX searches the string specified in the first argument starting at that position. A value of 1 indicates the first character of the string.

char_expression
An expression that evaluates to a single character. PREFIX returns the substring that ends before that character. If PREFIX does not find the character, it returns the substring beginning with start_position, to the end of the string. If the expression evaluates to more than one character, PREFIX ignores all but the first character.

Example

```
SELECT C1, C2, PREFIX(C1, 1, '.') FROM T1;
C1          C2  PREFIX(C1,1,.
--          --  -------------
test.pref   .   test
pref.test   s   pref
2 records selected
```

```
SELECT C1, C2, PREFIX(C1, 1, C2) FROM T1;
C1          C2  PREFIX(C1,1,C
--          --  -------------
test.pref   .   test
pref.test   s   pref.te
2 records selected
```

```
SELECT C1, C2, PREFIX(C1, 1, 'Q') FROM T1;
C1          C2  PREFIX(C1,1,Q
--          --  -------------
test.pref   .   test
pref.test   s   pref.test
2 records selected
```

4.9.2.62  QUARTER function (ODBC compatible)

Syntax

QUARTER ( time_expression )

Functions
Description

The scalar function QUARTER returns the quarter in the year specified by the argument as a short integer value in the range of 1 - 4.

Example

```
SELECT *
FROM orders
WHERE QUARTER (order_date) = 3 ;
```

Notes

- The argument to the function must be of type DATE.
- The argument must be specified in the format MM/DD/YYYY.
- The result is of type SHORT.
- If the argument expression evaluates to null, the result is null.

4.9.2.63 RADIANS function (ODBC compatible)

Syntax

```
RADIANS ( expression )
```

Description

The scalar function RADIANS returns the number of radians in an angle specified in degrees by expression.

Example

```
SELECT RADIANS(180) 'Radians in 180 degrees'
FROM SYSTPE.SYSCALCTABLE;
```

Notes

- expression specifies an angle in degrees
- expression must evaluate to a numeric data type.

4.9.2.64 RAND function (ODBC compatible)

Syntax

```
RAND ( [ expression ] )
```

Description

The scalar function RAND returns a randomly-generated number, using expression as an optional seed value.

Example

```
SELECT RAND(3) 'Random number using 3 as seed value'
FROM SYSTPE.SYSCALCTABLE;
```
Notes

- expression must evaluate to an exact numeric data type.

### 4.9.2.65 REPLACE function (ODBC compatible)

#### Syntax

```
REPLACE ( string_exp1, string_exp2, string_exp3 )
```

#### Description

The scalar function REPLACE replaces all occurrences of `string_exp2` in `string_exp1` with `string_exp3`.

#### Example

```
SELECT REPLACE ( name, 'mi', 'moo' )
FROM customer
WHERE name = 'Smith';
```

```
REPLACE(NAME,MI,MOO)
Smooth
1 record selected
```

#### Notes

- `string_exp` can be of the type fixed or variable length CHARACTER.
- If any of the arguments of the expression evaluates to null, the result is null.
- If the replacement string is not found in the search string, it returns the original string.

### 4.9.2.66 RIGHT function (ODBC compatible)

#### Syntax

```
RIGHT ( string_exp, count )
```

#### Description

The scalar function RIGHT returns the rightmost count of characters of `string_exp`.

#### Example

```
SELECT RIGHT(fld1, 6)
FROM test100
WHERE fld1 = 'Afghanistan';
```

```
RIGHT(FLD1,6)
nistan
1 record selected
```

#### Notes

- The `string_exp` can be of the type fixed or variable length CHARACTER.
- The `count` can be of the type INTEGER, SMALLINT, BIGINT, or TINYINT.
• If any of the arguments of the expression evaluates to a null, the result would be null.
• If the count is negative, the result evaluates to a null.

4.9.2.67 REPEAT function (ODBC compatible)

Syntax

REPEAT ( string_exp, count )

Description

The scalar function REPEAT returns a character string composed of string_exp repeated count times.

Example

SELECT REPEAT(fld1, 3) FROM test100 WHERE fld1 = 'Afghanistan'

Results

REPEAT(FLD1, 3)

AfghanistanAfghanistanAfghanistan

1 record selected

Notes

• The string exp. can be of the type fixed length or variable length CHARACTER.
• The count can be of the type INTEGER, SMALLINT, BIGINT, or TINYINT.
• If any of the arguments of the expression evaluates to a null, the result would be null.
• If the count is negative or zero, the result evaluates to a null.

4.9.2.68 ROWID (extension)

Syntax

ROWID

Description

ROWID returns the row identifier of the current row in a table. This function takes no arguments. The ROWID of a row is determined when the row is inserted into the table. Once assigned, the ROWID remains the same for the row until the row is deleted. At any given time, each row in a table is uniquely identified by its ROWID.

The format of the row identifier returned by this function varies between storage managers.

Selecting a row in a table using its ROWID is the most efficient way of selecting the row. For example,
SELECT *  
FROM customers  
WHERE ROWID = '10';

4.9.2.69  ROWIDTOCHAR (extension)

Syntax

ROWIDTOCHAR ( expression )

Description

The scalar function ROWIDTOCHAR returns the character form of a ROWID contained in the input argument. The representation of a row identifier depends on the storage manager. The format of the argument to this function is defined by the storage manager. See the documentation for your storage manager for details.

Example

The following example uses ROWIDTOCHAR to convert a row identifier from its internal representation to a character string. This example is specific to the Dharma storage manager's representation of a row identifier:

SELECT cust_no,  
    SUBSTR (ROWIDTOCHAR (ROWID), 1, 8) 'PAGE NUMBER',  
    SUBSTR (ROWIDTOCHAR (ROWID), 10, 4) 'LINE NUMBER',  
    SUBSTR (ROWIDTOCHAR (ROWID), 15, 4) 'TABLE SPACE NUMBER'  
FROM customer;

Notes

- The argument to the function must be a ROWID, as defined by the storage manager.
- The result is of CHARACTER type.
- If the argument expression evaluates to null, the result is null.

4.9.2.70  RPAD function (extension)

Syntax

RPAD ( char_expression, length [, pad_expression] )

Description

The scalar function RPAD pads the character string corresponding to the first argument on the right with the character string corresponding to the third argument so that after the padding, the length of the result would be equal to the value of the second argument length.

Example

SELECT RPAD (name, 30)  
FROM customer;

SELECT RPAD (name, 30, '.')  
FROM customer;
Notes

The first argument to the function must be of type CHARACTER.
The second argument to the function must be of type INTEGER.
The third argument, if specified, must be of type CHARACTER.
If the third argument is not specified, the default value is a string of length 1 containing one blank.
If \( L_1 \) is the length of the first argument and \( L_2 \) is the value of the second argument, then:

- If \( L_1 \) is less than \( L_2 \), the number of characters padded is equal to \( L_2 - L_1 \).
- If \( L_1 \) is equal to \( L_2 \), no characters are padded and the result string is the same as the first argument.
- If \( L_1 \) is greater than \( L_2 \), the result string is equal to the first argument truncated to first \( L_2 \) characters.

The result is of type CHARACTER.

If the argument expression evaluates to null, the result is null.

4.9.2.71 RTRIM function (ODBC compatible)

Syntax

\[
\text{RTRIM ( char_expression [ , char_set ] )}
\]

Description

The scalar function RTRIM removes all the trailing characters in \( \text{char_expression} \), that are present in \( \text{char_set} \) and returns the resultant string. Thus, the last character in the result is guaranteed to be not in \( \text{char_set} \). If the \( \text{char_set} \) argument is omitted, the function removes the leading and trailing blanks from \( \text{char_expression} \).

Example

\[
\begin{align*}
\text{SELECT RPAD ( RTRIM (addr, ')\'), 30, '.')} \\
\text{FROM customer ;}
\end{align*}
\]

Notes

- The first argument to the function must be of type CHARACTER.
- The second argument to the function must be of type CHARACTER.
- The result is of type CHARACTER.
- If the argument expression evaluates to null, the result is null.

4.9.2.72 SECOND function (ODBC compatible)

Syntax

\[
\text{SECOND ( time_expression )}
\]
Description

The scalar function SECOND returns the seconds in the argument as a short integer value in the range of 0 - 59.

Example

```sql
SELECT *
FROM arrivals
WHERE SECOND(in_time) <= 40;
```

Notes

- The argument to the function must be of type TIME.
- The argument must be specified in the format HH:MI:SS.
- The result is of type SHORT.
- If the argument expression evaluates to null, the result is null.

4.9.2.73 SIGN function (ODBC compatible)

Syntax

```sql
SIGN(expression)
```

Description

The scalar function SIGN returns 1 if `expression` is positive, -1 if `expression` is negative, or zero if it is zero.

Example

```sql
SELECT SIGN(-14) 'Sign'
FROM SYSTPE.SYSCALCTABLE;
```

Notes

- `expression` must evaluate to a numeric data type.

4.9.2.74 SIN function (ODBC compatible)

Syntax

```sql
SIN(expression)
```

Description

The scalar function SIN returns the sine of `expression`.

Example

```sql
select sin(45 * pi()/180) 'Sine of 45 degrees' from syscalctable;
```

```
SINE OF 45 DEGREES
-----------------
0.707106781186547
1 record selected
```
Notes

SIN takes an angle (expression) and returns the ratio of two sides of a right triangle. The ratio is the length of the side opposite the angle divided by the length of the hypotenuse.

- expression specifies an angle in radians
- expression must evaluate to an approximate numeric data type.

To convert degrees to radians, multiply degrees by $\pi/180$. To convert radians to degrees, multiply radians by $180/\pi$.

4.9.2.75 SOUNDEX function (ODBC compatible)

Syntax

```sql
SOUNDEX ( string_exp )
```

Description

The scalar function SOUNDEX returns a four-character soundex code for character strings that are composed of a contiguous sequence of valid single- or double-byte roman letters.

Example

```sql
SELECT SOUNDEX('Roberts')
FROM syscalctable;
```

4.9.2.76 SPACE function (ODBC compatible)

Syntax

```sql
SPACE ( count )
```

Description

The scalar function SPACE returns a character string consisting of count spaces.

Example

```sql
SELECT CONCAT(SPACE(3), name)
FROM customer
WHERE name = 'Roberts';
```

Notes

- The count argument can be of type INTEGER, SMALLINT, BIGINT, or TINYINT.
- If count is null, the result is null.
- If count is negative, the result is null.
4.9.2.77 SQRT function (ODBC compatible)

**Syntax**

```
SQRT ( expression )
```

**Description**

The scalar function SQRT returns the square root of `expression`.

**Example**

```
SELECT SQRT(28) 'square root of 28'
     FROM SYSTPE.SYSCALCTABLE;
```

**Notes**

- The value of `expression` must be positive.
- `expression` must evaluate to an approximate numeric data type.

4.9.2.78 SUBSTR function (extension)

**Syntax**

```
SUBSTR ( char_expression, start_position [, length ] )
```

**Description**

The scalar function SUBSTR returns the substring of the character string corresponding to the first argument starting at `start_position` and `length` characters long. If the third argument `length` is not specified, substring starting at `start_position` up to the end of `char_expression` is returned.

**Example**

```
SELECT name, '(', SUBSTR (phone, 1, 3) , ')',
       SUBSTR (phone, 4, 3), '-',
       SUBSTR (phone, 7, 4)
     FROM customer ;
```

**Notes**

- The first argument must be of type CHARACTER.
- The second argument must be of type INTEGER.
- The third argument, if specified, must be of type INTEGER.
- The values for specifying position in the character string start from 1: The very first character in a string is at position 1, the second character is at position 2 and so on.
- The result is of type CHARACTER.
- If any of the argument expressions evaluates to null, the result is null.
4.9.2.79  SUBSTRING function (ODBC compatible)

Syntax

```
SUBSTRING ( char_expression, start_position [, length ] )
```

Description

The scalar function SUBSTRING returns the substring of the character string corresponding to the first argument starting at start_position and length characters long. If the third argument length is not specified, the substring starting at start_position up to the end of char_expression is returned. SUBSTRING is identical to SUBSTR but provides ODBC-compatible syntax.

Example

```
SELECT name, '(', SUBSTRING (phone, 1, 3) , ')',
       SUBSTRING (phone, 4, 3), '-',
       SUBSTRING (phone, 7, 4)
FROM customer ;
```

Notes

- The first argument must be of type CHARACTER.
- The second argument must be of type INTEGER.
- The third argument, if specified, must be of type INTEGER.
- The values for specifying position in the character string start from 1: The very first character in a string is at position 1, the second character is at position 2 and so on.
- The result is of type CHARACTER.
- If any of the argument expressions evaluates to null, the result is null.

4.9.2.80  SUFFIX function (extension)

Syntax

```
SUFFIX(char_expression, start_position, char_expression)
```

Description

The scalar function SUFFIX returns the substring of a character string starting after the position specified by start_position and the second char_expression, to the end of the string.

Arguments

- **char_expression**
  An expression that evaluates to a character string, typically a character-string literal or column name. If the expression evaluates to null, SUFFIX returns null.

- **start_position**
  An expression that evaluates to an integer value. SUFFIX searches the string specified in the first argument starting at that position. A value of 1 indicates the first character of the string.
**char_expression**

An expression that evaluates to a single character. `SUFFIX` returns the substring that begins with that character. If `SUFFIX` does not find the character after `start_position`, it returns null. If the expression evaluates to more than one character, `SUFFIX` ignores all but the first character.

**Example**

```sql
SELECT C1, C2, SUFFIX(C1, 6, '.') FROM T1;
C1 C2 SUFFIX(C1,6,.
-- -- --------------
test.pref .
pref.test s
2 records selected

SELECT C1, C2, SUFFIX(C1, 1, C2) FROM T1;
C1 C2 SUFFIX(C1,1,C
-- -- --------------
test.pref . pref
test.test s t
2 records selected
```

4.9.2.81 **SUSER_NAME function (extension)**

**Syntax**

```sql
SUSER_NAME ( [user_id] )
```

**Description**

The scalar function `SUSER_NAME` returns the user login name for the `user_id` specified in the input argument. If no `user_id` is specified, `SUSER_NAME` returns the name of the current user.

This function provides compatibility with the Sybase SQL Server function `suser_name`. It is identical to the `USER_NAME` function.
Example

```
select suser_name() from systpe.syscalctable;
SUSER_NAME
----------
searle
1 record selected

select suser_name(104) from systpe.syscalctable;
SUSER_NAME(104)
----------
dbp
1 record selected

select id, tbl, owner from systpe.systables
 1 where owner = suser_name();
   ID  TBL  OWNER
    --  ---  ----
     41  test searle
     42    t2 searle
     43    t1 searle
3 records selected
```

4.9.2.82 SYSDATE function (extension)

Syntax

```
SYSDATE [ () ]
```

Description

SYSDATE returns the current date as a DATE value. This function takes no arguments, and the trailing parentheses are optional.

SQL statements can refer to SYSDATE anywhere they can refer to a DATE expression. For example,

```
INSERT INTO objects (object_owner, object_id, create_date)
VALUES (USER, 1001, SYSDATE);
```

4.9.2.83 SYSTIME function (extension)

Syntax

```
SYSTIME [ () ]
```

Description

SYSTIME returns the current time as a TIME value. This function takes no arguments, and the trailing parentheses are optional.

SQL statements can refer to SYSTIME anywhere they can refer to a TIME expression. For example,

```
INSERT INTO objects (object_owner, object_id, create_time)
VALUES (USER, 1001, SYSTIME);
```
4.9.2.84 SYSTIMESTAMP function (extension)

**Syntax**

```sql
SYSTIMESTAMP ()
```

**Description**

The `SYSTIMESTAMP` function returns the current date and time as a `TIMESTAMP` value. This function takes no arguments, and the trailing parentheses are optional.

The following SQL example shows the different formats for `SYSDATE`, `SYSTIME`, and `SYSTIMESTAMP`:

```sql
SELECT SYSDATE FROM test;
SYSDATE
-------
09/13/1994
1 record selected

SELECT SYSTIME FROM test;
SYSTIME
-------
14:44:07:000
1 record selected

SELECT SYSTIMESTAMP FROM test;
SYSTIMESTAMP
-------------
1994-09-13 14:44:15:000
1 record selected
```

4.9.2.85 TAN function (ODBC compatible)

**Syntax**

```sql
TAN ( expression )
```

**Description**

The scalar function `TAN` returns the tangent of `expression`.

**Example**

```sql
SELECT TAN(45 * pi()/180) 'Tangent of 45 degrees' FROM syscalctable;
TANGENT OF 45 DE
--------------
1.000000000000000
1 record selected
```

**Notes**

- `TAN` takes an angle (expression) and returns the ratio of two sides of a right triangle. The ratio is the length of the side opposite the angle divided by the length of the side adjacent to the angle.
  - `expression` specifies an angle in radians
  - `expression` must evaluate to an approximate numeric data type.

To convert degrees to radians, multiply degrees by `Pi/180`. To convert radians to degrees, multiply radians by `180/Pi`. 
4.9.2.86 TIMESTAMPADD function (ODBC compatible)

Syntax

TIMESTAMPADD(interval, integer_exp, date_time_exp)

interval::=
    SQL_TSI_FRAC_SECOND
|    SQL_TSI_SECOND
|    SQL_TSI_MINUTE
|    SQL_TSI_HOUR
|    SQL_TSI_DAY
|    SQL_TSI_WEEK
|    SQL_TSI_MONTH
|    SQL_TSI_QUARTER
|    SQL_TSI_YEAR

Description

Returns the timestamp calculated by adding integer_exp intervals of type interval to
timestamp_exp.

Arguments

interval
Keywords that specify the interval to add to timestamp_exp. The
SQL_TSI_FRAC_SECOND keyword specifies fractional seconds as billionths of a
second.

integer_exp
The number of interval values to add to timestamp_exp. integer_exp can be any expression
that evaluates to an integer data type.

date_time_exp
A date-time expression from which TIMESTAMPADD calculates the return value. If
date_time_exp is a date value and interval specifies fractional seconds, seconds, minutes,
or hours, the time portion of timestamp_exp is set to 0 before calculating the resulting
timestamp.

Example

The following example displays the current system time and uses the TIMESTAMPADD
scalar function to add 8 hours to it.

> select systime, timestampadd(sql_tsi_hour, 8, systime) from
  syscalctable;
15:03:57:000    06/08/1999 23:03:57:000
------------    -----------------------
15:03:57:000,   1999-06-08 23:03:57:000

4.9.2.87 TIMESTAMPDIFF function (ODBC compatible)

Syntax

TIMESTAMPDIFF(interval, date_time_exp1, date_time_exp2)
interval::
| SQL_TSI_FRAC_SECOND
| SQL_TSI_SECOND
| SQL_TSI_MINUTE
| SQL_TSI_HOUR
| SQL_TSI_DAY
| SQL_TSI_WEEK
| SQL_TSI_MONTH
| SQL_TSI_QUARTER
| SQL_TSI_YEAR

Description

Returns an integer representing the number of intervals by which `date_time_exp2` is greater than `date_time_exp1`.

Arguments

interval
Keywords that specify the interval in which to express the difference between the two date-time arguments. The SQL_TSI_FRAC_SECOND keyword specifies fractional seconds as billionths of a second.

date_time_exp1
A date-time expression which `TIMESTAMPADD` subtracts from `date_time_exp2`.

date_time_exp1
A date-time expression from which `TIMESTAMPADD` subtracts `date_time_exp1`.

Example

The following example displays difference in seconds between the current system time and one day later.

```sql
> select timestampdiff(sql_tsi_second, sysdate, sysdate + 1) from syscalctable;
86400
```

Notes

If either date-time expression is a time value and `interval` specifies days, weeks, months, quarters, or years, the date portion of that expression is set to the current date before calculating the difference between the expressions.

If either date-time expression is a date value and `interval` specifies fractional seconds, seconds, minutes, or hours, the time portion of that expression is set to 0 before calculating the difference between the expressions.

4.9.2.88 TO_CHAR function (extension)

Syntax

```sql
TO_CHAR ( expression [, format_string ] )
```
Description

The scalar function TO_CHAR converts the given expression to character form and returns the result. The primary use for TO_CHAR is to format the output of date-time expressions through the format_string argument.

Arguments

expression
Specifies the expression to be converted to character form. To use the format_string argument, expression must evaluate to a date or time value.

format_string
A date-time format string that specifies the format of the output. See “Date Format Strings” and “Time Format Strings” for details on format strings.

SQL ignores the format string if the expression argument does not evaluate to a date or time.

Example

```
SELECT C1 FROM T2;
C1
--
09/29/1952
1 record selected
SELECT TO_CHAR(C1, 'Day, Month ddth'),
       TO_CHAR(C2, 'HH12 a.m.') FROM T2;
TO_CHAR(C1,DAY, MONTH DDTH)  TO_CHAR(C2,HH12 A.M.)
----------------------------  ---------------------
Monday, September 29th    02 p.m.
1 record selected
```

Notes

- The first argument to the function can be of any type.
- The second argument, if specified, must be of type CHARACTER.
- The result is of type CHARACTER.
- The format argument can be used only when the type of the first argument is DATE.
- If any of the argument expressions evaluates to null, the result is null.

4.9.2.89 TO_DATE function (extension)

Syntax

```
TO_DATE ( date_lit )
```

Description

The scalar function TO_DATE converts the given date literal to a date value.

Example

```
SELECT *
FROM  orders
WHERE order_date <= TO_DATE ('12/31/1991') ;
```
Notes

- The result is of type DATE.
- Supply the date literal in any valid format. See "Date Literals" for valid formats.

4.9.2.90 TO_NUMBER function (extension)
Syntax

TO_NUMBER ( char_expression )

Description

The scalar function TO_NUMBER converts the given character expression to a number value.

Example

SELECT *
FROM customer
WHERE TO_NUMBER (SUBSTR (phone, 1, 3)) = 603 ;

Notes

- The argument to the function must be of type CHARACTER.
- The result is of type NUMERIC.
- If any of the argument expressions evaluates to null, the result is null.

4.9.2.91 TO_TIME function (extension)
Syntax

TO_TIME ( time_lit )

Description

The scalar function TO_TIME converts the given time literal to a time value.

Example

SELECT *
FROM orders
WHERE order_date < TO_DATE ('05/15/1991')
AND order_time < TO_TIME ('12:00:00') ;

Notes

- The result is of type TIME.
- Supply the time literal in any valid format. See "Time Literals" for valid formats.

4.9.2.92 TO_TIMESTAMP function (extension)
Syntax

TO_TIMESTAMP ( timestamp_lit )
Description

The scalar function TO_TIMESTAMP converts the given timestamp literal to a timestamp value.

Example

```
SELECT * FROM DTEST
WHERE C3 = TO_TIMESTAMP('4/18/95 10:41:19')
```

Notes

- The result is of type TIME.
- Supply the timestamp literal in any valid format. See "Timestamp Literals" for valid formats.

4.9.2.94 TRANSLATE function (extension)

Syntax

```
TRANSLATE ( char_expression, from_set, to_set )
```

Description

The scalar function TRANSLATE translates each character in char_expression that is in from_set to the corresponding character in to_set. The translated character string is returned as the result. This function is similar to the Oracle TRANSLATE function.

Example

This example substitutes underscores for spaces in customer names.

```
SELECT TRANSLATE (customer_name, ' ', '_')
   "TRANSLATE Example" from customers;
```

Notes

- char_expression, from_set, and to_set can be any character expression.
- For each character in char_expression, TRANSLATE checks for the same character in from_set:
  - If it is in from_set, TRANSLATE translates it to the corresponding character in to_set (if the character is the nth character in from_set, the nth character in to_set).
  - If the character is not in from_set TRANSLATE does not change it.
• If `from_set` is longer than `to_set`, `TRANSLATE` does not change trailing characters in `from_set` that do not have a corresponding character in `to_set`.

• If either `from_set` or `to_set` is null, `TRANSLATE` does nothing.

4.9.2.95 UCASE function (ODBC compatible)

Syntax

```
UCASE ( char_expression )
```

Description

The scalar function UCASE returns the result of the argument character expression after converting all the characters to upper case. UCASE is identical to UPPER, but provides ODBC-compatible syntax.

Example

```
SELECT *
FROM customer
WHERE UCASE (name) = 'SMITH' ;
```

Notes

• The argument to the function must be of type CHARACTER.

• The result is of type CHARACTER.

• If the argument expression evaluates to null, the result is null.

4.9.2.96 UID function (extension)

UID returns an integer identifier for the user of the current transaction, as determined by the host operating system.

The value contained in this register is of INTEGER type. The host representation is of long integer type. SQL statements can refer to UID anywhere they can refer to an integer expression. For example,

```
INSERT INTO objects (owner_id, object_id)
VALUES (UID, 1001) ;
```

```
SELECT *
FROM objects
WHERE owner_id = UID ;
```

4.9.2.97 UPPER function (SQL-92 compatible)

Syntax

```
UPPER ( char_expression )
```

Description

The scalar function UPPER returns the result of the argument character expression after converting all the characters to upper case.
Example

```sql
SELECT *
FROM customer
WHERE UPPER (name) = 'SMITH' ;
```

Notes

- The argument to the function must be of type CHARACTER.
- The result is of type CHARACTER.
- If the argument expression evaluates to null, the result is null.

### 4.9.2.98 USER function (ODBC compatible)

**Syntax**

```
USER [ ( ) ]
```

**Description**

USER returns a character-string identifier for the database user, as specified in the current connection. If the current connection did not specify a user, USER returns the login name as determined by the host operating system. This function takes no arguments, and the trailing parentheses are optional.

SQL statements can refer to USER anywhere they can refer to a character string expression.

**Example**

The following interactive SQL example shows connecting to a database as the user fred. Queries on two system tables illustrate the USER scalar function and retrieve the names of any tables owned by the user fred:

```sql
% isql -u fred tstdb

Dharma/isql Version 6.2.1
Dharma Systems Inc (C) 1988-99.
Dharma Computers Pvt Ltd (C) 1988-99.

ISQL> select user from systpe.syscalctable;
FRED
----
fred
1 record selected
ISQL> select tbl, owner from systpe.systables where owner = user();
TBL       OWNER
---       ----
flab       fred
1 record selected
```

### 4.9.2.99 USER_NAME function (extension)

**Syntax**

```
USER_NAME ( [user_id] )
```
Description

The scalar function USER_NAME returns the user login name for the user_id specified in the input argument. If no user_id is specified, USER_NAME returns the name of the current user.

The scalar function USER_NAME is identical to SUSER_NAME.

4.9.2.100 WEEK function (ODBC compatible)

Syntax

WEEK ( time_expression )

Description

The scalar function WEEK returns the week of the year as a short integer value in the range of 1 - 53.

Example

```
SELECT *
FROM  orders
WHERE WEEK (order_date) = 5 ;
```

Notes

- The argument to the function must be of type DATE.
- The argument must be specified in the format MM/DD/YYYY.
- The result is of type SHORT.
- If the argument expression evaluates to null, the result is null.

4.9.2.101 YEAR function (ODBC compatible)

Syntax

YEAR ( date_expression )

Description

The scalar function YEAR returns the year as a short integer value in the range of 0 - 9999.

Example

```
SELECT *
FROM  orders
WHERE YEAR (order_date) = 1992 ;
```

Notes

- The argument to the function must be of type DATE.
- The argument must be specified in the format MM/DD/YYYY.
- The result is of type SHORT.
• If the argument expression evaluates to null, the result is null.
This chapter provides detailed reference material on each SQL statement.

Syntax Conventions

Syntax diagrams appear in helvetica type and use the following conventions:

**UPPERCASE**

Uppercase type denotes reserved words. You must include reserved words in statements, but they can be upper or lower case.

**lowercase**

Lowercase type denotes either user-supplied elements or names of other syntax diagrams. User-supplied elements include names of tables, host-language variables, expressions, and literals. Syntax diagrams can refer to each other by name. If a diagram is named, the name appears in lowercase type above and to the left of the diagram, followed by a double-colon (for example, privilege ::). The name of that diagram appears in lowercase in diagrams that refer to it.

{} Braces denote a choice among mandatory elements. They enclose a set of options, separated by vertical bars ( | ). You must choose at least one of the options.

[] Brackets denote an optional element or a choice among optional elements.

| Vertical bars separate a set of options.

... A horizontal ellipsis denotes that the preceding element can optionally be repeated any number of times.

() , ; Parentheses and other punctuation marks are required elements. Enter them as shown in syntax diagrams.
5.1 CALL

Description

Invokes a stored procedure.

Syntax

[ ? = ] CALL proc_name([parameter][,...]);

Arguments

[ ? = ]
A parameter marker for the return value of the procedure. Programs must determine if
the procedure returns a value and use the parameter marker as a placeholder for a
variable that will receive the return value.

CALL proc_name
The name of the procedure to invoke.

parameter
Literal or variable values to pass to the procedure.

Example

The following examples show invocation of the SQLPROC built-in procedure from
interactive SQL. It invokes SQLPROC to retrieve information about another built-in
procedure, SQLTABLES:

CALL SQLPROC('',0,'systpe',6,'sqltables',9); -- specific procedure

<table>
<thead>
<tr>
<th>PROCEDURE_QUALIFIER</th>
<th>PROCEDURE_OWNER</th>
<th>PROCEDURE_NAME</th>
<th>NUM_INP</th>
<th>NUM_OUTP</th>
<th>NUM_RESU</th>
<th>REMARKS</th>
</tr>
</thead>
<tbody>
<tr>
<td>systpe</td>
<td>sqltables</td>
<td>6</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>Returns info about a table</td>
</tr>
</tbody>
</table>

1 record returned

Authorization

- Users must have the DBA or EXECUTE privilege to invoke a stored procedure.
- Users invoking a stored procedure do not need privileges to database objects
  accessed by the procedure. When a user executes a stored procedure, SQL checks
  the privileges of the procedure owner, not the procedure user, on any objects that the
  procedure accesses. This enables a user to execute a procedure successfully even
  when that user does not have the privileges to directly access objects used by the
  procedure.

SQL Compliance

ODBC Extended SQL grammar, when enclosed in ODBC escape clause

Environment

Embedded SQL, interactive SQL, ODBC applications

Related Statements

CREATE PROCEDURE, DROP PROCEDURE
5.2 CREATE INDEX

Description

Creates an index on the specified table using the specified columns of the table. An index improves the performance of SQL operations whose predicates are based on the indexed column. However, an index slows performance of INSERT, UPDATE and DELETE operations.

Syntax

```
CREATE [ UNIQUE ] INDEX index_name
    ON table_name
    ([column_name 
      [ASC | DESC]] [, ...])
    [ PCTFREE number]
    [ STORAGE_ATTRIBUTES 'attributes'
    [ TYPE 'ix_type' ];
```

Arguments

**UNIQUE**
A UNIQUE index will not allow the table to contain any rows with duplicate column values for the set of columns specified for that index.

**index_name**
The name of the index has to be unique within the local database.

**table_name**
The name of the table on which the index is being built.

**column_name [, ...]**
The columns on which searches and retrievals will be ordered. These columns are called the index key. When more than one column is specified in the CREATE INDEX statement a concatenated index is created.

**ASC | DESC**
The index can be ordered as either ascending (ASC) or descending (DESC) on each column of the concatenated index. The default is ASC.

**PCTFREE number**
Specifies the desired percentage of free space for a index. The PCTFREE clause indicates to the storage system how much of the space allocated to an index should be left free to accommodate growth. However, the actual behavior of the PCTFREE clause depends entirely on the underlying storage system. The SQL engine passes the PCTFREE value to the storage system, which may ignore it or interpret it. If the CREATE statement does not include a PCTFREE clause, the default is 20. See the documentation for your storage system for details.

**STORAGE_ATTRIBUTES 'attributes'**
A quoted string that specifies index attributes that are specific to a particular storage system. The SQL engine passes this string to the storage system, and its effects are defined by the storage manager. See the documentation for your storage system for details.
Overview

**TYPE 'ix_type'**
A single-character that specifies the type of index. The valid values for the TYPE argument and their meanings are specific to the underlying storage system. See the documentation for your storage system for details.

**Example**

```
CREATE UNIQUE INDEX custindex ON customer (cust_no) ;
```

**Authorization**

The user executing this statement must have any of the following privileges:

- DBA privilege
- Ownership of the index.
- INDEX privilege on the table.

**SQL Compliance**

ODBC Core SQL grammar. Extensions: PCTFREE, STORAGE_ATTRIBUTES, and TYPE

**Environment**

Embedded SQL, interactive SQL, ODBC applications

**Related Statements**

CREATE TABLE, DROP INDEX, ALTER INDEX
5.3 CREATE SYNONYM

Description

Creates a synonym for the table, view or synonym specified. A synonym is an alias that SQL statements can use instead of the name specified when the table, view, or synonym was created.

Syntax

CREATE [PUBLIC] SYNONYM synonym
  FOR [owner_name.] { table_name | view_name | synonym } ;

Arguments

PUBLIC
Specifies that the synonym will be public: all users can refer to the name without qualifying it. By default, the synonym is private: other users must qualify the synonym by preceding it with the user name of the user who created it.

Users must have the DBA privilege to create public synonyms.

SYNONYM synonym
Name for the synonym.

FOR [owner_name.] { table_name | view_name | synonym }
Table, view, or synonym for which SQL creates the new synonym.

Example

CREATE SYNONYM customer FOR smith.customer ;
CREATE PUBLIC SYNONYM public_suppliers FOR smith.suppliers ;

Authorization

Users executing CREATE SYNONYM must have the DBA privilege or RESOURCE privilege. Users executing CREATE PUBLIC SYNONYM statement must have the DBA privilege.

SQL Compliance

Extension

Environment
Embedded SQL, interactive SQL, ODBC applications

Related Statements
DROP SYNONYM
5.4 CREATE TABLE

Description

Creates a table definition. A table definition consists of a list of column definitions that make up a table row. SQL provides two forms of the CREATE TABLE statement. The first form explicitly specifies column definitions. The second form, with the AS query_expression clause, implicitly defines the columns using the columns in the query expression.

Syntax

CREATE TABLE [ owner_name. ] table_name
               ( column_definition [ , { column_definition | table_constraint } ] ... )
               [ TABLE SPACE table_space_name ]
               [ PCTFREE number ]
               [ STORAGE_MANAGER 'sto-mgr-id' ]
               [ STORAGE_ATTRIBUTES 'attributes' ]
               ;

CREATE TABLE [ owner_name. ] table_name
               [ ( column_name [NULL | NOT NULL], ...) ]
               [ TABLE SPACE table_space_name ]
               [ PCTFREE number ]
               [ STORAGE_MANAGER 'sto-mgr-id' ]
               [ STORAGE_ATTRIBUTES 'attributes' ]
               AS query_expression
               ;

column_definition ::
   column_name data_type
   [ COLLATE collation_name ]
   [ DEFAULT { literal | USER | NULL | UID | SYSDATE | SYSTIME | SYSTIMESTAMP } ]
   [ column_constraint [ column_constraint ... ] ]

Arguments

owner_name
Specifies the owner of the table. If the name is different from the user name of the user executing the statement, then the user must have DBA privileges.

table_name
Names the table definition. SQL defines the table in the database named in the last CONNECT statement.

column_name data_type
Names a column and associates a data type with it. The column names specified must be different than other column names in the table definition. The data_type must be one of the supported data types described in "Data Types" on page 4-4.
CREATE TABLE

If data_type specifies a character column, the column definition can include an optional COLLATE clause. The COLLATE clause specifies a collation sequence supported by the underlying storage system. (See "Specifying the Character Set for Character Data Types" on page 4-6 for notes on character sets and collations, including the CASE_INSENSITIVE collation sequence supported for the default character set. See the documentation for your underlying storage system for details on any supported collations.)

DEFAULT
Specifies an explicit default value for a column. The column takes on the value if an INSERT statement does not include a value for the column. If a column definition omits the DEFAULT clause, the default value is NULL.

The DEFAULT clause accepts the following arguments:

 literal An integer, numeric or string constant.
 USER The name of the user issuing the INSERT or UPDATE statement on the table. Valid only for columns defined with character data types.
 NULL A null value.
 UID The user id of the user executing the INSERT or UPDATE statement on the table.
 SYSDATE The current date. Valid only for columns defined with DATE data types.
 SYSTIME The current time. Valid only for columns defined with TIME data types.
 SYSTIMESTAMP The current date and time. Valid only for columns defined with TIMESTAMP data types.

column_constraint Specifies a constraint that applies while inserting or updating a value in the associated column. For more information, see "Column Constraints".

table_constraint Specifies a constraint that applies while inserting or updating a row in the table. For more information, see "Table Constraints".

TABLE SPACE table_space_name
Specifies the name of the table space where data stored in the table will reside. Table spaces provide a way to partition tables among different storage areas. In some storage systems, for instance, table spaces correspond to separate data files among which data in tables can be distributed. This arrangement can improve performance by distributing data in a table on different disk drives.

Different storage systems implement the concept of storage areas in different ways, if at all. So the actual behavior of the TABLE SPACE clause depends on the underlying storage system. See the documentation for your storage system for more details.
PCTFREE number
Specifies the desired percentage of free space for a table. The PCTFREE clause indicates to
the storage system how much of the space allocated to a table should be left free to
accommodate growth.

However, the actual behavior of the PCTFREE clause depends entirely on the underlying
storage system. The SQL engine passes the PCTFREE value to the storage system, which
may ignore it or interpret it. If the CREATE statement does not include a PCTFREE
clause, the default is 20. See the documentation for your storage system for details.

STORAGE_MANAGER 'sto-mgr-id'
A quoted string that identifies the storage system. The SQL engine uses the string to
identify which storage system will create the table. If the CREATE TABLE statement
omits the STORAGE_MANAGER clause, the SQL engine uses the string 'default'. What
constitutes a valid name, and how the table is mapped to a specific storage system is
defined by the implementation.

STORAGE_ATTRIBUTES ‘attributes’
A quoted string that specifies table attributes that are specific to a particular storage
system. The SQL engine passes this string to the storage system, and its effects are
defined by the storage manager. See the documentation for your storage system for
details.

AS query_expression
Specifies a query expression to use for the data types and contents of the columns for the
table. The types and lengths of the columns of the query expression result become the
types and lengths of the respective columns in the table created. The rows in the
resultant set of the query expression are inserted into the table after creating the table. In
this form of the CREATE TABLE statement, column names are optional.

If omitted, the names for the table columns are also derived from the query expression.
For more information, see “Query Expressions”.

Examples
In the following example, the user issuing the CREATE TABLE statement must have
REFERENCES privilege on the column itemno of the table john.item.

```sql
CREATE TABLE supplier_item (  
supp_no INTEGER NOT NULL PRIMARY KEY,  
item_no INTEGER NOT NULL REFERENCES john.item (itemno),  
qty INTEGER  
) ;
```

The following CREATE TABLE statement explicitly specifies a table owner, systpe

```sql
CREATE TABLE systpe.account (  
account integer,  
balance money (12),  
info char (84)  
) ;
```

The following example shows the AS query_expression form of CREATE TABLE to
create and load a table with a subset of the data in the customer table:
CREATE TABLE systpe.dealer (name, street, city, state) AS
    SELECT name, street, city, state
    FROM customer
    WHERE customer.state IN ('CA', 'NY', 'TX');

The following example includes a NOT NULL column constraint and DEFAULT clauses for column definitions:

CREATE TABLE emp (empno  integer NOT NULL,
                deptno  integer DEFAULT 10,
                join_date date DEFAULT NULL
           ) ;

Authorization

The user executing this statement must have either DBA or RESOURCE privilege. If the
CREATE TABLE statement specifies a foreign key that references a table owned by a
different user, the user must have the REFERENCES privilege on the corresponding
columns of the referenced table.

The AS query_expression form of CREATE TABLE requires the user to have select
privilege on all the tables and views named in the query expression.

SQL Compliance  SQL-92, ODBC Minimum SQL grammar. Extensions: TABLE SPACE, PCTFREE,
STORAGE_MANAGER, and AS query_expression

Environment  Embedded SQL, interactive SQL, ODBC applications

Related Statements  DROP TABLE, Query Expressions

5.4.1 Column Constraints

Description

Specifies a constraint for a column that restricts the values that the column can store.
INSERT, UPDATE, or DELETE statements that violate the constraint fail. SQL returns a
Constraint violation error with SQLCODE of -20116.

Column constraints are similar to table constraints but their definitions are associated
with a single column.

Syntax

column_constraint ::
  NOT NULL [ PRIMARY KEY | UNIQUE ]
  | REFERENCES [ owner_name. ] table_name [ ( column_name ) ]
  | CHECK ( search_condition )

Arguments

NOT NULL
Restricts values in the column to values that are not null.
**NOT NULL PRIMARY KEY**
Defines the column as the primary key for the table. There can be exactly one primary key for a table. A column with the NOT NULL PRIMARY KEY constraint cannot contain null or duplicate values. Other tables can name primary keys as foreign keys in their REFERENCES clauses.

Other tables can name primary keys in their REFERENCES clauses. If they do, SQL restricts operations on the table containing the primary key:

- DROP TABLE statements that delete the table fail
- DELETE and UPDATE statements that modify values in the column that match a foreign key’s value also fail

The following example shows the creation of a primary key column on the table `supplier`:

```
CREATE TABLE supplier (
    supp_no     INTEGER NOT NULL PRIMARY KEY,
    name        CHAR (30),
    status      SMALLINT,
    city        CHAR (20)
) ;
```

**NOT NULL UNIQUE**
Defines the column as a unique key that cannot contain null or duplicate values. Columns with NOT NULL UNIQUE constraints defined for them are also called candidate keys.

Other tables can name unique keys in their REFERENCES clauses. If they do, SQL restricts operations on the table containing the unique key:

- DROP TABLE statements that delete the table fail
- DELETE and UPDATE statements that modify values in the column that match a foreign key’s value also fail

The following example creates a NOT NULL UNIQUE constraint to define the column `ss_no` as a unique key for the table `employee`:

```
CREATE TABLE employee (
    empno        INTEGER NOT NULL PRIMARY KEY,
    ss_no        INTEGER NOT NULL UNIQUE,
    ename        CHAR (19),
    sal          NUMERIC (10, 2),
    deptno       INTEGER NOT NULL
) ;
```

**REFERENCES table_name [ (column_name) ]**
Defines the column as a foreign key and specifies a matching primary or unique key in another table. The REFERENCES clause names the matching primary or unique key.

A foreign key and its matching primary or unique key specify a referential constraint: A value stored in the foreign key must either be null or be equal to some value in the matching unique or primary key.
You can omit the `column_name` argument if the table specified in the `REFERENCES` clause has a primary key and you want the primary key to be the matching key for the constraint.

The following example defines `order_item.orditem_order_no` as a foreign key that references the primary key `orders.order_no`.

```sql
CREATE TABLE orders (
    order_no INTEGER NOT NULL PRIMARY KEY,
    order_date DATE
) ;
CREATE TABLE order_item (
    orditem_order_no INTEGER REFERENCES orders ( order_no ),
    orditem_quantity INTEGER
) ;
```

Note that the second CREATE TABLE statement in the previous example could have omitted the column name `order_no` in the `REFERENCES` clause, since it refers to the primary key of table `orders`.

**CHECK (search_condition)**

Specifies a column-level check constraint. SQL restricts the form of the search condition. The search condition must not:

- Refer to any column other than the one with which it is defined
- Contain aggregate functions, subqueries, or parameter references

The following example creates a check constraint:

```sql
CREATE TABLE supplier (
    supp_no     INTEGER NOT NULL,
    name        CHAR (30),
    status      SMALLINT,
    city        CHAR (20) CHECK (supplier.city <> 'MOSCOW')
) ;
```

### 5.4.2 Table Constraints

**Description**

Specifies a constraint for a table that restricts the values that the table can store. INSERT, UPDATE, or DELETE statements that violate the constraint fail. SQL returns a Constraint violation error.

Table constraints have syntax and behavior similar to column constraints. Note the following differences:

- The syntax for table constraints is separated from column definitions by commas.
- Table constraints must follow the definition of columns they refer to.
- Table constraint definitions can include more than one column and SQL evaluates the constraint based on the combination of values stored in all the columns.
SQL Statements

Syntax

```plaintext
table_constraint ::
    PRIMARY KEY ( column [, ... ] )
    | UNIQUE ( column [, ... ] )
    | FOREIGN KEY ( column [, ... ] )
    REFERENCES [ owner_name. ] table_name [ ( column [, ... ] ) ]
    | CHECK ( search_condition )
```

Arguments

**PRIMARY KEY ( column [, ... ] )**
Defines the column list as the primary key for the table. There can be at most one primary key for a table.

All the columns that make up a table-level primary key must be defined as NOT NULL, or the CREATE TABLE statement fails. The combination of values in the columns that make up the primary key must be unique for each row in the table.

Other tables can name primary keys in their REFERENCES clauses. If they do, SQL restricts operations on the table containing the primary key:

- DROP TABLE statements that delete the table fail
- DELETE and UPDATE statements that modify values in the combination of columns that match a foreign key's value also fail

The following example shows creation of a table-level primary key. Note that its definition is separated from the column definitions by a comma:

```sql
CREATE TABLE supplier_item (
    supp_no   INTEGER NOT NULL,
    item_no   INTEGER NOT NULL,
    qty       INTEGER NOT NULL DEFAULT 0,
    PRIMARY KEY (supp_no, item_no)
) ;
```

**UNIQUE ( column [, ... ] )**
Defines the column list as a unique, or candidate, key for the table. Unique key table-level constraints have the same rules as primary key table-level constraints, except that you can specify more than one UNIQUE table-level constraint in a table definition.

The following example shows creation of a table with two UNIQUE table-level constraints:

```sql
CREATE TABLE order_item (
    order_no    INTEGER NOT NULL,
    item_no     INTEGER NOT NULL,
    qty         INTEGER NOT NULL,
    price       MONEY NOT NULL,
    UNIQUE (order_no, item_no),
    UNIQUE (qty, price)
) ;
```

**FOREIGN KEY ... REFERENCES**
Defines the first column list as a foreign key and, in the REFERENCES clause, specifies a matching primary or unique key in another table.
A foreign key and its matching primary or unique key specify a **referential constraint**: The combination of values stored in the columns that make up a foreign key must either:

- Have at least one of the column values be null
- Be equal to some corresponding combination of values in the matching unique or primary key

You can omit the column list in the REFERENCES clause if the table specified in the REFERENCES clause has a primary key and you want the primary key to be the matching key for the constraint.

The following example defines the combination of columns `student_courses.teacher` and `student_courses.course_title` as a foreign key that references the primary key of the table `courses`. Note that the REFERENCES clause does not specify column names because the foreign key refers to the primary key of the `courses` table.

```sql
CREATE TABLE courses (
    teacher        CHAR (20) NOT NULL,
    course_title   CHAR (30) NOT NULL,
    PRIMARY KEY (teacher, course_title)
) ;
CREATE TABLE student_courses (
    student_id     INTEGER,
    teacher        CHAR (20),
    course_title   CHAR (30),
    FOREIGN KEY (teacher, course_title) REFERENCES courses
) ;
```

SQL evaluates the referential constraint to see if it satisfies the following search condition:

```sql
(student_courses.teacher IS NULL
  OR student_courses.course_title IS NULL) OR
EXISTS (SELECT * FROM student_courses WHERE
  (student_courses.teacher = courses.teacher AND
   student_courses.course_title = courses.course_title)
)
```

**CHECK (search_condition)**

Specifies a table-level check constraint. The syntax for table-level and column level check constraints is identical. Table-level check constraints must be separated by commas from surrounding column definitions.

SQL restricts the form of the search condition. The search condition must not:

- Refer to any column other than columns that precede it in the table definition
- Contain aggregate functions, subqueries, or parameter references

The following example creates a table with two column-level check constraints and one table-level check constraint:
CREATE TABLE supplier (  
supp_no INTEGER NOT NULL,  
name CHAR (30),  
status SMALLINT CHECK (supplier.status BETWEEN 1 AND 100),  
city CHAR (20) CHECK (supplier.city IN ('NEW YORK', 'BOSTON', 'CHICAGO')),  
CHECK (supplier.city <> 'CHICAGO' OR supplier.status = 20)  
) ;
5.5 CREATE VIEW

Description

Creates a view with the specified name on existing tables and/or views.

Syntax

CREATE VIEW [ owner_name. ] view_name
  [ ( column_name, column_name,... ) ]
  AS [ ( ) query_expression ]
  [ WITH CHECK OPTION ] ;

Notes

- The owner_name is made the owner of the created view.
- The column names specified for the view are optional and provide an alias for the columns selected by the query specification. If the column names are not specified then the view will be created with the same column names as the tables and/or views it is based on.
- A view is deletable if deleting rows from that view is allowed. For a view to be deletable, the view definition has to satisfy the following conditions:
  - The first FROM clause contains only one table reference or one view reference.
  - There are no aggregate functions, DISTINCT clause, GROUP BY or HAVING clause in the view definition.
  - If the first FROM clause contains a view reference, then the view referred to is deletable.
- A view is updatable if updating rows from that view is allowed. For a view to be updatable, the view has to satisfy the following conditions:
  - The view is deletable (That is, it satisfies all the conditions specified above for deletability).
  - All the select expressions in the first SELECT clause of the view definition are simple column references.
  - If the first FROM clause contains a view reference, then the view referred to is updatable.
- A view is insertable if inserting rows into that view is allowed. For a view to be insertable, the view has to satisfy the following conditions:
  - The view is updatable (That is, it satisfies all the conditions specified above for updatability).
  - If the first FROM clause contains a table reference, then all NOT NULL columns of the table are selected in the first SELECT clause of the view definition.
• If the first FROM clause contains a view reference, then the view referred to is insertable.

• The WITH CHECK OPTION clause can be specified only if the view is updatable.

• If WITH CHECK OPTION clause is specified when defining a view, then during any update or insert of a row on this view, it is checked that the updated/inserted row satisfies the view definition (That is, the row is selectable using the view).

Examples

```sql
CREATE VIEW ne_customers AS
    SELECT cust_no, name, street, city, state, zip
    FROM customer
    WHERE state IN ('NH', 'MA', 'NY', 'VT')
    WITH CHECK OPTION;

CREATE VIEW order_count (cust_number, norders) AS
    SELECT cust_no, COUNT(*)
    FROM orders
    GROUP BY cust_no;
```

Authorization

The user executing this statement must have the following privileges:

• DBA or RESOURCE privilege.

• SELECT privilege on all the tables/views referred to in the view definition.

If owner_name is specified and is different from the name of the user executing the statement, then the user must have DBA privilege.

SQL Compliance

SQL-92, ODBC Core SQL grammar

Environment

Embedded SQL, interactive SQL, ODBC applications

Related Statements

Query Expressions, DROP VIEW
5.6 DELETE

Description

Deletes zero, one or more rows from the specified table that satisfy the search condition specified in the WHERE clause. If the optional WHERE clause is not specified, then the DELETE statement deletes all rows of the specified table.

Syntax

```
DELETE FROM [owner_name.] { table_name | view_name }
    [ WHERE search_condition ];
```

Notes

- The FROM clause of a subselect statement in the search condition, if any, can refer only to the table being deleted.
- If the table has primary/candidate keys, and if there exists references from other tables to the rows to be deleted, the statement is rejected.

Example

```
DELETE FROM customer
    WHERE customer_name = 'RALPH' ;
```

Authorization

The user executing this statement must have any of the following privileges:

- DBA privilege.
- Ownership of the table.
- DELETE permission on the table.

If the target is a view, then the DELETE privilege is required on the target base table referred to in the view definition.

SQL Compliance

SQL-92, ODBC Extended SQL grammar

Environment

Embedded SQL, interactive SQL, ODBC applications

Related Statements

Search Conditions
5.7 DROP INDEX

Description

Deletes an index on the specified table.

Syntax

DROP INDEX [owner_name.]index_name
 [ON [table_owner_name.]table_name]

Arguments

owner_name
If owner_name is specified and is different from the name of the user executing the statement, then the user must have DBA privileges.

index_name
The name of the index to delete.

table_name
The table_name argument is optional. If specified, index_name is verified to correspond to the table.

Example

DROP INDEX custindex ON customer ;

Authorization

The user executing this statement must have any of the following privileges:

• DBA privilege
• Ownership of the index

SQL Compliance

ODBC Core SQL grammar

Environment

Embedded SQL, interactive SQL, ODBC applications

Related Statements

CREATE INDEX, ALTER INDEX
5.8 DROP SYNONYM

Description

Drops the specified synonym.

Syntax

DROP [PUBLIC] SYNONYM [owner_name.]synonym ;

Arguments

PUBLIC
Specifies that the synonym was created with the PUBLIC argument.

SQL generates the Base table not found error if DROP SYNONYM specifies PUBLIC and the synonym was not a public synonym. Conversely, the same error message occurs if DROP SYNONYM does not specify public and the synonym was created with the PUBLIC argument.

To drop a public synonym, you must have the DBA privilege.

owner_name
If owner_name is specified and is different from the name of the user executing the statement, then the user must have DBA privileges.

synonym
Name for the synonym.

Example

DROP SYNONYM customer ;
DROP PUBLIC SYNONYM public_suppliers ;

Authorization

Users executing DROP SYNONYM must have either the DBA privilege or be the owner of the synonym. Users executing DROP PUBLIC SYNONYM must have the DBA privilege.

SQL Compliance

Extension

Environment
Embedded SQL, interactive SQL, ODBC applications

Related Statements
CREATE SYNONYM
5.9 DROP TABLE

Description

Deletes the specified table.

Syntax

```
DROP TABLE [owner_name.]table_name ;
```

Notes

- If `owner_name` is specified and is different from the name of the user executing the statement, then the user must have DBA privileges.
- When a table is dropped, the indexes on the table and the privileges associated with the table are dropped automatically.
- Views dependent on the dropped table are not automatically dropped, but become invalid.
- If the table is part of another table's referential constraint (if the table is named in another table's REFERENCES clause), the DROP TABLE statement fails. Use the ALTER TABLE statement to delete any referential constraints that refer to the table before issuing the DROP TABLE statement.

Example

```
DROP TABLE customer ;
```

Authorization

The user executing this statement must have any of the following privileges:

- DBA privilege.
- Ownership of the table.

SQL Compliance

- SQL-92, ODBC Minimum SQL grammar

Environment

- Embedded SQL, interactive SQL, ODBC applications

Related Statements

- CREATE TABLE
5.10 DROP VIEW

Description

Deletes the view from the database.

Syntax

DROP VIEW [owner_name.]view_name ;

Notes

If owner_name is specified and is different from the name of the user executing the statement, then the user must have DBA privileges.

When a view is dropped, other views which are dependent on this view are not dropped. The dependent views become invalid.

Example

DROP VIEW newcustomers ;

Authorization

The user executing this statement must have any of the following privileges:

• DBA privilege
• Ownership of the view.

SQL Compliance

SQL-92, ODBC Core SQL grammar

Environment

Embedded SQL, interactive SQL, ODBC applications

Related Statements

CREATE VIEW
5.11 GRANT

Description

Grants various privileges to the specified users for the database. There are three forms of the GRANT statement:

- The first form grants database-wide privileges, either system administration (DBA) or general creation (RESOURCE)
- The second form grants various privileges on the specified tables or view
- The third form grants the privilege to execute the specified stored procedure

Syntax

\[
\text{GRANT} \ {\text{RESOURCE, DBA}} \\\n\text{TO user\_name [ , user\_name ] ... ;}
\]

\[
\text{GRANT} \ {\text{privilege [ , privilege ] ... | ALL [ PRIVILEGES ]}} \\\n\text{ON table\_name} \\\n\text{TO \{ user\_name [ , user\_name ] ... | PUBLIC} \\\n\text{[WITH GRANT OPTION] ;}
\]

\[
\text{GRANT EXECUTE ON procedure\_name} \\\n\text{TO \{ user\_name [ , user\_name ] ... | PUBLIC ;}
\]

\[
\text{privilege ::} \\\n\{ \text{SELECT | INSERT | DELETE | ALTER | INDEX} \\\n\text{| UPDATE [ (column, column, ... ) ]} \\\n\text{| REFERENCES [ (column, column, ... ) ]} \}
\]

Arguments

DBA
Allows the specified users to create, access, modify, or delete any database object, and to grant other users any privileges.

RESOURCE
Allows the specified users to issue CREATE statements. The RESOURCE privilege does not allow users to issue DROP statements on database objects. Only the owner of the object and users with the DBA privilege can drop database objects.

ALTER
Allows the specified users to modify the table or view

DELETE
Allows the specified users to delete rows in the table or view

INDEX
Allows the specified users to create an index on the table or view

INSERT
Allows the specified users to add new rows to the table or view
SELECT
Allows the specified users to read data in the table or view.

UPDATE [ (column, column, ... ) ]
Allows the specified users to modify existing rows in the table or view. If followed by a column list, the users can modify values only in the columns named.

REFERENCES [ (column, column, ... ) ]
Allows the specified users to refer to the table from other tables' constraint definitions. If followed by a column list, constraint definitions can refer only to the columns named. For more detail on constraint definitions, see “Table Constraints” and “Column Constraints”.

ALL
Grants all privileges for the table or view.

ON table_name
The table or view for which SQL grants the specified privileges.

EXECUTE ON procedure_name
Allows execution of the specified stored procedure.

TO user_name [ , user_name ] ...
The list of users for which SQL grants the specified privileges.

TO PUBLIC
Grants the specified privileges to any user with access to the system.

WITH GRANT OPTION
Allows the specified users to grant their access rights or a subset of their rights to other users.

Example

GRANT ALTER ON cust_view TO dbuser1;

GRANT SELECT ON newcustomers TO dbuser2;

GRANT EXECUTE ON sample_proc TO searle;

Authorization

The user granting DBA or RESOURCE privileges must have the DBA privilege.
The user granting privileges on a table must have any of the following privileges:

- DBA privilege
- Ownership of the table
- All the specified privileges on the table, granted with the WITH GRANT OPTION clause
Overview

**SQL Compliance**  
SQL-92, ODBC Core SQL grammar. Extensions: ALTER, INDEX, RESOURCE, DBA privileges

**Environment**  
Embedded SQL, interactive SQL, ODBC applications

**Related Statements**  
REVOKE
5.12 INSERT

Description

Inserts new rows into the specified table/view that will contain either the explicitly specified values or the values returned by the query expression.

Syntax

```
INSERT INTO [owner_name.] { table_name | view_name }
    [ (column_name, column_name, ...) ]
    { VALUES (value, value, ...) | query_expression };
```

Notes

- If the optional list of column names is specified, then only the values for those columns need be supplied. The rest of the columns of the inserted row will contain NULL values, provided the table definition allows NULL values and there is no DEFAULT clause for the columns. If a DEFAULT clause is specified for a column and the column name is not present in the optional column list, then the column takes the default value.

- If the optional list is not specified then all the column values have to be either explicitly specified or returned by the query expression. The order of the values should be the same as the order in which the columns have been declared in the declaration of the table/view.

- Explicit specification of the column values provides for insertion of only one row at a time. The query expression option allows for insertion of multiple rows at a time.

- If the table contains a foreign key, and there does not exist a corresponding primary key that matches the values of the foreign key in the record being inserted, the insert operation is rejected.

Examples

```
INSERT INTO customer (cust_no, name, street, city, state)
VALUES
    (1001, 'RALPH', '#10 Columbia Street', 'New York', 'NY') ;

INSERT INTO neworders (order_no, product, qty)
SELECT order_no, product, qty
FROM orders
WHERE order_date = SYSDATE ;
```

Authorization

The user executing this statement must have any of the following privileges:

- DBA privilege.
- Ownership of the table.
- INSERT privilege on the table.

If a query_expression is specified, then the user must have any of the following privileges:
Overview

- DBA privilege.
- SELECT privilege on all the tables/views referred to in the query_expression.

SQL Compliance
- SQL-92, ODBC Core SQL grammar

Environment
- Embedded SQL, interactive SQL, ODBC applications

Related Statements
- Query Expressions
5.13 RENAME

Description
Renames the specified table name, view name or synonym to the new name specified

Syntax
RENAME [owner_name.] oldname TO [owner_name.] newname ;

Arguments
[owner_name.]
Optional owner-name qualifier for the name. If the owner name is not the same as that of the current user, the current user must have the DBA privilege.

If specified, the owner name must be the same for oldname and newname. In other words, you cannot change the owner of a table, view, or synonym with RENAME.

oldname
Current name of the table, view, or synonym.

newname
New name for the table, view, or synonym.

Example
RENAME sitem TO supplier_item ;

Authorization
The user executing this statement must have any of the following privileges:

• DBA privilege
• Ownership of the table/view/synonym.
• ALTER privilege on the table/view.

SQL Compliance
Extension

Environment
Embedded SQL, interactive SQL, ODBC applications

Related Statements
CREATE TABLE, CREATE VIEW, CREATE SYNONYM
5.14 REVOKE

Description

Revokes various privileges to the specified users for the database. There are three forms of the REVOKE statement:

- The first form revokes database-wide privileges, either system administration (DBA) or general creation (RESOURCE)
- The second form revokes various privileges on specific tables and views
- The third form revokes the privilege to execute the specified stored procedure

Syntax

REVOKE { RESOURCE | DBA }
    FROM { user_name [ , user_name ] ... } ;

REVOKE [ GRANT OPTION FOR ]
    { privilege [ , privilege, ] ... | ALL [ PRIVILEGES ] }
    ON table_name
    FROM { user_name [ , user_name ] ... | PUBLIC } [ RESTRICT | CASCADE ] ;

REVOKE [ GRANT OPTION FOR ] EXECUTE ON procedure_name
    FROM { user_name [ , user_name ] ... | PUBLIC } [ RESTRICT | CASCADE ] ;

privilege :: [
    SELECT | INSERT | DELETE | ALTER | INDEX |
    UPDATE [ (column, column, ... ) ] |
    REFERENCES [ (column, column, ... ) ]
]

Arguments

GRANT OPTION FOR

Revokes the grant option for the privilege from the specified users. The actual privilege itself is not revoked. If specified with RESTRICT, and the privilege was passed on to other users, the REVOKE statement fails and generates an error. Otherwise, GRANT OPTION FOR implicitly revokes any rights the user may have in turn given to other users.

{ privilege [ , privilege, ] ... | ALL [ PRIVILEGES ] }

List of privileges to be revoked. See the description in the GRANT statement for details on specific privileges. Revoking RESOURCE and DBA rights can only be done by the administrator or a user with DBA rights.

If a user has been granted access to a table by more than one user then all the users have to perform a revoke for the user to lose his access to the table.

Using the keyword ALL revokes all the rights granted on the table/view.
ON table_name
The table or view for which SQL revokes the specified privileges.

EXECUTE ON procedure_name
Revokes the right to execute the specified stored procedure.

FROM user_name [, user_name] ...
Revokes the specified rights on the table or view from the specified list of users.

FROM PUBLIC
Revokes the specified rights on the table or view from any user with access to the system.

RESTRICT | CASCADE
If the REVOKE statement specifies RESTRICT, SQL checks to see if the privilege being revoked was passed on to other users (possible only if the original privilege included the WITH GRANT OPTION clause). If so, the REVOKE statement fails and generates an error. If the privilege was not passed on, the REVOKE statement succeeds.

If the REVOKE statement specifies CASCADE, revoking the access right of a user also revokes the rights from all users who received the privilege as a result of that user giving the privilege to others.

If the REVOKE statement specifies neither RESTRICT nor CASCADE, the behavior is the same as for CASCADE.

Example

REVOKE INSERT ON customer FROM dbuser1;
REVOKE ALTER ON cust_view FROM dbuser2;

Authorization

The user revoking DBA or RESOURCE privileges must have the DBA privilege.

The user revoking privileges on a table must have any of the following privileges:

- DBA privilege
- Ownership of the table
- All the specified privileges on the table, granted with the WITH GRANT OPTION clause

SQL Compliance

SQL-92, ODBC Core SQL grammar. Extensions: ALTER, INDEX, RESOURCE, DBA privileges

Environment

Embedded SQL, interactive SQL, ODBC applications

Related Statements

GRANT
5.15 SELECT

Description

Selects the specified column values from one or more rows contained in the table(s) specified in the FROM clause. The selection of rows is restricted by the WHERE clause. The temporary table derived through the clauses of a select statement is called a result table.

The format of the SELECT statement is a query expression with optional ORDER BY and FOR UPDATE clauses. For more detail on query expressions, see "Query Expressions".

Syntax

```
select_statement ::= 
  query_expression
  ORDER BY { expr | posn } [ COLLATE collation_name ] [ ASC | DESC ]
  [ , { expr | posn } [ COLLATE collation_name ] [ASC | DESC] ,... ]
  FOR UPDATE [ OF [table].column_name, ... ] [ NOWAIT ]

query_expression ::= 
  query_specification
  | query_expression set_operator query_expression
  | ( query_expression )

set_operator ::= 
  { UNION [ ALL ] | INTERSECT | MINUS }

query_specification ::= 
  SELECT [ALL | DISTINCT]
  { *
    | { table_name | alias } . * [ , { table_name | alias } . * ] ... 
    | expr [ [ AS ] [ ' ] column_title [ ' ] ] [ , expr [ [ AS ] [ ' ] column_title [ ' ] ] ] ... 
  }
  FROM table_ref [ ( dharma ORDERED ) ] [ , table_ref [ ( dharma ORDERED ) ] ] ...
  [ WHERE search_condition ]
  [ GROUP BY [table.]column_name [ COLLATE collation-name ]
    [ , [table.]column_name [ COLLATE collation-name ] ] ... 
  [ HAVING search_condition ]

table_ref ::= 
  table_name [ AS ] [ alias [ ( column_alias [ , ... ] ) ] ]
  | ( query_expression ) [ AS ] alias [ ( column_alias [ , ... ] ) ]
  | [ ( ] joined_table [ ) ]

joined_table ::= 
  table_ref CROSS JOIN table_ref
  | table_ref [ INNER | LEFT [ OUTER ] ] JOIN table_ref ON search_condition
```
Arguments

query_expression
See Section 4.4.

ORDER BY clause
See Section 5.15.1.

FOR UPDATE clause
See Section 5.15.2.

Authorization

The user executing this statement must have any of the following privileges:

- DBA privilege
- SELECT permission on all the tables/views referred to in the query_expression.

SQL Compliance


Environment

Embedded SQL (within DECLARE), interactive SQL, ODBC applications

Related Statements

Query Expressions, DECLARE CURSOR, OPEN, FETCH, CLOSE

5.15.1 ORDER BY Clause

Description

The ORDER BY clause specifies the sorting of rows retrieved by the SELECT statement. SQL does not guarantee the sort order of rows unless the SELECT statement includes an ORDER BY clause.

Syntax

ORDER BY \{ expr | posn \} \[ COLLATE collation_name \] \[ ASC | DESC \] \[ , \{ expr | posn \} \[ COLLATE collation_name \] [ASC | DESC] ,... \]

Notes

- Ascending order is the default ordering. The descending order will be used only if the keyword DESC is specified for that column.

- Each expr is an expression of one or more columns of the tables specified in the FROM clause of the SELECT statement. Each posn is a number identifying the column position of the columns being selected by the SELECT statement.

- The selected rows are ordered on the basis of the first expr or posn and if the values are the same then the second expr or posn is used in the ordering.

- The ORDER BY clause if specified should follow all other clauses of the SELECT statement.

- A query expression followed by an optional ORDER BY clause can be specified. In such a case, if the query expression contains set operators, then the ORDER BY clause can specify only the positions. For example:
-- Get a merged list of customers and suppliers
-- sorted by their name.
(SELECT name, street, state, zip
 FROM customer
UNION
 SELECT name, street, state, zip
 FROM supplier)
ORDER BY 1 ;

• If expr or posn refers to a character column, the reference can include an optional COLLATE clause. The COLLATE clause specifies a collation sequence supported by the underlying storage system. (See “Specifying the Character Set for Character Data Types” on page 4-6 for notes on character sets and collations. See the documentation for your underlying storage system for details on any supported collations.)

Example

SELECT name, street, city, state, zip
 FROM customer
 ORDER BY name ;

5.15.2 FOR UPDATE Clause

Description

The FOR UPDATE clause specifies update intention on the rows selected by the SELECT statement.

Syntax

FOR UPDATE [ OF [table].column_name, ... ] [ NOWAIT ] Notes

• If FOR UPDATE clause is specified, WRITE locks are acquired on all the rows selected by the SELECT statement.

• If NOWAIT is specified, an error is returned when a lock cannot be acquired on a row in the selection set because of the lock held by some other transaction. Otherwise, the transaction would wait until it gets the required lock or until it times out waiting for the lock.
5.16 SET SCHEMA

Description

SET SCHEMA specifies a new default qualifier for database object names. (Database objects include tables, indexes, views, synonyms, procedures, and triggers.)

When you connect to a database with a particular user name, that name becomes the default qualifier for database object names. This means you do not have to qualify references to tables, for instance, that were created under the same user name. However, you must qualify references to all other tables with the user name of the user who created them.

SET SCHEMA allows you to change the user name that SQL uses as the default qualifier for database object names. The name specified in SET SCHEMA becomes the new default qualifier for object names.

Note

SET SCHEMA does not change your user name or affect authentication. It only changes the default qualifier.

Syntax

SET SCHEMA ' qualifier_name ' ;

Arguments

' qualifier_name '

The new qualifier name, enclosed in single quotation marks.

Notes

- SET SCHEMA does not check whether qualifier_name is a valid user name.
- Metadata for objects created without an explicit qualifier will show qualifier_name as the owner.
- SET SCHEMA does not start or end a transaction.

Examples

The following interactive SQL example shows changing the default qualifier through SET SCHEMA. The example:

- Invokes ISQL as the user syste, the owner of the system catalog tables
- Queries the systables catalog tables as syste
- Uses SET SCHEMA to change the default qualifier to fred
- Creates a table and queries systables to show that the newly-created table is owned by fred
ISQL> -- What is the user name for the current connection?
ISQL> select user() from syscalctable;
SYSTPE
-------
systpe
1 record selected

ISQL> -- Show the name and owner of non-system tables:
ISQL> select tbl, owner from systables where tbltype <> 'S';

<table>
<thead>
<tr>
<th>TBL</th>
<th>OWNER</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1</td>
<td>systpe</td>
</tr>
<tr>
<td>test</td>
<td>systpe</td>
</tr>
<tr>
<td>test</td>
<td>dharma</td>
</tr>
</tbody>
</table>

3 records selected

ISQL> set schema 'fred';
ISQL> create table freds_table (c1 int);
ISQL> create index freds_table_ix on freds_table (c1);
ISQL> select tbl, owner from systables where tbltype <> 'S';

error(-20005): Table/View/Synonym not found

ISQL> -- Oops! Must now qualify references to the systpe-owned tables:
ISQL> select tbl, owner from systpe.systables where tbltype <> 'S';

<table>
<thead>
<tr>
<th>TBL</th>
<th>OWNER</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1</td>
<td>systpe</td>
</tr>
<tr>
<td>test</td>
<td>systpe</td>
</tr>
<tr>
<td>test</td>
<td>dharma</td>
</tr>
<tr>
<td>freds_table</td>
<td>fred</td>
</tr>
</tbody>
</table>

4 records selected

Authorization
None.

SQL Compliance  SQL-92

Environment  Embedded SQL and interactive

Related Statements  None
5.17 UPDATE

Description

Updates the columns of the specified table with the given values that satisfy the search_condition.

Syntax

UPDATE table_name
  SET assignment, assignment, ...
  [ WHERE search_condition ]

assignment ::
  column = { expr | NULL }
  | ( column, column, ... ) = ( expr, expr, ... )
  | ( column, column, ... ) = ( query_expression )

Arguments

- If the optional WHERE clause is specified, then only rows that satisfy the search_condition are updated. If the WHERE clause is not specified then all rows of the table are updated.
- The expressions in the SET clause are evaluated for each row of the table if they are dependent on the columns of the target table.
- If a query expression is specified on the right hand side for an assignment, the number of expressions in the first SELECT clause of the query expression must be the same as the number of columns listed on the left hand side of the assignment.
- If a query expression is specified on the right hand side for an assignment, the query expression must return one row.
- If a table has check constraints and if the columns to be updated are part of a check expression, then the check expression is evaluated. If the result of evaluation is FALSE, the UPDATE statement fails.
- If a table has primary/ candidate keys and if the columns to be updated are part of the primary/ candidate key, a check is made as to whether there exists any corresponding row in the referencing table. If so, the UPDATE operation fails.

Examples

UPDATE orders
  SET qty = 12000
  WHERE order_no = 1001 ;

UPDATE orders
  SET (product) =
    (SELECT item_name
     FROM items
     WHERE item_no = 2401 )
  WHERE order_no = 1002 ;
Overview

UPDATE orders
  SET (amount) = (2000 * 30)
  WHERE order_no = 1004 ;

UPDATE orders
  SET (product, amount) =
      (SELECT item_name, price * 30
       FROM items
       WHERE item_no = 2401
      )
  WHERE order_no = 1002 ;

Authorization

The user executing this statement must have:

- DBA privilege.
- UPDATE privilege on all the specified columns of the target table and SELECT privilege on all the other tables referred to in the statement.

SQL Compliance

SQL-92, ODBC Extended SQL grammar. Extensions: assignments of the form (column, column, ...) = (expr, expr, ...)

Environment

Embedded SQL, interactive SQL, ODBC applications

Related Statements

SELECT, OPEN, FETCH, search conditions, query expressions
Overview

This sections contains reference information on utilities used to configure the ODBC Server.

- The dhdaemon executable image starts the ODBC server and enables network access from clients.
- On Windows NT, the pcntreg utility registers the dhdaemon executable image as a service in the system registry.
- mdsql loads metadata into the data dictionary and provides a simple, general-purpose SQL interface on the server.
- mdcreate creates a data dictionary and provides a name for access to the proprietary storage system.

dhdaemon

The dhdaemon executable image starts the ODBC server and enables network access from clients:

- On UNIX, dhdaemon is the only way to start the server process.
- On Windows NT, dhdaemon is an alternative to starting the server process as a service. This alternative allows options not available through starting the server as a service, which can be useful for using the debugging implementations.
Syntax

dhdaemon [ option [ option ... ] ] [ start | stop | status ]

option ::
   -c
   -e server name
   -s service name
   -q

Arguments

-c
On Windows NT, starts the server as a console application. This approach allows you to use debugging tools and allows user-level environment variables (such as TPESQLDBG) to affect the dhserver process. (When started as a service, the dhserver process only sees system environment variables.) The -c option is applicable only to Windows NT, and required there to start the server from the command line.

-e server name
The name of the executable to use for the ODBC Server process. For example, use the -e option to specify the sample implementation executable demo as the ODBC Server process:

$ dhdaemon -e ~/dharma/bin/dhdemo start

-s service name
The name of a network service in the services file. If the dhdaemon command does not include the -s option, the default is sqlnw.

-q
Starts the dhdaemon process in "quiet mode", which displays fewer messages.

start
Starts the dhdaemon process.

stop
Stops the dhdaemon process.

status
Displays the status of the process and any child processes it has spawned. For example:

$ dhdaemon status
Dharma/dhdaemon Version 6.20.1000
Dharma Systems Inc (C) 1988-99.
Dharma Computers Pvt Ltd (C) 1988-99.

Daemon version: Jun 18 1999 18:18:18
running since: 6/19/99 13:51:27 on isis
Working directory: /u11/v5_work_sdk/dbp.dbs
SQL-Server version: /u11/v5_work_sdk/bin/demo
Nr of servers started: 5
running: 0
crashed: 0
**pcntreg**

Adds and deletes entries for the ODBC SDK in the Windows NT registry.

___________________________________________________________________________

**Note**

The pcntreg utility is only applicable to Windows NT.

___________________________________________________________________________

Syntax

```
pcntreg { p path | d }
```

Arguments

**p path**

Register dhdaemon. The path argument specifies the disk and directory name for the top-level dharmadir directory (for example, C:dharma).

**d**

Delete the registry entry for dhdaemon.

---

**mdcreate**

Creates a data dictionary that stores metadata (details on the structure of SQL tables and indexes).

Syntax

```
mdcreate [ -v ] [ -d directory_spec ] dbname
```

Arguments

**-v**

Specifies verbose mode, so mdcreate generates detailed status messages.

**-d directory_spec**

Specifies an alternative directory specification in which to create the data dictionary. This argument is valid only for the Desktop configuration.

The mdcreate utility creates a subdirectory to contain the data dictionary files. It uses the name specified in the dbname argument for the subdirectory. There are three levels of defaults that determine where mdcreate creates this subdirectory:

- The directory specified by the -d argument
- If the mdcreate does not specify -d, the directory specified by the TPE_DATADIR environment variable
- If TPE_DATADIR is not set, mdcreate creates the dbname subdirectory under the directory specified by the TPEROOT directory.

For example:

```
$ C:dharma\bin\mdcreate -d E:databases demo_db
```
This command creates a subdirectory called \texttt{demo\_db.dbs} under the \texttt{E:\databases} directory and populates the directory with the necessary files.

Once you create the database subdirectory in this manner, you must explicitly specify its location in \texttt{mdsql} command lines and when you add ODBC data source names:

- In \texttt{mdsql}, use the `-d` option to specify the same directory path as you used for \texttt{mdcreate}. For example:
  
  $ \texttt{mdsql -s C:\dharm\odbc\dk\sample\md\template -d E:\databases demo\_db}$

- In the Microsoft ODBC Administrator utility, the ODBC Setup dialog box contains a Data Dir text-box field. Use it to specify the same directory path as you used for \texttt{mdcreate}.

\textit{dbname}

The name of the database. ODBC applications and the \texttt{mdsql} utility specify \textit{dbname} to access the database.

---

\textbf{\textit{mdsql}}

The primary use for \texttt{mdsql} is to load metadata into data dictionaries via an SQL script, which contains CREATE TABLE and INDEX statements with the STORAGE_ATTRIBUTES 'METADATA ONLY' clause. This clause directs the SQL engine to insert metadata into the data dictionary without requiring the proprietary storage system to create an empty table or index. The table or index name used in the CREATE statement must be the same as an existing table or index in the proprietary storage system.

You can also use \texttt{mdsql} to create new tables or issue SQL queries interactively. Invoke it without the `-s` option and specify the database you want to access. Terminate statements with a semicolon. To exit from interactive `mdsql`, type CTRL/D.

\textbf{Syntax}

\texttt{mdsql [-s script\_file] [-u user\_name] [-a password] [-d directory\_spec ] dbname}

\textbf{Arguments}

- \texttt{-s script\_file}
  The name of an SQL script file \texttt{mdsql} executes.

- \texttt{-u user\_name}
  The user name to connect to the database specified. The default is the current user of the operating system. Unless you log in as dharma, you should specify `-u dharma` on the \texttt{mdsql} command line.

- \texttt{-a password}
  The password to connect to the database specified. The default is null.
-d directory_spec
A n alternative location for the data dictionary directory. This argument is valid only for the Desktop configuration. If the mdcreate utility specified the -d argument, mdsql must specify the same argument (or the TPE_DATADIR environment variable should specify directory_spec).

dbname
The name of the database, as specified to the mdcreate utility.
Reserved Keywords

Reserved words are keywords you can use as identifiers in SQL statements if you delimit them with double quotation marks. If you use keywords without delimiting them, the statement generates one of the following errors:

error(-20003): Syntax error
error(-20049): Keyword used for a name

The following table lists reserved words. The list is alphabetic and reads left to right.

<table>
<thead>
<tr>
<th>A</th>
<th>ABS</th>
<th>ACOS</th>
<th>ADD</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD_MONTHS</td>
<td>AFTER</td>
<td>ALL</td>
<td>ALTER</td>
</tr>
<tr>
<td>AN</td>
<td>AND</td>
<td>ANY</td>
<td>ARRAY</td>
</tr>
<tr>
<td>AS</td>
<td>ASC</td>
<td>ASCII</td>
<td>ASIN</td>
</tr>
<tr>
<td>ATAN</td>
<td>ATAN2</td>
<td>AVG</td>
<td>BEFORE</td>
</tr>
<tr>
<td>BEGIN</td>
<td>BETWEEN</td>
<td>BIGINT</td>
<td>BINARY</td>
</tr>
<tr>
<td>BIND</td>
<td>BINDING</td>
<td>BIT</td>
<td>BY</td>
</tr>
<tr>
<td>CALL</td>
<td>CASCADE</td>
<td>CASE</td>
<td>CAST</td>
</tr>
<tr>
<td>CEILING</td>
<td>CHAR</td>
<td>CHAR_LENGTH</td>
<td>CHARACTER</td>
</tr>
<tr>
<td>CHARACTER_LENGTH</td>
<td>CHARTOROWID</td>
<td>CHECK</td>
<td>CHR</td>
</tr>
<tr>
<td>CLEANUP</td>
<td>CLOSE</td>
<td>CLUSTERED</td>
<td>COALESCE</td>
</tr>
<tr>
<td>COLGROUP</td>
<td>COLLATE</td>
<td>COLUMN</td>
<td>COMMIT</td>
</tr>
<tr>
<td>COMPLEX</td>
<td>COMPRESS</td>
<td>CONCAT</td>
<td>CONNECT</td>
</tr>
<tr>
<td>CONSTRAINT</td>
<td>CONTAINS</td>
<td>CONTINUE</td>
<td>CONVERT</td>
</tr>
<tr>
<td>COS</td>
<td>COUNT</td>
<td>CREATE</td>
<td>CROSS</td>
</tr>
<tr>
<td>CURDATE</td>
<td>CURRENT</td>
<td>CURSOR</td>
<td>CURTIME</td>
</tr>
<tr>
<td>CVAR</td>
<td>DATABASE</td>
<td>DATAPAGES</td>
<td>DATE</td>
</tr>
<tr>
<td>Reserved Keywords</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-------------------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DAYNAME</td>
<td>DAYOFMONTH</td>
<td>DAYOFYEAR</td>
<td></td>
</tr>
<tr>
<td>DB_NAME</td>
<td>DBA</td>
<td>DEC</td>
<td>DECIMAL</td>
</tr>
<tr>
<td>DECLARATION</td>
<td>DECLARE</td>
<td>DECODE</td>
<td>DEFAULT</td>
</tr>
<tr>
<td>DEFINITION</td>
<td>DEGREES</td>
<td>DELETE</td>
<td>DESC</td>
</tr>
<tr>
<td>DESCRIBE</td>
<td>DESCRIPTOR</td>
<td>DTYPE</td>
<td>DIFFERENCE</td>
</tr>
<tr>
<td>DISTINCT</td>
<td>DOUBLE</td>
<td>DROP</td>
<td>EACH</td>
</tr>
<tr>
<td>ELSE</td>
<td>END</td>
<td>END</td>
<td>ESCAPE</td>
</tr>
<tr>
<td>EXCLUSIVE</td>
<td>EXEC</td>
<td>EXECUTE</td>
<td>EXISTS</td>
</tr>
<tr>
<td>EXIT</td>
<td>EXP</td>
<td>EXPlicit</td>
<td>FETCH</td>
</tr>
<tr>
<td>FIELD</td>
<td>FILE</td>
<td>FLOAT</td>
<td>FLOOR</td>
</tr>
<tr>
<td>FOR</td>
<td>FOREIGN</td>
<td>FOUND</td>
<td>FROM</td>
</tr>
<tr>
<td>FULL</td>
<td>GO</td>
<td>GOTO</td>
<td>GRANT</td>
</tr>
<tr>
<td>GREATEST</td>
<td>GROUP</td>
<td>HASH</td>
<td>HAVING</td>
</tr>
<tr>
<td>HOUR</td>
<td>IDENTIFIED</td>
<td>IFNULL</td>
<td>IMMEDIATE</td>
</tr>
<tr>
<td>IN</td>
<td>INDEX</td>
<td>INDEXPAGES</td>
<td>INDICATOR</td>
</tr>
<tr>
<td>INITCAP</td>
<td>INNER</td>
<td>INOUT</td>
<td>INPUT</td>
</tr>
<tr>
<td>INSERT</td>
<td>INSTR</td>
<td>INT</td>
<td>INTEGER</td>
</tr>
<tr>
<td>INTERFACE</td>
<td>INTERSECT</td>
<td>INTO</td>
<td>IS</td>
</tr>
<tr>
<td>JOIN</td>
<td>KEY</td>
<td>LAST_DAY</td>
<td>LCASE</td>
</tr>
<tr>
<td>LEAST</td>
<td>LEFT</td>
<td>LEFT</td>
<td>LENGTH</td>
</tr>
<tr>
<td>LIKE</td>
<td>LINK</td>
<td>LIST</td>
<td>LOCATE</td>
</tr>
<tr>
<td>LOCK</td>
<td>LOG</td>
<td>LOG10</td>
<td>LONG</td>
</tr>
<tr>
<td>LONG</td>
<td>LOWER</td>
<td>LPAD</td>
<td>LTRIM</td>
</tr>
<tr>
<td>LVARCHAR</td>
<td>MAIN</td>
<td>MAX</td>
<td></td>
</tr>
<tr>
<td>METADATA_ONLY</td>
<td>MIN</td>
<td>MINUS</td>
<td>MINUTE</td>
</tr>
<tr>
<td>MOD</td>
<td>MODE</td>
<td>MODIFY</td>
<td>MONEY</td>
</tr>
<tr>
<td>MONTH</td>
<td>MONTHNAME</td>
<td>MONTHS_BETWEEN</td>
<td>NATIONAL</td>
</tr>
<tr>
<td>NATURAL</td>
<td>NCHAR</td>
<td>NEWROW</td>
<td>NEXT_DAY</td>
</tr>
<tr>
<td>NOCOMPRESS</td>
<td>NOT</td>
<td>NOW</td>
<td>NOWAIT</td>
</tr>
<tr>
<td>NULL</td>
<td>NULLIF</td>
<td>NULLVALUE</td>
<td>NUMBER</td>
</tr>
<tr>
<td>NUMERIC</td>
<td>NVL</td>
<td>OBJECT_ID</td>
<td>ODBC_CONVERT</td>
</tr>
<tr>
<td>ODBCINFO</td>
<td>OF</td>
<td>OLDROW</td>
<td>ON</td>
</tr>
<tr>
<td>OPEN</td>
<td>OPTION</td>
<td>OR</td>
<td>ORDER</td>
</tr>
<tr>
<td>OUT</td>
<td>OUTER</td>
<td>OUTPUT</td>
<td>PCTFREE</td>
</tr>
<tr>
<td>PI</td>
<td>POWER</td>
<td>PRECISION</td>
<td>PREFIX</td>
</tr>
<tr>
<td>Reserved Keywords</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------------------------------------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PREPARE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PUBLIC</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RANGE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REFERENCES</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REPLACE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RETURN</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ROLLBACK</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ROWNUM</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SECOND</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SET</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SIMPLE_CASE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SOME</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SQL_BIGINT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SQL_DATE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SQL_INTEGER</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SQL_REAL</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SQL_TINYINT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SQL_TSI_MINUTE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SQL_TSI_WEEK</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SQLERROR</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STATEMENT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STORAGE_MANAGER</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SUFFIX</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SYSDATE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TAN</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TO</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TO_TIME</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TRANSLATE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UID</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UPDATE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>USER_NAME</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VARBINARY</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VERSION</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WHenever</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PRIMARY</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>QUARTER</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PRIVILEGES</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RADIANS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REAL</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RENAME</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RESULT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RIGHT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ROWID</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ROWIDTOCHAR</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RTRIM</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SEARCHED_CASE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SELECT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SERVICE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SHARE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SHORT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SIZE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SMALLINT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SOUNDEX</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SPACE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SQL</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SQL_BIGINT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SQL_BIT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SQL_CHAR</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SQL_DECIMAL</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SQL_DOUBLE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SQL_FLOAT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SQL_LONGVARBINARY</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SQL_LONGVARCHAR</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SQL_TIME</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SQL_TIMESTAMP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SQL_TSI_HOUR</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SQL_TSI_HOUR</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SQL_TSI_DAY</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SQL_TSI_FRACTIONSECOND</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SQL_TSI_MONTH</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SQL_TSI_QUARTER</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SQL_TSI_WEEK</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SQL_VARBINARY</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SQRT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>START</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STORE_IN_DHARMA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SUM</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SUBSTR</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SUBSTRING</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SYSTIME</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SYSTIME_TIMESTAMP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TABLE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TINYINT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIMESTAMPADD</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TO_CHAR</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TO_DATE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TO_NUMBER</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TO_NUMBER</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TO_TIMESTAMP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TPE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TRANSACTION</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UCASE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UNIQUE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>USER</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>USER_ID</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VALUES</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VARYING</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WEEK</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WHEN</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WORK</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Reserved Keywords

YEAR
Overview

This appendix lists the error messages generated by the various components of the Dharma DataLink SDK.

You can receive error messages not only from the SQL engine, but also from underlying storage systems, including the storage system Dharma supplies for use as a data dictionary.

Storage systems on which the ODBC SDK is implemented will likely generate their own error messages. See the documentation for your storage system for details.

Error Codes, SQLSTATE Values, and Messages

Table C–1 lists the Dharma error messages, ordered by error code number.

Table C–1  Error Codes and Messages

<table>
<thead>
<tr>
<th>Error Code</th>
<th>SQLSTATE Value</th>
<th>Class Condition</th>
<th>Subclass Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000</td>
<td>00000</td>
<td>Successful completion</td>
<td>***status okay</td>
</tr>
<tr>
<td>100L</td>
<td>02000</td>
<td>no data</td>
<td>**sql not found.</td>
</tr>
<tr>
<td>10002</td>
<td>02503</td>
<td>No data</td>
<td>Tuple not found for the specified TID.</td>
</tr>
<tr>
<td>10012</td>
<td>n0n12</td>
<td>flag</td>
<td>ETPL_SCAN_EOP flag is set.</td>
</tr>
<tr>
<td>10013</td>
<td>02514</td>
<td>No data</td>
<td>No more records to be fetched.</td>
</tr>
<tr>
<td>10100</td>
<td>2150b</td>
<td>Cardinality violation</td>
<td>Too many fields specified.</td>
</tr>
<tr>
<td>10101</td>
<td>02701</td>
<td>No data</td>
<td>No more records exists.</td>
</tr>
<tr>
<td>10102</td>
<td>5050c</td>
<td>Dharma/ SQL rds error</td>
<td>Duplicate record specified.</td>
</tr>
<tr>
<td>Error Code</td>
<td>SQLSTATE Value</td>
<td>Class Condition</td>
<td>Subclass Message</td>
</tr>
<tr>
<td>------------</td>
<td>----------------</td>
<td>-----------------</td>
<td>------------------</td>
</tr>
<tr>
<td>10104</td>
<td>22505</td>
<td>Data exception</td>
<td>field size is too high.</td>
</tr>
<tr>
<td>10106</td>
<td>m0m06</td>
<td>Dharma/SQL rss error</td>
<td>Specified index method is not supported.</td>
</tr>
<tr>
<td>10107</td>
<td>n0n07</td>
<td>flag</td>
<td>EIX_SCAN_EOP flag is set.</td>
</tr>
<tr>
<td>10108</td>
<td>2350i</td>
<td>Integrity constraint</td>
<td>Duplicate primary / index key value.</td>
</tr>
<tr>
<td>10301</td>
<td>m030a</td>
<td>Dharma/SQL rss error</td>
<td>Table is locked and LCKF_NOWAIT.</td>
</tr>
<tr>
<td>10400</td>
<td>22501</td>
<td>Data exception</td>
<td>Invalid file size for alter log statement.</td>
</tr>
<tr>
<td>10920</td>
<td>22521</td>
<td>Data exception</td>
<td>Already existing value specified.</td>
</tr>
<tr>
<td>11100</td>
<td>5050b</td>
<td>Dharma/SQL rds error</td>
<td>Invalid transaction id.</td>
</tr>
<tr>
<td>11102</td>
<td>5050d</td>
<td>Dharma/SQL rds error</td>
<td>TDS area specified is not found.</td>
</tr>
<tr>
<td>11103</td>
<td>50504</td>
<td>Dharma/SQL rds error</td>
<td>TDS not found for binding.</td>
</tr>
<tr>
<td>11104</td>
<td>50505</td>
<td>Dharma/SQL rds error</td>
<td>Transaction aborted.</td>
</tr>
<tr>
<td>11105</td>
<td>50506</td>
<td>Dharma/SQL rds error</td>
<td>Active Transaction error.</td>
</tr>
<tr>
<td>11109</td>
<td>50510</td>
<td>Dharma/SQL rds error</td>
<td>Invalid Transaction handle.</td>
</tr>
<tr>
<td>11111</td>
<td>50912</td>
<td>Dharma/SQL rds error</td>
<td>Invalid isolation level.</td>
</tr>
<tr>
<td>11300</td>
<td>m0m00</td>
<td>Dharma/SQL rss error</td>
<td>Specified INFO type is not supported.</td>
</tr>
<tr>
<td>11301</td>
<td>m0m01</td>
<td>Dharma/SQL rss error</td>
<td>Specified index type is not supported.</td>
</tr>
<tr>
<td>15001</td>
<td>60601</td>
<td>dharma/SQL ff errors</td>
<td>FF- File IO error</td>
</tr>
<tr>
<td>15002</td>
<td>60602</td>
<td>dharma/SQL ff errors</td>
<td>FF- No more records</td>
</tr>
<tr>
<td>15003</td>
<td>42603</td>
<td>Access violation error</td>
<td>FF- Table already exists</td>
</tr>
<tr>
<td>15004</td>
<td>22604</td>
<td>Data exception</td>
<td>FF- Invalid record number</td>
</tr>
<tr>
<td>15005</td>
<td>60605</td>
<td>dharma/SQL ff errors</td>
<td>FF- Record deleted</td>
</tr>
<tr>
<td>15006</td>
<td>60606</td>
<td>dharma/SQL ff errors</td>
<td>FF- Invalid type</td>
</tr>
<tr>
<td>15007</td>
<td>60607</td>
<td>dharma/SQL ff errors</td>
<td>FF- Duplicate value</td>
</tr>
<tr>
<td>15008</td>
<td>08608</td>
<td>Connection exception</td>
<td>FF- Database exists</td>
</tr>
<tr>
<td>15009</td>
<td>08609</td>
<td>Connection exception</td>
<td>FF- No database found</td>
</tr>
<tr>
<td>15010</td>
<td>60610</td>
<td>dharma/SQL ff errors</td>
<td>FF- Version mis-match</td>
</tr>
<tr>
<td>15011</td>
<td>60611</td>
<td>dharma/SQL ff errors</td>
<td>FF- Virtual file cache exceeded</td>
</tr>
<tr>
<td>15012</td>
<td>60612</td>
<td>dharma/SQL ff errors</td>
<td>FF- Physical file open error</td>
</tr>
<tr>
<td>15013</td>
<td>60613</td>
<td>dharma/SQL ff errors</td>
<td>FF- Corrupt virtual file handle</td>
</tr>
<tr>
<td>15014</td>
<td>22614</td>
<td>Data exception</td>
<td>FF- Overflow error</td>
</tr>
<tr>
<td>15021</td>
<td>60615</td>
<td>dharma/SQL ff errors</td>
<td>FF- dbm_calls not implemented</td>
</tr>
<tr>
<td>16001</td>
<td>22701</td>
<td>Data exception</td>
<td>MM- No data block</td>
</tr>
<tr>
<td>Error Code</td>
<td>SQLSTATE Value</td>
<td>Class Condition</td>
<td>Subclass Message</td>
</tr>
<tr>
<td>------------</td>
<td>----------------</td>
<td>-----------------</td>
<td>--------------------------------</td>
</tr>
<tr>
<td>16002</td>
<td>70702</td>
<td>dharma/ SQL MM errors</td>
<td>MM- Bad swap block</td>
</tr>
<tr>
<td>16003</td>
<td>70703</td>
<td>dharma/ SQL MM errors</td>
<td>MM- No cache block</td>
</tr>
<tr>
<td>16004</td>
<td>22704</td>
<td>Data exception</td>
<td>MM- Invalid row number</td>
</tr>
<tr>
<td>16005</td>
<td>70705</td>
<td>dharma/ SQL MM errors</td>
<td>MM- Invalid cache block</td>
</tr>
<tr>
<td>16006</td>
<td>70706</td>
<td>dharma/ SQL MM errors</td>
<td>MM- Bad swap file</td>
</tr>
<tr>
<td>16007</td>
<td>70707</td>
<td>dharma/ SQL MM errors</td>
<td>MM- Row too big</td>
</tr>
<tr>
<td>16008</td>
<td>70708</td>
<td>dharma/ SQL MM errors</td>
<td>MM- Array initialized</td>
</tr>
<tr>
<td>16009</td>
<td>70709</td>
<td>dharma/ SQL MM errors</td>
<td>MM- Invalid chunk number</td>
</tr>
<tr>
<td>16010</td>
<td>70710</td>
<td>dharma/ SQL MM errors</td>
<td>MM- Can't create table</td>
</tr>
<tr>
<td>16011</td>
<td>70711</td>
<td>dharma/ SQL MM errors</td>
<td>MM- Can't alter table</td>
</tr>
<tr>
<td>16012</td>
<td>70712</td>
<td>dharma/ SQL MM errors</td>
<td>MM- Can't drop table</td>
</tr>
<tr>
<td>16020</td>
<td>70713</td>
<td>dharma/ SQL MM errors</td>
<td>MM- TPL ctor error</td>
</tr>
<tr>
<td>16021</td>
<td>70714</td>
<td>dharma/ SQL MM errors</td>
<td>MM- Insertion error</td>
</tr>
<tr>
<td>16022</td>
<td>70715</td>
<td>dharma/ SQL MM errors</td>
<td>MM- Deletion error</td>
</tr>
<tr>
<td>16023</td>
<td>70716</td>
<td>dharma/ SQL MM errors</td>
<td>MM- Updation error</td>
</tr>
<tr>
<td>16024</td>
<td>70717</td>
<td>dharma/ SQL MM errors</td>
<td>MM- Fetching error</td>
</tr>
<tr>
<td>16025</td>
<td>70718</td>
<td>dharma/ SQL MM errors</td>
<td>MM- Sorting error</td>
</tr>
<tr>
<td>16026</td>
<td>70719</td>
<td>dharma/ SQL MM errors</td>
<td>MM- Printing error</td>
</tr>
<tr>
<td>16027</td>
<td>70720</td>
<td>dharma/ SQL MM errors</td>
<td>MM- TPLSCAN ctor error</td>
</tr>
<tr>
<td>16028</td>
<td>70721</td>
<td>dharma/ SQL MM errors</td>
<td>MM- Scan fetching error</td>
</tr>
<tr>
<td>16030</td>
<td>70722</td>
<td>dharma/ SQL MM errors</td>
<td>MM- Can't create index</td>
</tr>
<tr>
<td>16031</td>
<td>70723</td>
<td>dharma/ SQL MM errors</td>
<td>MM- Can't drop index</td>
</tr>
<tr>
<td>16032</td>
<td>70724</td>
<td>dharma/ SQL MM errors</td>
<td>MM- IXSCAN ctor error</td>
</tr>
<tr>
<td>16033</td>
<td>70725</td>
<td>dharma/ SQL MM errors</td>
<td>MM- IX ctor error</td>
</tr>
<tr>
<td>16034</td>
<td>70726</td>
<td>dharma/ SQL MM errors</td>
<td>MM- IX deletion error</td>
</tr>
<tr>
<td>16035</td>
<td>70727</td>
<td>dharma/ SQL MM errors</td>
<td>MM- IX appending error</td>
</tr>
<tr>
<td>16036</td>
<td>70728</td>
<td>dharma/ SQL MM errors</td>
<td>MM- IX insertion error</td>
</tr>
<tr>
<td>16037</td>
<td>70729</td>
<td>dharma/ SQL MM errors</td>
<td>MM- IX scan fetching error</td>
</tr>
<tr>
<td>16040</td>
<td>70730</td>
<td>dharma/ SQL MM errors</td>
<td>MM- Begin transaction</td>
</tr>
<tr>
<td>16041</td>
<td>70731</td>
<td>dharma/ SQL MM errors</td>
<td>MM- Commit transaction</td>
</tr>
<tr>
<td>16042</td>
<td>40000</td>
<td>Transaction rollback</td>
<td>***MM- Rollback transaction</td>
</tr>
<tr>
<td>16043</td>
<td>70732</td>
<td>dharma/ SQL MM errors</td>
<td>MM- Mark point</td>
</tr>
<tr>
<td>Error Code</td>
<td>SQLSTATE Value</td>
<td>Class Condition</td>
<td>Subclass Message</td>
</tr>
<tr>
<td>------------</td>
<td>----------------</td>
<td>-----------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>16044</td>
<td>70733</td>
<td>dharma/SQL MM errors</td>
<td>MM - Rollback savepoint</td>
</tr>
<tr>
<td>16045</td>
<td>70734</td>
<td>dharma/SQL MM errors</td>
<td>MM - Set &amp; Get isolation</td>
</tr>
<tr>
<td>16050</td>
<td>70735</td>
<td>dharma/SQL MM errors</td>
<td>MM- TID to char</td>
</tr>
<tr>
<td>16051</td>
<td>70736</td>
<td>dharma/SQL MM errors</td>
<td>MM- char to TID</td>
</tr>
<tr>
<td>20000</td>
<td>50501</td>
<td>dharma/SQL rds error</td>
<td>SQL internal error</td>
</tr>
<tr>
<td>20001</td>
<td>50502</td>
<td>dharma/SQL rds error</td>
<td>Memory allocation failure</td>
</tr>
<tr>
<td>20002</td>
<td>50503</td>
<td>dharma/SQL rds error</td>
<td>Open database failed</td>
</tr>
<tr>
<td>20003</td>
<td>2a504</td>
<td>Syntax error</td>
<td>Syntax error</td>
</tr>
<tr>
<td>20004</td>
<td>28505</td>
<td>Invalid auth specs</td>
<td>User not found</td>
</tr>
<tr>
<td>20005</td>
<td>22506</td>
<td>Data exception</td>
<td>Table/ View/ Synonym not found</td>
</tr>
<tr>
<td>20006</td>
<td>22507</td>
<td>Data exception</td>
<td>Column not found/ specified</td>
</tr>
<tr>
<td>20007</td>
<td>22508</td>
<td>Data exception</td>
<td>No columns in table</td>
</tr>
<tr>
<td>20008</td>
<td>22509</td>
<td>Data exception</td>
<td>Inconsistent types</td>
</tr>
<tr>
<td>20009</td>
<td>22510</td>
<td>Data exception</td>
<td>Column ambiguously specified</td>
</tr>
<tr>
<td>20010</td>
<td>22511</td>
<td>Data exception</td>
<td>Duplicate column specification</td>
</tr>
<tr>
<td>20011</td>
<td>22512</td>
<td>Data exception</td>
<td>Invalid length</td>
</tr>
<tr>
<td>20012</td>
<td>22513</td>
<td>Data exception</td>
<td>Invalid precision</td>
</tr>
<tr>
<td>20013</td>
<td>22514</td>
<td>Data exception</td>
<td>Invalid scale</td>
</tr>
<tr>
<td>20014</td>
<td>22515</td>
<td>Data exception</td>
<td>Missing input parameters</td>
</tr>
<tr>
<td>20015</td>
<td>22516</td>
<td>Data exception</td>
<td>Subquery returns multiple rows</td>
</tr>
<tr>
<td>20016</td>
<td>22517</td>
<td>Data exception</td>
<td>Null value supplied for a mandatory (not null) column</td>
</tr>
<tr>
<td>20017</td>
<td>22518</td>
<td>Data exception</td>
<td>Too many values specified</td>
</tr>
<tr>
<td>20018</td>
<td>22519</td>
<td>Data exception</td>
<td>Too few values specified</td>
</tr>
<tr>
<td>20019</td>
<td>50520</td>
<td>dharma/SQL rds error</td>
<td>Can not modify table referred to in subquery</td>
</tr>
<tr>
<td>20020</td>
<td>42521</td>
<td>Access rule violation</td>
<td>Bad column specification for group by clause</td>
</tr>
<tr>
<td>20021</td>
<td>42522</td>
<td>Access rule violation</td>
<td>Non-group-by expression in having clause</td>
</tr>
<tr>
<td>20022</td>
<td>42523</td>
<td>Access rule violation</td>
<td>Non-group-by expression in select clause</td>
</tr>
<tr>
<td>20023</td>
<td>42524</td>
<td>Access rule violation</td>
<td>Aggregate function not allowed here</td>
</tr>
<tr>
<td>20024</td>
<td>0a000</td>
<td>feature not supported</td>
<td>Sorry, operation not yet implemented</td>
</tr>
<tr>
<td>20025</td>
<td>42526</td>
<td>Access rule violation</td>
<td>Aggregate functions nested</td>
</tr>
<tr>
<td>20026</td>
<td>50527</td>
<td>dharma/SQL rds error</td>
<td>Too many table references</td>
</tr>
<tr>
<td>20027</td>
<td>42528</td>
<td>Access rule violation</td>
<td>Bad field specification in order by clause</td>
</tr>
<tr>
<td>Error Code</td>
<td>SQLSTATE Value</td>
<td>Class Condition</td>
<td>Subclass Message</td>
</tr>
<tr>
<td>------------</td>
<td>----------------</td>
<td>-----------------</td>
<td>------------------</td>
</tr>
<tr>
<td>20028</td>
<td>50529</td>
<td>dharma/SQL rds error</td>
<td>An index with the same name already exists</td>
</tr>
<tr>
<td>20029</td>
<td>50530</td>
<td>dharma/SQL rds error</td>
<td>Index referenced not found</td>
</tr>
<tr>
<td>20030</td>
<td>22531</td>
<td>Data exception</td>
<td>Table space with same name already exists</td>
</tr>
<tr>
<td>20031</td>
<td>50532</td>
<td>dharma/SQL rds error</td>
<td>Cluster with same name already exists</td>
</tr>
<tr>
<td>20032</td>
<td>50533</td>
<td>dharma/SQL rds error</td>
<td>No cluster with this name</td>
</tr>
<tr>
<td>20033</td>
<td>22534</td>
<td>Data exception</td>
<td>Tablespace not found</td>
</tr>
<tr>
<td>20034</td>
<td>50535</td>
<td>dharma/SQL rds error</td>
<td>Bad free percentage specification</td>
</tr>
<tr>
<td>20035</td>
<td>50536</td>
<td>dharma/SQL rds error</td>
<td>At least column spec or null clause should be specified</td>
</tr>
<tr>
<td>20036</td>
<td>07537</td>
<td>dynamic sql error</td>
<td>Statement not prepared</td>
</tr>
<tr>
<td>20037</td>
<td>24538</td>
<td>Invalid cursor state</td>
<td>Executing select statement</td>
</tr>
<tr>
<td>20038</td>
<td>24539</td>
<td>Invalid cursor state</td>
<td>Cursor not closed</td>
</tr>
<tr>
<td>20039</td>
<td>24540</td>
<td>Invalid cursor state</td>
<td>Open for non select statement</td>
</tr>
<tr>
<td>20040</td>
<td>24541</td>
<td>Invalid cursor state</td>
<td>Cursor not opened</td>
</tr>
<tr>
<td>20041</td>
<td>22542</td>
<td>Data exception</td>
<td>Table/View/Synonym already exists</td>
</tr>
<tr>
<td>20042</td>
<td>2a543</td>
<td>Syntax error</td>
<td>Distinct specified more than once in query</td>
</tr>
<tr>
<td>20043</td>
<td>50544</td>
<td>dharma/SQL rds error</td>
<td>Tuple size too high</td>
</tr>
<tr>
<td>20044</td>
<td>50545</td>
<td>dharma/SQL rds error</td>
<td>Array size too high</td>
</tr>
<tr>
<td>20045</td>
<td>08546</td>
<td>Connection exception</td>
<td>File does not exist or not accessible</td>
</tr>
<tr>
<td>20046</td>
<td>50547</td>
<td>dharma/SQL rds error</td>
<td>Field value not null for some tuples</td>
</tr>
<tr>
<td>20047</td>
<td>42548</td>
<td>Access rule violation</td>
<td>Granting to self not allowed</td>
</tr>
<tr>
<td>20048</td>
<td>42549</td>
<td>Access rule violation</td>
<td>Revoking for self not allowed</td>
</tr>
<tr>
<td>20049</td>
<td>22550</td>
<td>Data exception</td>
<td>Keyword used for a name</td>
</tr>
<tr>
<td>20050</td>
<td>21551</td>
<td>Cardinality violation</td>
<td>Too many fields specified</td>
</tr>
<tr>
<td>20051</td>
<td>21552</td>
<td>Cardinality violation</td>
<td>Too many indexes on this table</td>
</tr>
<tr>
<td>20052</td>
<td>22553</td>
<td>Data exception</td>
<td>Overflow/Underflow error</td>
</tr>
<tr>
<td>20053</td>
<td>08554</td>
<td>Connection exception</td>
<td>Database not opened</td>
</tr>
<tr>
<td>20054</td>
<td>08555</td>
<td>Connection exception</td>
<td>Database not specified or improperly specified</td>
</tr>
<tr>
<td>20055</td>
<td>08556</td>
<td>Connection exception</td>
<td>Database not specified or Database not started</td>
</tr>
<tr>
<td>20056</td>
<td>28557</td>
<td>Invalid auth specs</td>
<td>No DBA access rights</td>
</tr>
<tr>
<td>20057</td>
<td>28558</td>
<td>Invalid auth specs</td>
<td>No RESOURCE privileges</td>
</tr>
<tr>
<td>20058</td>
<td>40559</td>
<td>Transaction rollback</td>
<td>Executing SQL statement for an aborted transaction</td>
</tr>
</tbody>
</table>
## Error Codes, SQLSTATE Values, and Messages

<table>
<thead>
<tr>
<th>Error Code</th>
<th>SQLSTATE Value</th>
<th>Class Condition</th>
<th>Subclass Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>20059</td>
<td>22560</td>
<td>Data exception</td>
<td>No files in the table space</td>
</tr>
<tr>
<td>20060</td>
<td>22561</td>
<td>Data exception</td>
<td>Table not empty</td>
</tr>
<tr>
<td>20061</td>
<td>22562</td>
<td>Data exception</td>
<td>Input parameter size too high</td>
</tr>
<tr>
<td>20062</td>
<td>42563</td>
<td>Syntax error</td>
<td>Full pathname not specified</td>
</tr>
<tr>
<td>20063</td>
<td>50564</td>
<td>dharma/SQL rds error</td>
<td>Duplicate file specification</td>
</tr>
<tr>
<td>20064</td>
<td>08565</td>
<td>Connection exception</td>
<td>Invalid attach type</td>
</tr>
<tr>
<td>20065</td>
<td>26000</td>
<td>Invalid SQL statement name</td>
<td>Invalid statement type</td>
</tr>
<tr>
<td>20066</td>
<td>33567</td>
<td>Invalid SQL descriptor name</td>
<td>Invalid sqlda</td>
</tr>
<tr>
<td>20067</td>
<td>08568</td>
<td>Connection exception</td>
<td>More than one database can’t be attached locally</td>
</tr>
<tr>
<td>20068</td>
<td>42569</td>
<td>Syntax error</td>
<td>Bad arguments</td>
</tr>
<tr>
<td>20069</td>
<td>33570</td>
<td>Invalid SQL descriptor name</td>
<td>SQLDA size not enough</td>
</tr>
<tr>
<td>20070</td>
<td>33571</td>
<td>Invalid SQL descriptor name</td>
<td>SQLDA buffer length too high</td>
</tr>
<tr>
<td>20071</td>
<td>42572</td>
<td>Access rule violation</td>
<td>Specified operation not allowed on the view</td>
</tr>
<tr>
<td>20072</td>
<td>50573</td>
<td>dharma/SQL rds error</td>
<td>Server is not allocated</td>
</tr>
<tr>
<td>20073</td>
<td>2a574</td>
<td>Access rule violation</td>
<td>View query specification for view too long</td>
</tr>
<tr>
<td>20074</td>
<td>2a575</td>
<td>Access rule violation</td>
<td>View column list must be specified as expressions are given</td>
</tr>
<tr>
<td>20075</td>
<td>21576</td>
<td>Cardinality violation</td>
<td>Number of columns in column list is less than in select list</td>
</tr>
<tr>
<td>20076</td>
<td>21577</td>
<td>Cardinality violation</td>
<td>Number of columns in column list is more than in select list</td>
</tr>
<tr>
<td>20077</td>
<td>42578</td>
<td>Access rule violation</td>
<td>Check option specified for non-insertable view</td>
</tr>
<tr>
<td>20078</td>
<td>42579</td>
<td>Access rule violation</td>
<td>Given SQL statement is not allowed on the view</td>
</tr>
<tr>
<td>20079</td>
<td>50580</td>
<td>dharma/SQL rds error</td>
<td>More Tables cannot be created.</td>
</tr>
<tr>
<td>20080</td>
<td>44581</td>
<td>Check option violation</td>
<td>View check option violation</td>
</tr>
<tr>
<td>20081</td>
<td>22582</td>
<td>Data exception</td>
<td>No of expressions projected on either side of set-op don’t match</td>
</tr>
<tr>
<td>20082</td>
<td>42583</td>
<td>Access rule violation</td>
<td>Column names not allowed in order by clause for this statement</td>
</tr>
<tr>
<td>20083</td>
<td>42584</td>
<td>Access rule violation</td>
<td>Outerjoin specified on a complex predicate</td>
</tr>
<tr>
<td>20084</td>
<td>42585</td>
<td>Access rule violation</td>
<td>Outerjoin specified on a sub_query</td>
</tr>
<tr>
<td>20085</td>
<td>42586</td>
<td>Access rule violation</td>
<td>Invalid Outerjoin specification</td>
</tr>
<tr>
<td>20086</td>
<td>42587</td>
<td>Access rule violation</td>
<td>Duplicate table constraint specification</td>
</tr>
<tr>
<td>20087</td>
<td>21588</td>
<td>Cardinality violation</td>
<td>Column count mismatch</td>
</tr>
<tr>
<td>Error Code</td>
<td>SQLSTATE Value</td>
<td>Class Condition</td>
<td>Subclass Message</td>
</tr>
<tr>
<td>------------</td>
<td>----------------</td>
<td>-----------------</td>
<td>------------------</td>
</tr>
<tr>
<td>20088</td>
<td>28589</td>
<td>Invalid auth specs</td>
<td>Invalid user name</td>
</tr>
<tr>
<td>20089</td>
<td>22590</td>
<td>Data exception</td>
<td>System date retrieval failed</td>
</tr>
<tr>
<td>20090</td>
<td>42591</td>
<td>Access rule violation</td>
<td>Table columnlist must be specified as expressions are given</td>
</tr>
<tr>
<td>20091</td>
<td>2a592</td>
<td>Access rule violation</td>
<td>Query statement too long.</td>
</tr>
<tr>
<td>20092</td>
<td>2d593</td>
<td>Invalid transaction termination</td>
<td>No tuples selected by the subquery for update</td>
</tr>
<tr>
<td>20093</td>
<td>22594</td>
<td>Data exception</td>
<td>Synonym already exists</td>
</tr>
<tr>
<td>20094</td>
<td>22595</td>
<td>Remote database access</td>
<td>Database link with same name already exists</td>
</tr>
<tr>
<td>20095</td>
<td>22596</td>
<td>Remote database access</td>
<td>Database link not found</td>
</tr>
<tr>
<td>20096</td>
<td>08597</td>
<td>Connection exception</td>
<td>Connect String not specified/ incorrect</td>
</tr>
<tr>
<td>20097</td>
<td>22598</td>
<td>Remote database access</td>
<td>Specified operation not allowed on a remote table</td>
</tr>
<tr>
<td>20098</td>
<td>22599</td>
<td>Data exception</td>
<td>More than one row selected by the query</td>
</tr>
<tr>
<td>20099</td>
<td>24000</td>
<td>Invalid cursor state</td>
<td>Cursor not positioned on a valid row</td>
</tr>
<tr>
<td>20100</td>
<td>4250a</td>
<td>Access rule violation</td>
<td>Subquery not allowed here</td>
</tr>
<tr>
<td>20101</td>
<td>2350b</td>
<td>Integrity constraint</td>
<td>No references for the table</td>
</tr>
<tr>
<td>20102</td>
<td>2350c</td>
<td>Integrity constraint</td>
<td>Primary/ Candidate key column defined null</td>
</tr>
<tr>
<td>20103</td>
<td>2350d</td>
<td>Integrity constraint</td>
<td>No matching key defined for the referenced table</td>
</tr>
<tr>
<td>20104</td>
<td>2350e</td>
<td>Integrity constraint</td>
<td>Keys in reference constraint incompatible</td>
</tr>
<tr>
<td>20105</td>
<td>5050f</td>
<td>dharma/ SQL rds error</td>
<td>Statement not allowed in readonly isolation level</td>
</tr>
<tr>
<td>20106</td>
<td>2150g</td>
<td>Cardinality violation</td>
<td>Invalid ROWID</td>
</tr>
<tr>
<td>20107</td>
<td>2250h</td>
<td>Remote database access</td>
<td>Remote Database not started</td>
</tr>
<tr>
<td>20108</td>
<td>0850i</td>
<td>Connection exception</td>
<td>Remote Network Server not started.</td>
</tr>
<tr>
<td>20109</td>
<td>2250j</td>
<td>Remote database access</td>
<td>Remote Database Name not valid</td>
</tr>
<tr>
<td>20110</td>
<td>0850k</td>
<td>Connection exception</td>
<td>TCP/ IP Remote HostName is unknown.</td>
</tr>
<tr>
<td>20114</td>
<td>33002</td>
<td>Invalid SQL descriptor name</td>
<td>Fetched Value NULL &amp; indicator var not defined</td>
</tr>
<tr>
<td>20115</td>
<td>5050l</td>
<td>dharma/ SQL rds error</td>
<td>References to the table/ record present</td>
</tr>
<tr>
<td>20116</td>
<td>2350m</td>
<td>Integrity constraint</td>
<td>Constraint violation</td>
</tr>
<tr>
<td>20117</td>
<td>2350n</td>
<td>Integrity constraint</td>
<td>Table definition not complete</td>
</tr>
<tr>
<td>20118</td>
<td>4250o</td>
<td>Access rule violation</td>
<td>Duplicate constraint name</td>
</tr>
<tr>
<td>20119</td>
<td>2350p</td>
<td>Integrity constraint</td>
<td>Constraint name not found</td>
</tr>
<tr>
<td>20120</td>
<td>22000</td>
<td>Data exception</td>
<td>**use of reserved word</td>
</tr>
<tr>
<td>20121</td>
<td>5050q</td>
<td>dharma/ SQL rds error</td>
<td>permission denied</td>
</tr>
<tr>
<td>Error Code</td>
<td>SQLSTATE Value</td>
<td>Class Condition</td>
<td>Subclass Message</td>
</tr>
<tr>
<td>------------</td>
<td>----------------</td>
<td>-----------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>20122</td>
<td>5050r</td>
<td>dharma/SQL rds error</td>
<td>procedure not found</td>
</tr>
<tr>
<td>20123</td>
<td>5050s</td>
<td>dharma/SQL rds error</td>
<td>invalid arguments to procedure</td>
</tr>
<tr>
<td>20124</td>
<td>5050t</td>
<td>dharma/SQL rds error</td>
<td>Query conditionally terminated</td>
</tr>
<tr>
<td>20125</td>
<td>0750u</td>
<td>Dynamic sql-error</td>
<td>Number of open cursors exceeds limit</td>
</tr>
<tr>
<td>20126</td>
<td>34000</td>
<td>Invalid cursor name</td>
<td>Invalid cursor name</td>
</tr>
<tr>
<td>20127</td>
<td>07001</td>
<td>Dynamic sql-error</td>
<td>Bad parameter specification for the statement</td>
</tr>
<tr>
<td>20128</td>
<td>2250x</td>
<td>Data Exception</td>
<td>Numeric value out of range.</td>
</tr>
<tr>
<td>20129</td>
<td>2250y</td>
<td>Data Exception</td>
<td>Data truncated.</td>
</tr>
<tr>
<td>20130</td>
<td>40000</td>
<td>Dharma/SQL rds error</td>
<td>Rolled back transaction.</td>
</tr>
<tr>
<td>20131</td>
<td>50522</td>
<td>dharma/SQL rds error</td>
<td>Value for Parameter marker is Missing</td>
</tr>
<tr>
<td>20132</td>
<td>5050u</td>
<td>dharma/SQL rds error</td>
<td>Revoke failed because of restrict</td>
</tr>
<tr>
<td>20133</td>
<td>0a001</td>
<td>Feature not supported</td>
<td>Feature not supported</td>
</tr>
<tr>
<td>20133</td>
<td>0a000</td>
<td>feature not supported</td>
<td>Sorry, Feature not supported in this Edition.</td>
</tr>
<tr>
<td>20134</td>
<td>5050v</td>
<td>dharma/SQL rds error</td>
<td>Invalid long datatype column references</td>
</tr>
<tr>
<td>20135</td>
<td>5050x</td>
<td>dharma/SQL rds error</td>
<td>Contains operator is not supported in this context</td>
</tr>
<tr>
<td>20135</td>
<td>m0m01</td>
<td>dharma/SQL diagnostics error</td>
<td>diagnostics statement failed.</td>
</tr>
<tr>
<td>20136</td>
<td>5050z</td>
<td>dharma/SQL rds error</td>
<td>Contains operator is not supported for this datatype</td>
</tr>
<tr>
<td>20137</td>
<td>50514</td>
<td>dharma/SQL rds error</td>
<td>Index is not defined or does not support CONTAINS</td>
</tr>
<tr>
<td>20138</td>
<td>50513</td>
<td>dharma/SQL rds error</td>
<td>Index on long fields requires that it can push down only CONTAINS</td>
</tr>
<tr>
<td>20140</td>
<td>50512</td>
<td>dharma/SQL rds error</td>
<td>Procedure already exists</td>
</tr>
<tr>
<td>20141</td>
<td>85001</td>
<td>dharma/SQL Stored procedure Compilation</td>
<td>Error in Stored Procedure Compilation.</td>
</tr>
<tr>
<td>20142</td>
<td>86001</td>
<td>dharma/SQL Stored procedure Execution</td>
<td>Error in Stored Procedure Execution.</td>
</tr>
<tr>
<td>20143</td>
<td>86002</td>
<td>dharma/SQL Stored procedure Execution</td>
<td>Too many recursions in call procedure.</td>
</tr>
<tr>
<td>20144</td>
<td>86003</td>
<td>dharma/SQL Stored procedure Execution</td>
<td>Null value fetched.</td>
</tr>
<tr>
<td>20145</td>
<td>86004</td>
<td>dharma/SQL Stored procedure Execution</td>
<td>Invalid field reference.</td>
</tr>
<tr>
<td>20146</td>
<td>86005</td>
<td>dharma/SQL Triggers</td>
<td>Trigger with this name already exists.</td>
</tr>
<tr>
<td>20147</td>
<td>86006</td>
<td>dharma/SQL Triggers</td>
<td>Trigger with this name does not exist.</td>
</tr>
<tr>
<td>Error Code</td>
<td>SQLSTATE Value</td>
<td>Class Condition</td>
<td>Subclass Message</td>
</tr>
<tr>
<td>------------</td>
<td>----------------</td>
<td>-----------------</td>
<td>------------------</td>
</tr>
<tr>
<td>20148</td>
<td>86007</td>
<td>dharma/SQL Triggers</td>
<td>Trigger Execution Failed.</td>
</tr>
<tr>
<td>20150</td>
<td>50512z</td>
<td>dharma/SQL rds error</td>
<td>view manager ID is not found.</td>
</tr>
<tr>
<td>20151</td>
<td>50515</td>
<td>dharma/SQL rds error</td>
<td>Cannot drop all columns; use DROP TABLE instead</td>
</tr>
<tr>
<td>20152</td>
<td>50516</td>
<td>dharma/SQL rds error</td>
<td>Cannot preallocate memory for n cache items</td>
</tr>
<tr>
<td>20153</td>
<td>50517</td>
<td>dharma/SQL rds error</td>
<td>Tree not present in cache; search failed</td>
</tr>
<tr>
<td>20154</td>
<td>50518</td>
<td>dharma/SQL rds error</td>
<td>Statement cache insert failed</td>
</tr>
<tr>
<td>20155</td>
<td>50519</td>
<td>dharma/SQL rds error</td>
<td>Environment variable used for the creation of multi component index is not set properly.</td>
</tr>
<tr>
<td>20156</td>
<td>50521</td>
<td>dharma/SQL rds error</td>
<td>No SQL statement</td>
</tr>
<tr>
<td>20157</td>
<td>50523</td>
<td>dharma/SQL rds error</td>
<td>Invalid JSP/ T method sequence.</td>
</tr>
<tr>
<td>20211</td>
<td>22800</td>
<td>Data exception</td>
<td>Remote procedure call error</td>
</tr>
<tr>
<td>20212</td>
<td>08801</td>
<td>Connection exception</td>
<td>SQL client bind to daemon failed</td>
</tr>
<tr>
<td>20213</td>
<td>08802</td>
<td>Connection exception</td>
<td>SQL client bind to SQL server failed</td>
</tr>
<tr>
<td>20214</td>
<td>08803</td>
<td>Connection exception</td>
<td>SQL NETWORK service entry is not available</td>
</tr>
<tr>
<td>20215</td>
<td>08804</td>
<td>Connection exception</td>
<td>Invalid TCP/IP hostname</td>
</tr>
<tr>
<td>20216</td>
<td>hz805</td>
<td>Remote database access</td>
<td>Invalid remote database name</td>
</tr>
<tr>
<td>20217</td>
<td>08806</td>
<td>Connection exception</td>
<td>network error on server</td>
</tr>
<tr>
<td>20218</td>
<td>08807</td>
<td>Connection exception</td>
<td>Invalid protocol</td>
</tr>
<tr>
<td>20219</td>
<td>2e000</td>
<td>Invalid connection name</td>
<td>***Invalid connection name</td>
</tr>
<tr>
<td>20220</td>
<td>08809</td>
<td>Connection exception</td>
<td>Duplicate connection name</td>
</tr>
<tr>
<td>20221</td>
<td>08810</td>
<td>Connection exception</td>
<td>No active connection</td>
</tr>
<tr>
<td>20222</td>
<td>08811</td>
<td>Connection exception</td>
<td>No environment defined database</td>
</tr>
<tr>
<td>20223</td>
<td>08812</td>
<td>Connection exception</td>
<td>Multiple local connections</td>
</tr>
<tr>
<td>20224</td>
<td>08813</td>
<td>Connection exception</td>
<td>Invalid protocol in connect_string</td>
</tr>
<tr>
<td>20225</td>
<td>08814</td>
<td>Connection exception</td>
<td>Exceeding permissible number of connections</td>
</tr>
<tr>
<td>20226</td>
<td>80815</td>
<td>dharma/SQL snw errors</td>
<td>Bad database handle</td>
</tr>
<tr>
<td>20227</td>
<td>08816</td>
<td>Connection exception</td>
<td>Invalid host name in connect string</td>
</tr>
<tr>
<td>20228</td>
<td>28817</td>
<td>Invalid auth specs</td>
<td>Access denied (Authorization failed)</td>
</tr>
<tr>
<td>20229</td>
<td>22818</td>
<td>Data exception</td>
<td>Invalid date value</td>
</tr>
<tr>
<td>20230</td>
<td>22819</td>
<td>Data exception</td>
<td>Invalid date string</td>
</tr>
<tr>
<td>20231</td>
<td>22820</td>
<td>Data exception</td>
<td>Invalid number strings</td>
</tr>
<tr>
<td>20232</td>
<td>22821</td>
<td>Data exception</td>
<td>Invalid number string</td>
</tr>
<tr>
<td>Error Code</td>
<td>SQLSTATE Value</td>
<td>Class Condition</td>
<td>Subclass Message</td>
</tr>
<tr>
<td>------------</td>
<td>----------------</td>
<td>-----------------</td>
<td>------------------</td>
</tr>
<tr>
<td>20233</td>
<td>22822</td>
<td>Data exception</td>
<td>Invalid time value</td>
</tr>
<tr>
<td>20234</td>
<td>22523</td>
<td>Data exception</td>
<td>Invalid time string</td>
</tr>
<tr>
<td>20235</td>
<td>22007</td>
<td>Data exception</td>
<td>Invalid time stamp string</td>
</tr>
<tr>
<td>20236</td>
<td>22012</td>
<td>Data exception</td>
<td>Division by zero attempted</td>
</tr>
<tr>
<td>20238</td>
<td>22615</td>
<td>Data exception</td>
<td>Error in format type</td>
</tr>
<tr>
<td>20239</td>
<td>2c000</td>
<td>Invalid character set name</td>
<td>Invalid character set name specified.</td>
</tr>
<tr>
<td>20240</td>
<td>5050y</td>
<td>dharma/SQL rds errors</td>
<td>Invalid collation name specified.</td>
</tr>
<tr>
<td>20241</td>
<td>08815</td>
<td>Connection Exception</td>
<td>Service in use.</td>
</tr>
<tr>
<td>20242</td>
<td>22008</td>
<td>Data exception</td>
<td>Invalid timestamp</td>
</tr>
<tr>
<td>20300</td>
<td>90901</td>
<td>dharma/DBS errors</td>
<td>Column group column doesn't exist</td>
</tr>
<tr>
<td>20301</td>
<td>90902</td>
<td>dharma/DBS errors</td>
<td>Column group column already specified</td>
</tr>
<tr>
<td>20302</td>
<td>90903</td>
<td>dharma/DBS errors</td>
<td>Column group name already specified</td>
</tr>
<tr>
<td>20303</td>
<td>90904</td>
<td>dharma/DBS errors</td>
<td>Column groups haven't covered all columns</td>
</tr>
<tr>
<td>20304</td>
<td>90905</td>
<td>dharma/DBS errors</td>
<td>Column groups are not implemented in Dharma storage</td>
</tr>
<tr>
<td>23000</td>
<td>22563</td>
<td>dharma/SQL Data exception</td>
<td>Table create returned invalid table id</td>
</tr>
<tr>
<td>23001</td>
<td>22564</td>
<td>dharma/SQL Data exception</td>
<td>Index create returned invalid index id</td>
</tr>
<tr>
<td>25001</td>
<td>i0i01</td>
<td>dharma/SQL odbc integrator</td>
<td>Operation is valid only on a linked table</td>
</tr>
<tr>
<td>25002</td>
<td>i0i02</td>
<td>dharma/SQL odbc integrator</td>
<td>Operation not allowed on a linked table</td>
</tr>
<tr>
<td>25003</td>
<td>i0i03</td>
<td>dharma/SQL odbc integrator</td>
<td>Copy object exists</td>
</tr>
<tr>
<td>25004</td>
<td>i0i04</td>
<td>dharma/SQL odbc integrator</td>
<td>Unknown copy object</td>
</tr>
<tr>
<td>25005</td>
<td>i0i05</td>
<td>dharma/SQL odbc integrator</td>
<td>Dropping table failed</td>
</tr>
<tr>
<td>25006</td>
<td>i0i06</td>
<td>dharma/SQL odbc integrator</td>
<td>Bad copy sql statement</td>
</tr>
<tr>
<td>25007</td>
<td>i0i07</td>
<td>dharma/SQL odbc integrator</td>
<td>Unknown data type</td>
</tr>
<tr>
<td>25008</td>
<td>i0i08</td>
<td>dharma/SQL odbc integrator</td>
<td>Bad insert statement</td>
</tr>
<tr>
<td>25009</td>
<td>i0i09</td>
<td>dharma/SQL odbc integrator</td>
<td>Fetch operation failed</td>
</tr>
<tr>
<td>25010</td>
<td>i0i10</td>
<td>dharma/SQL odbc integrator</td>
<td>Insert operation failed</td>
</tr>
<tr>
<td>25011</td>
<td>i0i11</td>
<td>dharma/SQL odbc integrator</td>
<td>Operation not started</td>
</tr>
<tr>
<td>25012</td>
<td>i0i12</td>
<td>dharma/SQL odbc integrator</td>
<td>Operation marked for abort</td>
</tr>
<tr>
<td>25013</td>
<td>i0i13</td>
<td>dharma/SQL odbc integrator</td>
<td>Commit operation failed</td>
</tr>
<tr>
<td>25014</td>
<td>i0i14</td>
<td>dharma/SQL odbc integrator</td>
<td>Create table failed</td>
</tr>
<tr>
<td>25015</td>
<td>i0i15</td>
<td>dharma/SQL odbc integrator</td>
<td>Bad sync sql statement</td>
</tr>
<tr>
<td>25016</td>
<td>i0i16</td>
<td>dharma/SQL odbc integrator</td>
<td>Sync object exists</td>
</tr>
<tr>
<td>Error Code</td>
<td>SQLSTATE Value</td>
<td>Class Condition</td>
<td>Subclass Message</td>
</tr>
<tr>
<td>------------</td>
<td>----------------</td>
<td>-----------------</td>
<td>------------------------------------------</td>
</tr>
<tr>
<td>25017</td>
<td>i0i17</td>
<td>dharma/ SQL odbc integrator</td>
<td>Create sync object failed</td>
</tr>
<tr>
<td>25018</td>
<td>i0i18</td>
<td>dharma/ SQL odbc integrator</td>
<td>Create copy object failed</td>
</tr>
<tr>
<td>25019</td>
<td>i0i19</td>
<td>dharma/ SQL odbc integrator</td>
<td>Unknown sync object</td>
</tr>
<tr>
<td>25020</td>
<td>i0i20</td>
<td>dharma/ SQL odbc integrator</td>
<td>Illegal column name</td>
</tr>
<tr>
<td>25021</td>
<td>i0i21</td>
<td>dharma/ SQL odbc integrator</td>
<td>Duplicate column name</td>
</tr>
<tr>
<td>25022</td>
<td>i0i22</td>
<td>dharma/ SQL odbc integrator</td>
<td>Install failure</td>
</tr>
<tr>
<td>25023</td>
<td>i0i23</td>
<td>dharma/ SQL odbc integrator</td>
<td>Invalid sync mode</td>
</tr>
<tr>
<td>25024</td>
<td>i0i24</td>
<td>dharma/ SQL odbc integrator</td>
<td>Download or snapshot table missing</td>
</tr>
<tr>
<td>25025</td>
<td>i0i25</td>
<td>dharma/ SQL odbc integrator</td>
<td>Upload table missing</td>
</tr>
<tr>
<td>25026</td>
<td>i0i26</td>
<td>dharma/ SQL odbc integrator</td>
<td>Update operation failed</td>
</tr>
<tr>
<td>25027</td>
<td>i0i27</td>
<td>dharma/ SQL odbc integrator</td>
<td>Delete operation failed</td>
</tr>
<tr>
<td>25028</td>
<td>i0i28</td>
<td>dharma/ SQL odbc integrator</td>
<td>Close cursor failed</td>
</tr>
<tr>
<td>25029</td>
<td>i0i29</td>
<td>dharma/ SQL odbc integrator</td>
<td>No primary key</td>
</tr>
<tr>
<td>25030</td>
<td>i0i30</td>
<td>dharma/ SQL odbc integrator</td>
<td>Missing row</td>
</tr>
<tr>
<td>25031</td>
<td>i0i31</td>
<td>dharma/ SQL odbc integrator</td>
<td>Bad primary key</td>
</tr>
<tr>
<td>25032</td>
<td>i0i32</td>
<td>dharma/ SQL odbc integrator</td>
<td>Update contention</td>
</tr>
<tr>
<td>25033</td>
<td>i0i33</td>
<td>dharma/ SQL odbc integrator</td>
<td>Link table failed</td>
</tr>
<tr>
<td>25034</td>
<td>i0i34</td>
<td>dharma/ SQL odbc integrator</td>
<td>Unlink table failed</td>
</tr>
<tr>
<td>25035</td>
<td>i0i35</td>
<td>dharma/ SQL odbc integrator</td>
<td>Link data source failed</td>
</tr>
<tr>
<td>25036</td>
<td>i0i36</td>
<td>dharma/ SQL odbc integrator</td>
<td>Unlink data source failed</td>
</tr>
<tr>
<td>25037</td>
<td>i0i37</td>
<td>dharma/ SQL odbc integrator</td>
<td>Integrator internal error</td>
</tr>
<tr>
<td>25038</td>
<td>i0i38</td>
<td>dharma/ SQL odbc integrator</td>
<td>Operation already started</td>
</tr>
<tr>
<td>25039</td>
<td>i0i39</td>
<td>dharma/ SQL odbc integrator</td>
<td>Opening of copy sql stmt failed</td>
</tr>
<tr>
<td>25040</td>
<td>i0i40</td>
<td>dharma/ SQL odbc integrator</td>
<td>sync object failed</td>
</tr>
<tr>
<td>25041</td>
<td>i0i41</td>
<td>dharma/ SQL odbc integrator</td>
<td>Dropping copy object failed</td>
</tr>
<tr>
<td>25042</td>
<td>i0i42</td>
<td>dharma/ SQL odbc integrator</td>
<td>Closing copy sql stmt failed</td>
</tr>
<tr>
<td>25043</td>
<td>i0i43</td>
<td>dharma/ SQL odbc integrator</td>
<td>Failure to update metadata timestamp</td>
</tr>
<tr>
<td>25101</td>
<td>j0j01</td>
<td>dharma/ SQL odbc trans layer</td>
<td>SQLAllocEnv failed</td>
</tr>
<tr>
<td>25102</td>
<td>j0j02</td>
<td>dharma/ SQL odbc trans layer</td>
<td>SQLAllocConnect failed</td>
</tr>
<tr>
<td>25103</td>
<td>j0j03</td>
<td>dharma/ SQL odbc trans layer</td>
<td>SQLConnect failed</td>
</tr>
<tr>
<td>25104</td>
<td>j0j04</td>
<td>dharma/ SQL odbc trans layer</td>
<td>SQLGetConnectOption failed</td>
</tr>
<tr>
<td>25105</td>
<td>j0j05</td>
<td>dharma/ SQL odbc trans layer</td>
<td>SQLSetConnectOption failed</td>
</tr>
</tbody>
</table>
## Error Codes, SQLSTATE Values, and Messages

<table>
<thead>
<tr>
<th>Error Code</th>
<th>SQLSTATE Value</th>
<th>Class Condition</th>
<th>Subclass Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>25106</td>
<td>j0j06</td>
<td>dharma/SQL odbc trans layer</td>
<td>Failed to map stmt handle to UUID</td>
</tr>
<tr>
<td>25107</td>
<td>j0j07</td>
<td>dharma/SQL odbc trans layer</td>
<td>SQLSetParam failed</td>
</tr>
<tr>
<td>25108</td>
<td>j0j08</td>
<td>dharma/SQL odbc trans layer</td>
<td>SQLDisconnect failed</td>
</tr>
<tr>
<td>25109</td>
<td>j0j09</td>
<td>dharma/SQL odbc trans layer</td>
<td>SQLExecute failed</td>
</tr>
<tr>
<td>25110</td>
<td>j0j10</td>
<td>dharma/SQL odbc trans layer</td>
<td>SQLRowCount failed</td>
</tr>
<tr>
<td>25111</td>
<td>j0j11</td>
<td>dharma/SQL odbc trans layer</td>
<td>SQLSetParam failed</td>
</tr>
<tr>
<td>25112</td>
<td>j0j12</td>
<td>dharma/SQL odbc trans layer</td>
<td>SQLBindCol failed</td>
</tr>
<tr>
<td>25113</td>
<td>j0j13</td>
<td>dharma/SQL odbc trans layer</td>
<td>SQLPrepare failed</td>
</tr>
<tr>
<td>25114</td>
<td>j0j14</td>
<td>dharma/SQL odbc trans layer</td>
<td>SQLResultCols failed</td>
</tr>
<tr>
<td>25115</td>
<td>j0j15</td>
<td>dharma/SQL odbc trans layer</td>
<td>SQLDescribeCol failed</td>
</tr>
<tr>
<td>25116</td>
<td>j0j16</td>
<td>dharma/SQL odbc trans layer</td>
<td>SQLFreeStmt failed</td>
</tr>
<tr>
<td>25117</td>
<td>j0j17</td>
<td>dharma/SQL odbc trans layer</td>
<td>SQLFetch failed</td>
</tr>
<tr>
<td>25118</td>
<td>j0j18</td>
<td>dharma/SQL odbc trans layer</td>
<td>SQLTransact failed</td>
</tr>
<tr>
<td>25119</td>
<td>j0j19</td>
<td>dharma/SQL odbc trans layer</td>
<td>SQLAllocStmt failed</td>
</tr>
<tr>
<td>25120</td>
<td>j0j20</td>
<td>dharma/SQL odbc trans layer</td>
<td>SQLTables failed</td>
</tr>
<tr>
<td>25121</td>
<td>j0j21</td>
<td>dharma/SQL odbc trans layer</td>
<td>SQLColumns failed</td>
</tr>
<tr>
<td>25122</td>
<td>j0j22</td>
<td>dharma/SQL odbc trans layer</td>
<td>SQLStatistics failed</td>
</tr>
<tr>
<td>25123</td>
<td>j0j23</td>
<td>dharma/SQL odbc trans layer</td>
<td>ODBC Driver interface mismatch</td>
</tr>
<tr>
<td>25124</td>
<td>j0j24</td>
<td>dharma/SQL odbc trans layer</td>
<td>ODBC Driver metadata exceeds storage limits</td>
</tr>
<tr>
<td>25125</td>
<td>j0j25</td>
<td>dharma/SQL odbc trans layer</td>
<td>SQLGetInfo failed</td>
</tr>
<tr>
<td>25126</td>
<td>j0j26</td>
<td>dharma/SQL odbc trans layer</td>
<td>operation not allowed on the read-only database</td>
</tr>
<tr>
<td>25127</td>
<td>j0j27</td>
<td>dharma/SQL odbc trans layer</td>
<td>cannot update views-with-check-option on remote tables</td>
</tr>
<tr>
<td>25128</td>
<td>j0j28</td>
<td>dharma/SQL odbc trans layer</td>
<td>query terminated as max row limit exceeded for a remote table</td>
</tr>
<tr>
<td>25129</td>
<td>j0j29</td>
<td>dharma/SQL odbc trans layer</td>
<td>unable to read column info from remote table</td>
</tr>
<tr>
<td>26001</td>
<td>08001</td>
<td>dharma/SQL JDBC errors</td>
<td>Unable to connect to data source</td>
</tr>
<tr>
<td>26002</td>
<td>08003</td>
<td>dharma/SQL JDBC errors</td>
<td>Connection does not exist</td>
</tr>
<tr>
<td>26003</td>
<td>08501</td>
<td>dharma/SQL JDBC errors</td>
<td>Communication link failure</td>
</tr>
<tr>
<td>26004</td>
<td>08597</td>
<td>dharma/SQL JDBC errors</td>
<td>Connect String not specified/ incorrect</td>
</tr>
<tr>
<td>26005</td>
<td>0850i</td>
<td>dharma/SQL JDBC errors</td>
<td>Remote Network Server not started</td>
</tr>
<tr>
<td>26006</td>
<td>0850k</td>
<td>dharma/SQL JDBC errors</td>
<td>TCP/IP Remote HostName is unknown</td>
</tr>
<tr>
<td>26007</td>
<td>08801</td>
<td>dharma/SQL JDBC errors</td>
<td>SQL client bind to daemon failed</td>
</tr>
</tbody>
</table>

C-12 Error Messages
<table>
<thead>
<tr>
<th>Error Code</th>
<th>SQLSTATE Value</th>
<th>Class Condition</th>
<th>Subclass Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>26008</td>
<td>08802</td>
<td>dharma/ SQL JDBC errors</td>
<td>SQL client bind to SQL server failed</td>
</tr>
<tr>
<td>26009</td>
<td>08803</td>
<td>dharma/ SQL JDBC errors</td>
<td>SQL Network service entry is not available</td>
</tr>
<tr>
<td>26010</td>
<td>08804</td>
<td>dharma/ SQL JDBC errors</td>
<td>Invalid TCP/IP hostname</td>
</tr>
<tr>
<td>26011</td>
<td>08806</td>
<td>dharma/ SQL JDBC errors</td>
<td>Network error on server</td>
</tr>
<tr>
<td>26012</td>
<td>08807</td>
<td>dharma/ SQL JDBC errors</td>
<td>Invalid protocol</td>
</tr>
<tr>
<td>26013</td>
<td>08816</td>
<td>dharma/ SQL JDBC errors</td>
<td>Invalid host name in connect string</td>
</tr>
<tr>
<td>26014</td>
<td>08809</td>
<td>dharma/ SQL JDBC errors</td>
<td>Duplicate connection name</td>
</tr>
<tr>
<td>26015</td>
<td>08810</td>
<td>dharma/ SQL JDBC errors</td>
<td>No active connection</td>
</tr>
<tr>
<td>26016</td>
<td>08812</td>
<td>dharma/ SQL JDBC errors</td>
<td>Multiple local connections</td>
</tr>
<tr>
<td>26017</td>
<td>08813</td>
<td>dharma/ SQL JDBC errors</td>
<td>Invalid protocol in connect_string</td>
</tr>
<tr>
<td>26018</td>
<td>08814</td>
<td>dharma/ SQL JDBC errors</td>
<td>Exceeding permissible number of connections</td>
</tr>
<tr>
<td>26019</td>
<td>22818</td>
<td>dharma/ SQL JDBC errors</td>
<td>Invalid date value</td>
</tr>
<tr>
<td>26020</td>
<td>22819</td>
<td>dharma/ SQL JDBC errors</td>
<td>Invalid date string</td>
</tr>
<tr>
<td>26021</td>
<td>22820</td>
<td>dharma/ SQL JDBC errors</td>
<td>Invalid number strings</td>
</tr>
<tr>
<td>26022</td>
<td>22821</td>
<td>dharma/ SQL JDBC errors</td>
<td>Invalid number string</td>
</tr>
<tr>
<td>26023</td>
<td>22822</td>
<td>dharma/ SQL JDBC errors</td>
<td>Invalid time value</td>
</tr>
<tr>
<td>26024</td>
<td>22523</td>
<td>dharma/ SQL JDBC errors</td>
<td>Invalid time string</td>
</tr>
<tr>
<td>26025</td>
<td>22007</td>
<td>dharma/ SQL JDBC errors</td>
<td>Invalid time stamp string</td>
</tr>
<tr>
<td>26026</td>
<td>50501</td>
<td>dharma/ SQL JDBC errors</td>
<td>SQL internal error</td>
</tr>
<tr>
<td>26027</td>
<td>50502</td>
<td>dharma/ SQL JDBC errors</td>
<td>Memory allocation failure</td>
</tr>
<tr>
<td>26028</td>
<td>22509</td>
<td>dharma/ SQL JDBC errors</td>
<td>Inconsistent types</td>
</tr>
<tr>
<td>26029</td>
<td>22515</td>
<td>dharma/ SQL JDBC errors</td>
<td>Missing input parameters</td>
</tr>
<tr>
<td>26030</td>
<td>0e000</td>
<td>dharma/ SQL JDBC errors</td>
<td>Sorry, operation not yet implemented</td>
</tr>
<tr>
<td>26031</td>
<td>24538</td>
<td>dharma/ SQL JDBC errors</td>
<td>Executing select statement</td>
</tr>
<tr>
<td>26032</td>
<td>24539</td>
<td>dharma/ SQL JDBC errors</td>
<td>Cursor not closed</td>
</tr>
<tr>
<td>26033</td>
<td>24540</td>
<td>dharma/ SQL JDBC errors</td>
<td>Open for non select statement</td>
</tr>
<tr>
<td>26034</td>
<td>22553</td>
<td>dharma/ SQL JDBC errors</td>
<td>Overflow error</td>
</tr>
<tr>
<td>26035</td>
<td>08555</td>
<td>dharma/ SQL JDBC errors</td>
<td>Database not specified or improperly specified</td>
</tr>
<tr>
<td>26036</td>
<td>08556</td>
<td>dharma/ SQL JDBC errors</td>
<td>Database not specified or Database not started</td>
</tr>
<tr>
<td>26037</td>
<td>22562</td>
<td>dharma/ SQL JDBC errors</td>
<td>Input parameter size too high</td>
</tr>
<tr>
<td>26038</td>
<td>26000</td>
<td>dharma/ SQL JDBC errors</td>
<td>Invalid statement type</td>
</tr>
<tr>
<td>26039</td>
<td>07001</td>
<td>dharma/ SQL JDBC errors</td>
<td>Wrong number of parameters</td>
</tr>
<tr>
<td>Error Code</td>
<td>SQLSTATE Value</td>
<td>Class Condition</td>
<td>Subclass Message</td>
</tr>
<tr>
<td>------------</td>
<td>----------------</td>
<td>-----------------</td>
<td>------------------</td>
</tr>
<tr>
<td>26040</td>
<td>2250x</td>
<td>dharma/SQL JDBC errors</td>
<td>Numeric value out of range.</td>
</tr>
<tr>
<td>26041</td>
<td>2250y</td>
<td>dharma/SQL JDBC errors</td>
<td>Data truncated.</td>
</tr>
<tr>
<td>26042</td>
<td>22800</td>
<td>dharma/SQL JDBC errors</td>
<td>Remote procedure call error</td>
</tr>
<tr>
<td>26043</td>
<td>S1001</td>
<td>dharma/SQL JDBC errors</td>
<td>Memory allocation failure</td>
</tr>
<tr>
<td>26044</td>
<td>S1109</td>
<td>dharma/SQL JDBC errors</td>
<td>Invalid Cursor position</td>
</tr>
<tr>
<td>26045</td>
<td>S1010</td>
<td>dharma/SQL JDBC errors</td>
<td>Function sequence error</td>
</tr>
<tr>
<td>26046</td>
<td>24000</td>
<td>dharma/SQL JDBC errors</td>
<td>Invalid cursor state</td>
</tr>
<tr>
<td>26047</td>
<td>S1106</td>
<td>dharma/SQL JDBC errors</td>
<td>Fetch type out of range</td>
</tr>
<tr>
<td>26048</td>
<td>S1107</td>
<td>dharma/SQL JDBC errors</td>
<td>Row value out of range</td>
</tr>
<tr>
<td>26049</td>
<td>S1002</td>
<td>dharma/SQL JDBC errors</td>
<td>Invalid column number</td>
</tr>
<tr>
<td>26050</td>
<td>S1C00</td>
<td>dharma/SQL JDBC errors</td>
<td>Driver not capable</td>
</tr>
<tr>
<td>26051</td>
<td>S1T00</td>
<td>dharma/SQL JDBC errors</td>
<td>Timeout expired</td>
</tr>
<tr>
<td>26052</td>
<td>S1090</td>
<td>dharma/SQL JDBC errors</td>
<td>Invalid string or buffer length</td>
</tr>
<tr>
<td>26053</td>
<td>34000</td>
<td>dharma/SQL JDBC errors</td>
<td>Invalid cursor name</td>
</tr>
<tr>
<td>26054</td>
<td>3C000</td>
<td>dharma/SQL JDBC errors</td>
<td>Duplicate cursor name</td>
</tr>
<tr>
<td>26055</td>
<td>S1009</td>
<td>dharma/SQL JDBC errors</td>
<td>SQL argument value</td>
</tr>
<tr>
<td>26056</td>
<td>S1004</td>
<td>dharma/SQL JDBC errors</td>
<td>SQL data type out of range</td>
</tr>
<tr>
<td>26057</td>
<td>S1008</td>
<td>dharma/SQL JDBC errors</td>
<td>Operation canceled</td>
</tr>
<tr>
<td>26058</td>
<td>S1094</td>
<td>dharma/SQL JDBC errors</td>
<td>Invalid scale value</td>
</tr>
<tr>
<td>26059</td>
<td>S1104</td>
<td>dharma/SQL JDBC errors</td>
<td>Invalid precision value</td>
</tr>
<tr>
<td>26060</td>
<td>S1093</td>
<td>dharma/SQL JDBC errors</td>
<td>Invalid parameter number</td>
</tr>
<tr>
<td>26061</td>
<td>S1009</td>
<td>dharma/SQL JDBC errors</td>
<td>Invalid argument value</td>
</tr>
<tr>
<td>26062</td>
<td>08004</td>
<td>dharma/SQL JDBC errors</td>
<td>Data source rejected establishment of connection</td>
</tr>
<tr>
<td>26063</td>
<td>S1000</td>
<td>dharma/SQL JDBC errors</td>
<td>General error</td>
</tr>
<tr>
<td>26064</td>
<td>01002</td>
<td>dharma/SQL JDBC errors</td>
<td>Disconnect error</td>
</tr>
<tr>
<td>26065</td>
<td>IM001</td>
<td>dharma/SQL JDBC errors</td>
<td>Driver does not support this function</td>
</tr>
<tr>
<td>26066</td>
<td>IM002</td>
<td>dharma/SQL JDBC errors</td>
<td>Data source not found and no default driver specified</td>
</tr>
<tr>
<td>26067</td>
<td>IM003</td>
<td>dharma/SQL JDBC errors</td>
<td>Specified driver could not be loaded</td>
</tr>
<tr>
<td>26068</td>
<td>S1015</td>
<td>dharma/SQL JDBC errors</td>
<td>No cursor name available</td>
</tr>
<tr>
<td>26069</td>
<td>07006</td>
<td>dharma/SQL JDBC errors</td>
<td>Restricted data type attribute violation</td>
</tr>
<tr>
<td>30001</td>
<td>5050w</td>
<td>dharma/SQL rds errors</td>
<td>Query aborted on user request</td>
</tr>
<tr>
<td>30002</td>
<td>k0k02</td>
<td>dharma/SQL network interface</td>
<td>invalid network handle</td>
</tr>
<tr>
<td>Error Code</td>
<td>SQLSTATE Value</td>
<td>Class Condition</td>
<td>Subclass Message</td>
</tr>
<tr>
<td>------------</td>
<td>----------------</td>
<td>-----------------</td>
<td>------------------</td>
</tr>
<tr>
<td>30003</td>
<td>k0k03</td>
<td>dharma/ SQL network interface</td>
<td>invalid sqlnetwork INTERFACE</td>
</tr>
<tr>
<td>30004</td>
<td>k0k04</td>
<td>dharma/ SQL network interface</td>
<td>invalid sqlnetwork INTERFACE procedure</td>
</tr>
<tr>
<td>30005</td>
<td>k0k05</td>
<td>dharma/ SQL network interface</td>
<td>INTERFACE is already attached</td>
</tr>
<tr>
<td>30006</td>
<td>k0k06</td>
<td>dharma/ SQL network interface</td>
<td>INTERFACE entry not found</td>
</tr>
<tr>
<td>30007</td>
<td>k0k07</td>
<td>dharma/ SQL network interface</td>
<td>INTERFACE is already registered</td>
</tr>
<tr>
<td>30008</td>
<td>k0k08</td>
<td>dharma/ SQL network interface</td>
<td>mismatch in pkt header size and total argument size</td>
</tr>
<tr>
<td>30009</td>
<td>k0k09</td>
<td>dharma/ SQL network interface</td>
<td>invalid server id</td>
</tr>
<tr>
<td>30010</td>
<td>k0k10</td>
<td>dharma/ SQL network interface</td>
<td>reply does not match the request</td>
</tr>
<tr>
<td>30011</td>
<td>k0k02</td>
<td>dharma/ SQL network interface</td>
<td>memory allocation failure</td>
</tr>
<tr>
<td>30031</td>
<td>k0k11</td>
<td>dharma/ SQL network interface</td>
<td>error in transmission of packet</td>
</tr>
<tr>
<td>30032</td>
<td>k0k12</td>
<td>dharma/ SQL network interface</td>
<td>error in reception of packet</td>
</tr>
<tr>
<td>30033</td>
<td>k0k13</td>
<td>dharma/ SQL network interface</td>
<td>no packet received</td>
</tr>
<tr>
<td>30034</td>
<td>k0k14</td>
<td>dharma/ SQL network interface</td>
<td>connection reset</td>
</tr>
<tr>
<td>30051</td>
<td>k0k15</td>
<td>dharma/ SQL network interface</td>
<td>network handle is inprocess handle</td>
</tr>
<tr>
<td>30061</td>
<td>k0k16</td>
<td>dharma/ SQL network interface</td>
<td>could not connect to sql network daemon</td>
</tr>
<tr>
<td>30062</td>
<td>k0k17</td>
<td>dharma/ SQL network interface</td>
<td>error in number of arguments</td>
</tr>
<tr>
<td>30063</td>
<td>k0k18</td>
<td>dharma/ SQL network interface</td>
<td>requested INTERFACE not registered</td>
</tr>
<tr>
<td>30064</td>
<td>k0k19</td>
<td>dharma/ SQL network interface</td>
<td>invalid INTERFACE procedure id</td>
</tr>
<tr>
<td>30065</td>
<td>k0k20</td>
<td>dharma/ SQL network interface</td>
<td>requested server executable not found</td>
</tr>
<tr>
<td>30066</td>
<td>k0k21</td>
<td>dharma/ SQL network interface</td>
<td>invalid configuration information</td>
</tr>
<tr>
<td>30067</td>
<td>k0k22</td>
<td>dharma/ SQL network interface</td>
<td>INTERFACE not supported</td>
</tr>
<tr>
<td>30091</td>
<td>k0k23</td>
<td>dharma/ SQL network interface</td>
<td>invalid service name</td>
</tr>
<tr>
<td>30092</td>
<td>k0k24</td>
<td>dharma/ SQL network interface</td>
<td>invalid host</td>
</tr>
<tr>
<td>30093</td>
<td>k0k25</td>
<td>dharma/ SQL network interface</td>
<td>error in tcp/ ip accept call</td>
</tr>
<tr>
<td>30094</td>
<td>k0k26</td>
<td>dharma/ SQL network interface</td>
<td>error in tcp/ ip connect call</td>
</tr>
<tr>
<td>30095</td>
<td>k0k27</td>
<td>dharma/ SQL network interface</td>
<td>error in tcp/ ip bind call</td>
</tr>
<tr>
<td>30096</td>
<td>k0k28</td>
<td>dharma/ SQL network interface</td>
<td>error in creating socket</td>
</tr>
<tr>
<td>30097</td>
<td>k0k29</td>
<td>dharma/ SQL network interface</td>
<td>error in setting socket option</td>
</tr>
<tr>
<td>30101</td>
<td>k0k30</td>
<td>dharma/ SQL network interface</td>
<td>interrupt occured</td>
</tr>
<tr>
<td>30102</td>
<td>k0k31</td>
<td>dharma/ SQL network interface</td>
<td>Client/ Server not WideChar Compatible</td>
</tr>
<tr>
<td>40001</td>
<td>L0L01</td>
<td>dharma/ SQL env error</td>
<td>Error in reading configuration</td>
</tr>
<tr>
<td>50000</td>
<td>60614</td>
<td>dharma/ SQL DHRSS errors</td>
<td>DHRSS- Improper call to DFLT SS</td>
</tr>
</tbody>
</table>
Overview

The Dharma DataLink SDK maintains a set of system tables for storing information about table spaces, tables, columns, indexes, constraints, and privileges. These tables are called system catalog or dictionary tables.

SQL data definition statements and GRANT and REVOKE statements update system catalog tables. Users have read access to the system catalog tables. The database administrator has update access to the tables, but should avoid modifying them directly.

There are two types of tables in the system catalog: base tables and extended tables. Base tables store the information on the table spaces, tables, columns, and indexes that make up the database. The extended tables contain information on constraints, privileges, and statistical information.

The owner of the system tables is dharma. If you connect to a Dharma environment with a User ID other than dharma, you need to qualify references to the tables in SQL queries. For example:

```sql
SELECT * FROM Dharma.SYSTABLES
```

Table D-1 shows details of the columns in each system table. Here is the SQL query that generated the data for Table D-1. You can modify it to generate a similar list that includes user-created tables by omitting the line `and st.tbltype = 'S'`.

```sql
select sc.tbl 'Table', sc.col 'Column',
    sc.coltype 'Data Type', sc.width 'Size'
from systpe.syscolumns sc, systpe.systables st
where sc.tbl = st.tbl
    and st.tbltype = 'S'
order by sc.tbl, sc.id
```
Table D–1 lists all the tables in the system catalog. It gives a brief description of their purpose and lists the column definitions for every table.

**Table D–1  System Catalog Table Definitions**

<table>
<thead>
<tr>
<th>Table</th>
<th>Purpose</th>
<th>Column</th>
<th>Data Type</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>sys_chk_constrs</code></td>
<td>Contains the CHECK clause for each check constraint specified on a user table.</td>
<td><code>chkclause</code></td>
<td><code>varchar</code></td>
<td><code>2000</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>chkseq</code></td>
<td><code>integer</code></td>
<td><code>4</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>cnstrname</code></td>
<td><code>varchar</code></td>
<td><code>32</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>owner</code></td>
<td><code>varchar</code></td>
<td><code>32</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>tblname</code></td>
<td><code>varchar</code></td>
<td><code>32</code></td>
</tr>
<tr>
<td><code>sys_chkcol_usag</code></td>
<td>Contains one entry for each column on which the check constraint is specified</td>
<td><code>cnstrname</code></td>
<td><code>varchar</code></td>
<td><code>32</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>colname</code></td>
<td><code>varchar</code></td>
<td><code>32</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>owner</code></td>
<td><code>varchar</code></td>
<td><code>32</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>tblname</code></td>
<td><code>varchar</code></td>
<td><code>32</code></td>
</tr>
<tr>
<td><code>sys_keycol_usag</code></td>
<td>Contains one entry for each column on which primary or foreign key is specified</td>
<td><code>cnstrname</code></td>
<td><code>varchar</code></td>
<td><code>32</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>colname</code></td>
<td><code>varchar</code></td>
<td><code>32</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>colposition</code></td>
<td><code>integer</code></td>
<td><code>4</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>owner</code></td>
<td><code>varchar</code></td>
<td><code>32</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>tblname</code></td>
<td><code>varchar</code></td>
<td><code>32</code></td>
</tr>
<tr>
<td><code>sys_ref_constrs</code></td>
<td>Contains one entry for each referential constraint specified on a user table</td>
<td><code>cnstrname</code></td>
<td><code>varchar</code></td>
<td><code>32</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>deleterule</code></td>
<td><code>varchar</code></td>
<td><code>1</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>owner</code></td>
<td><code>varchar</code></td>
<td><code>32</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>refcnstrname</code></td>
<td><code>varchar</code></td>
<td><code>32</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>refowner</code></td>
<td><code>varchar</code></td>
<td><code>32</code></td>
</tr>
<tr>
<td>Table</td>
<td>Purpose</td>
<td>Column</td>
<td>Data Type</td>
<td>Size</td>
</tr>
<tr>
<td>---------------------</td>
<td>-------------------------------------------------------------------------</td>
<td>------------</td>
<td>-----------</td>
<td>------</td>
</tr>
<tr>
<td>reftblname</td>
<td></td>
<td>reftblname</td>
<td>varchar</td>
<td>32</td>
</tr>
<tr>
<td>tblname</td>
<td></td>
<td>tblname</td>
<td>varchar</td>
<td>32</td>
</tr>
<tr>
<td>sys_tbl_constrs</td>
<td>Contains one entry for each table constraint.</td>
<td>cnstrname</td>
<td>varchar</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td></td>
<td>cnstrtype</td>
<td>varchar</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>idxname</td>
<td>varchar</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td></td>
<td>owner</td>
<td>varchar</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td></td>
<td>tblname</td>
<td>varchar</td>
<td>32</td>
</tr>
<tr>
<td>syscalctable</td>
<td>Contains exactly one row with a single column with a value of 100.</td>
<td>fld</td>
<td>integer</td>
<td>4</td>
</tr>
<tr>
<td>syscolauth</td>
<td>Contains the update privileges held by users on individual columns of tables in the database.</td>
<td>col</td>
<td>varchar</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td></td>
<td>grantee</td>
<td>varchar</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td></td>
<td>grantor</td>
<td>varchar</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ref</td>
<td>varchar</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>tbl</td>
<td>varchar</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td></td>
<td>tblowner</td>
<td>varchar</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td></td>
<td>upd</td>
<td>varchar</td>
<td>1</td>
</tr>
<tr>
<td>syscolumns</td>
<td>Contains one row for each column of every table in the database.</td>
<td>charset</td>
<td>varchar</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td></td>
<td>col</td>
<td>varchar</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td></td>
<td>collation</td>
<td>varchar</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td></td>
<td>coltype</td>
<td>varchar</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>dflt_value</td>
<td>varchar</td>
<td>250</td>
</tr>
<tr>
<td></td>
<td></td>
<td>id</td>
<td>integer</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>nullflag</td>
<td>varchar</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>owner</td>
<td>varchar</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td></td>
<td>scale</td>
<td>integer</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>tbl</td>
<td>varchar</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td></td>
<td>width</td>
<td>integer</td>
<td>4</td>
</tr>
<tr>
<td>Table</td>
<td>Purpose</td>
<td>Column</td>
<td>Data Type</td>
<td>Size</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------------------------------------------------------------------</td>
<td>---------------</td>
<td>-----------</td>
<td>------</td>
</tr>
<tr>
<td>sysdatatypes</td>
<td>Contains information on each data type supported by the database.</td>
<td>autoincr</td>
<td>smallint</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>casesensitive</td>
<td>smallint</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>createparams</td>
<td>varchar</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td></td>
<td>datatype</td>
<td>smallint</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>dhtypename</td>
<td>varchar</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td></td>
<td>literalprefix</td>
<td>varchar</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>literalsuffix</td>
<td>varchar</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>localtypename</td>
<td>varchar</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>nullable</td>
<td>smallint</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>odbcmoney</td>
<td>smallint</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>searchable</td>
<td>smallint</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>typeprecision</td>
<td>integer</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>unsignedattr</td>
<td>smallint</td>
<td>2</td>
</tr>
<tr>
<td>sysdbauth</td>
<td>Contains the database-wide privileges held by users.</td>
<td>dba_acc</td>
<td>varchar</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>grantee</td>
<td>varchar</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td></td>
<td>res_acc</td>
<td>varchar</td>
<td>1</td>
</tr>
<tr>
<td>sysidxstat</td>
<td>Contains statistics for each index in the database.</td>
<td>idxid</td>
<td>integer</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>nleaf</td>
<td>integer</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>nlevels</td>
<td>smallint</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>recsz</td>
<td>integer</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>rssid</td>
<td>integer</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>tblid</td>
<td>integer</td>
<td>4</td>
</tr>
<tr>
<td>sysindexes</td>
<td>Contains one row for each component of an index in the database.</td>
<td>colname</td>
<td>varchar</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td></td>
<td>id</td>
<td>integer</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>idxcompress</td>
<td>varchar</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>idxmethod</td>
<td>varchar</td>
<td>1</td>
</tr>
<tr>
<td>Table</td>
<td>Purpose</td>
<td>Column</td>
<td>Data Type</td>
<td>Size</td>
</tr>
<tr>
<td>---------------</td>
<td>--------------------------------------------------------------------------</td>
<td>--------</td>
<td>-----------</td>
<td>------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>idxname</td>
<td>varchar</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td></td>
<td>idxorder</td>
<td>varchar</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>idxowner</td>
<td>varchar</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td></td>
<td>idxsegid</td>
<td>integer</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>idxseq</td>
<td>integer</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>idxtype</td>
<td>varchar</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>rssid</td>
<td>integer</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>tbl</td>
<td>varchar</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td></td>
<td>tblowner</td>
<td>varchar</td>
<td>32</td>
</tr>
<tr>
<td>syssynonyms</td>
<td>Contains one entry for each synonym in the database.</td>
<td>ispublic</td>
<td>smallint</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>screator</td>
<td>varchar</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td></td>
<td>sname</td>
<td>varchar</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td></td>
<td>sowner</td>
<td>varchar</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td></td>
<td>sremdb</td>
<td>varchar</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td></td>
<td>stbl</td>
<td>varchar</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td></td>
<td>stblowner</td>
<td>varchar</td>
<td>32</td>
</tr>
<tr>
<td>systabauth</td>
<td>Contains privileges held by users for tables, views, and procedures.</td>
<td>alt</td>
<td>varchar</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>del</td>
<td>varchar</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>exe</td>
<td>character</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>grantee</td>
<td>varchar</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td></td>
<td>grantor</td>
<td>varchar</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ins</td>
<td>varchar</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ndx</td>
<td>varchar</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ref</td>
<td>varchar</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>sel</td>
<td>varchar</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>tbl</td>
<td>varchar</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td></td>
<td>tblowner</td>
<td>varchar</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td></td>
<td>upd</td>
<td>varchar</td>
<td>1</td>
</tr>
<tr>
<td>Table</td>
<td>Purpose</td>
<td>Column</td>
<td>Data Type</td>
<td>Size</td>
</tr>
<tr>
<td>---------------</td>
<td>-------------------------------------------------------------------------</td>
<td>----------</td>
<td>-----------</td>
<td>------</td>
</tr>
<tr>
<td>systables</td>
<td>Contains one row for each table in the database.</td>
<td>creator</td>
<td>varchar</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td></td>
<td>has_ccnstrs</td>
<td>varchar</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>has_fcnstrs</td>
<td>varchar</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>has_pcnstrs</td>
<td>varchar</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>has_ucnstrs</td>
<td>varchar</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>id</td>
<td>integer</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>owner</td>
<td>varchar</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td></td>
<td>rssid</td>
<td>integer</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>segid</td>
<td>integer</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>tbl</td>
<td>varchar</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td></td>
<td>tbl_status</td>
<td>varchar</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>tblpctfree</td>
<td>integer</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>tbltype</td>
<td>varchar</td>
<td>1</td>
</tr>
<tr>
<td>systblspaces</td>
<td>No longer used.</td>
<td>id</td>
<td>integer</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>tsname</td>
<td>varchar</td>
<td>32</td>
</tr>
<tr>
<td>systblstat</td>
<td>Contains table statistics for each user table.</td>
<td>card</td>
<td>integer</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>npages</td>
<td>integer</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>pagesz</td>
<td>integer</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>recsz</td>
<td>integer</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>rssid</td>
<td>integer</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>tblid</td>
<td>integer</td>
<td>4</td>
</tr>
<tr>
<td>sysviews</td>
<td>Contains information on each view in the database.</td>
<td>creator</td>
<td>varchar</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td></td>
<td>owner</td>
<td>varchar</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td></td>
<td>seq</td>
<td>integer</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>viewname</td>
<td>varchar</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td></td>
<td>viewtext</td>
<td>varchar</td>
<td>2000</td>
</tr>
</tbody>
</table>
Maximum Values

Table E–1 lists the maximum sizes for various attributes of the ODBC Server environment.

Table E–1  ODBC Server System Limits

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum number of procedure arguments in an SQL CALL statement</td>
<td>TPE_MAX_PROC_ARGS</td>
<td>50</td>
</tr>
<tr>
<td>Maximum length of an SQL statement</td>
<td>TPE_MAX_SQLSTMTLEN</td>
<td>35000</td>
</tr>
<tr>
<td>Maximum length of a column: standard data types</td>
<td>TPE_MAX_FLDLEN</td>
<td>2000</td>
</tr>
<tr>
<td>Maximum length of a column: VARBINARY and VARCHAR specifying the character set designated as NATIONAL CHARACTER by the underlying storage system</td>
<td>TPE_EXT_MAX_FLDLEN</td>
<td>32752</td>
</tr>
<tr>
<td>Maximum length of default value specification</td>
<td>TPE_MAX_DFLT_LEN</td>
<td>250</td>
</tr>
<tr>
<td>Maximum length for an identifier (</td>
<td>TPE_MAX_IDLEN</td>
<td>32</td>
</tr>
<tr>
<td>Maximum length of a connect string</td>
<td>TPE_MAX_CONNLEN</td>
<td>100</td>
</tr>
<tr>
<td>Maximum length for a user-name in a connect string</td>
<td>TPE_UNAME_LEN</td>
<td>32</td>
</tr>
<tr>
<td>Maximum number of database connections</td>
<td>TPE_MAX_NO_CONN</td>
<td>10</td>
</tr>
<tr>
<td>Maximum length of an error message</td>
<td>TPE_MAX_ERRLEN</td>
<td>256</td>
</tr>
<tr>
<td>Maximum number of columns in a table</td>
<td>TPE_MAX_FIELDS</td>
<td>500</td>
</tr>
<tr>
<td>Maximum number of index components for a table, for all indexes on that table</td>
<td>SQL_MAXIDXENTRIES</td>
<td>100</td>
</tr>
</tbody>
</table>
## Maximum Values

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum length of a CHECK constraint clause</td>
<td>SQL_MAXCHKCL_SZ</td>
<td>240</td>
</tr>
<tr>
<td>Maximum number of check constraints in a table</td>
<td>SQL_MAXCHKCNSTRS</td>
<td>1000</td>
</tr>
<tr>
<td>Maximum number of foreign constraints in a table</td>
<td>SQL_MAXFRNCNSTRS</td>
<td>1000</td>
</tr>
<tr>
<td>Maximum number of nesting levels in an SQL statement</td>
<td>SQL_MAXLEVELS</td>
<td>25</td>
</tr>
<tr>
<td>Maximum number of table references in an SQL statement: Microsoft Windows</td>
<td>SQL_MAXTBLREF</td>
<td>50</td>
</tr>
<tr>
<td>Maximum number of table references in an SQL statement: other platforms</td>
<td>SQL_MAXTBLREF</td>
<td>250</td>
</tr>
<tr>
<td>Maximum size of input parameters for an SQL statement</td>
<td>SQL_MAXIPARAMS_SZ</td>
<td>512</td>
</tr>
<tr>
<td>Maximum number of outer references in an SQL statement</td>
<td>SQL_MAX_OUTER_REF</td>
<td>25</td>
</tr>
<tr>
<td>Maximum nesting level for view references</td>
<td>MAX_VIEW_LEVEL</td>
<td>25</td>
</tr>
</tbody>
</table>
**add [an ODBC data source]**
Make a data source available to ODBC through the Add operation of the ODBC Administrator utility. Adding a data source tells ODBC where a specific database resides and which ODBC driver to use to access it. Adding a data source also invokes a setup dialog box for the particular driver so you can provide other details the driver needs to connect to the database.

**cardinality**
Number of rows in a result table.

**client**
Generally, in client/server systems, the part of the system that sends requests to servers and processes the results of those requests.

**client/server configuration**
The version of the Dharma DataLink SDK that implements a network ODBC architecture. In the client/server configuration, the ODBC tool and a Dharma DataLink driver runs on Windows, while the ODBC SDK server runs on the UNIX or NT server hosting the proprietary storage system.

**data dictionary**
Another term for system catalog.

**data source**
See ODBC data source

**desktop configuration**
The version of the Dharma DataLink SDK that implements a “single-tier” ODBC architecture. In the desktop configuration, the ODBC tool, the Dharma DataLink SDK software, and the proprietary data all reside on the same Windows computer.

**dharma**
The default owner name for all system tables in a Dharma database. Users must qualify references to system tables as dharma.table-name.
metadata
Data that details the structure of tables and indexes in the proprietary storage system. The SQL engine stores metadata in the system catalog.

ODBC application
Any program that calls ODBC functions and uses them to issue SQL statements. Many vendors have added ODBC capabilities to their existing Windows-based tools.

ODBC data source
In ODBC terminology, a specific combination of a database system, the operating system it uses, and any network software required to access it. Before applications can access a database through ODBC, you use the ODBC Administrator to add a data source -- register information about the database and an ODBC driver that can connect to it -- for that database. More than one data source name can refer to the same database, and deleting a data source does not delete the associated database.

ODBC driver
Vendor-supplied software that processes ODBC function calls for a specific data source. The driver connects to the data source, translates the standard SQL statements into syntax the data source can process, and returns data to the application. The Dharma DataLink SDK includes an ODBC driver that provides access to proprietary storage systems underlying the ODBC server.

ODBC driver manager
A Microsoft-supplied program that routes calls from an application to the appropriate ODBC driver for a data source.

ODBC server
The executable that results from building an implementation of the storage interfaces with the SQL engine library. To get started with the ODBC SDK, you can build an ODBC Server from the supplied sample implementation of the storage interfaces. Eventually, you will build an ODBC Server from your own implementation of the storage system to provide access to a proprietary storage system.

primary key
A subset of the fields in a table, characterized by the constraint that no two records in a table may have the same primary key value, and that no fields of the primary key may have a null value. Primary keys are specified in a CREATE TABLE statement.

query expression
The fundamental element in SQL syntax. Query expressions specify a result table derived from some combination of rows from the tables or views identified in the FROM clause of the expression. Query expressions are the basis of SELECT, CREATE VIEW, and INSERT statements.

result set
Another term for result table.
result table
A temporary table of values derived from columns and rows of one or more tables that meet conditions specified by a query expression.

row identifier
Another term for tuple identifier.

search condition
The SQL syntax element that specifies a condition that is true or false about a given row or group of rows. Query expressions and UPDATE statements can specify a search condition. The search condition restricts the number of rows in the result table for the query expression or UPDATE statement. Search conditions contain one or more predicates. Search conditions follow the WHERE or HAVING keywords in SQL statements.

selectivity
The fraction of a table's rows returned by a query.

server
Generally, in client/server systems, the part of the system that receives requests from clients and responds with results to those requests.

SQL engine
The core component of the Dharma DataLink server. The SQL engine receives requests from the Dharma DataLink driver, processes them, and returns results.

storage interfaces
C routines called by the SQL engine that access and manipulate data in a proprietary storage system. A proprietary storage system must implement supplied storage stub templates to map the storage interfaces to the underlying storage system.

storage manager
A completed implementation of the Dharma DataLink SDK storage interfaces. A storage manager receives calls from the SQL engine and accesses the underlying proprietary storage system to retrieve and store data.

storage interfaces
Another term for stub interfaces.

storage system
The proprietary database system that underlies a storage manager. The Dharma DataLink SDK provides a SQL interface to a storage system through the SQL engine and its stub interfaces.

stub interfaces
Template routines provided with the Dharma DataLink SDK for implementing access to proprietary storage systems. Once filled in for a particular storage system, the completed stubs are called storage managers.
stubs
Another term for **stub interfaces**.

**system catalog**
Tables created by the SQL engine that store information about tables, columns, and indexes that make up the database. The SQL engine creates and manages the system catalog independent of the proprietary storage system.

**system tables**
Another term for **system catalog**.

**tid**
Another term for **tuple identifier**.

**transaction**
A group of database operations whose changes can be made permanent or undone only as a unit.

**tuple identifier**
A unique identifier for a tuple (row) in a table. Storage managers return a tuple identifier for the tuple that was inserted after an insert operation. The SQL engine passes a tuple identifier to the delete, update, and fetch stubs to indicate which tuple is affected. The SQL scalar function ROWID and related functions return tuple identifiers to applications.

**view**
A virtual table that recreates the result table specified by a SELECT statement. No data is stored in a view, but other queries can refer to it as if it were a table containing data corresponding to the result table it specifies.
Index

\{ 
\} 
\( \) escape clause for ODBC date literals, 4–33
\( t \) escape clause for ODBC time literals, 4–34
\( ts \) escape clause for ODBC timestamp literals, 4–34

= 
<= != relational operators, 4–22

A 
-a password argument, A–4
ABS scalar function, 4–40
ACOS scalar function, 4–40
Active connections, maximum number, 3–2, 3–9
Adding ODBC data sources, 2–2
  Windows (client/server), 2–2
Aggregate functions, 4–37
  AVG, 4–38
  COUNT, 4–38
  MAX, 4–39
  MIN, 4–39
  SUM, 4–39
Aliases, 4–14, 4–15, 4–27
  column aliases, 4–15
ALL argument to SELECT clause, 4–13
ALTER privilege, 5–21, 5–27
ALTER TABLE clauses, 3–2
AND logical operator, 4–22
Application Programming Interface (API) for
  ODBC
    supporting all Core and some functions, 3–1
    Supporting all Core and some functions, 3–18
Arithmetic expressions
  date, 4–29
  numeric, 4–28
Ascending indexes, 5–3
ASCII character set, 4–6, 4–7

ASCII scalar function, 4–41
ASCII_SET character set, 4–7, 4–8
ASIN scalar function, 4–42
ATAN scalar function, 4–42
ATAN2 scalar function, 4–43
AVG aggregate function, 4–38

B 
Basic predicates, 4–23
  query expressions in, 4–23
BETWEEN predicates, 4–24
BIGINT data type, 3–3
BINARY data type, 3–3, 4–11
BINGINT data type, 4–9
BIT data type, 3–3, 4–11
Bookmarks, 3–2

C 
CALL statement, 5–2
Cartesian product, 4–18
CASCADE clause of REVOKE, 5–27
CASE scalar function, 4–43
CASE_SENSITIVE collation sequence, 4–7
CAST scalar function, 4–46
CEILING scalar function, 4–46
Changing table and view names with 
  RENAME, 5–26
CHAR data type, 3–3
CHAR scalar function, 4–47
CHARACTER data type, 4–5
Character expressions
  concatenated, 4–28
  string literals, 4–32
Character sets
  ASCII, 4–6
  ASCII_SET, 4–7
  CASE_SENSITIVE collation, 4–7
  COLLATE clause
    in basic predicates, 4–23
    in column definitions, 5–7
in GROUP BY clause of query expressions, 4–16
in ORDER BY clause of select statements, 5–31
collation, 4–7
default, 4–6
default national character, 4–8
form of use, 4–6
names, 4–5, 4–6
octets, 4–6
repertoire, 4–6
specified by NATIONAL CHARACTER, 4–6
Unicode, 4–6
Character string literals, 4–32
CHARACTER VARYING data type, 4–5
CHARTOROWID scalar function, 4–47
CHR scalar function, 4–48
Clause
ORDER BY, 3–7
COALESCE scalar function, 4–48
COLLATE clause, 4–7
CASE_INSENSITIVE collation sequence, 4–7
in basic predicates, 4–23
in column definitions, 5–7
example, 4–7
in GROUP BY clause of query expressions, 4–16
in ORDER BY clause of SELECT statements, 5–31
Collations of character sets, 4–7
CASE_INSENSITIVE collation sequence, 4–7
in basic predicates, 4–23
in column definitions, 5–7
example, 4–7
in GROUP BY clause of query expressions, 4–16
in ORDER BY clause of select statements, 5–31
Column aliases, 3–3, 4–15
Column constraints in CREATE TABLE, 5–9
Column names, 4–3, 4–14, 4–15, 4–27
aliases in FROM clause, 4–15
in GROUP BY clause of query expressions, 4–16
in indexes, 5–3
in tables, 5–6
in views, 5–14
Column titles, 4–14
COMMIT operation, 3–6
CONCAT scalar function, 4–49
Concatenated character expressions
with CONCAT function, 4–49
with concatenation operator, 4–28
Conditional expressions, 4–30
CASE, 4–31
COALESCE, 4–31, 4–48
DECODE, 4–31, 4–54
IFNULL, 4–31, 4–57
INSERT, 4–58
NULLIF, 4–31, 4–67
subset of scalar functions, 4–30
summary, 4–30
Constants, 4–31
Constraints
column constraints in CREATE TABLE, 5–9
table constraints in CREATE TABLE, 5–11
CONTAINS predicate, 4–24
Controlling format of date-time values, 4–35
Conventional Identifiers, 4–3
Conventions, viii
Conversion of data types supported, 3–3
CONVERT scalar function
extension, 4–49
ODBC compatible, 4–50
Correlation names, 4–14, 4–15, 4–27
COS scalar function, 4–51
COUNT aggregate function, 4–38
CREATE INDEX statement, 5–3
CREATE SQL statements, support for NOT NULL clause, 3–10
CREATE SYNONYM statement, 5–5
CREATE TABLE statement, 5–5
column constraints, 5–9
DEFAULT clause, 5–7
table constraints, 5–11
CREATE VIEW statement, 5–14
Creating a data dictionary with mdcreate, A–3
CROSSJOIN syntax, 4–18
CURDATE scalar function, 4–51
CURTIME scalar function, 4–52
D
-d argument to mdcreate, A–3
Data dictionary
creating, A–3
loading, A–4
specifying alternate location for (desktop only), A–3
Data sources for ODBC, 2–2
adding for Windows (client/server), 2–3
Data sources, adding, 2–2
Data Types, 5–6
allowed in CONTAINS predicates, 4–24
approximate numeric, 4–10
BIGINT, 4–9
BIT, 4–11
INDEX, 4–11
<table>
<thead>
<tr>
<th>Data Type</th>
<th>Page Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>bit string</td>
<td>4-11</td>
</tr>
<tr>
<td>CHARACTER</td>
<td>4-5</td>
</tr>
<tr>
<td>CHARACTER VARYING</td>
<td>4-5</td>
</tr>
<tr>
<td>DATE</td>
<td>4-10</td>
</tr>
<tr>
<td>date-time</td>
<td>4-10</td>
</tr>
<tr>
<td>DECIMAL</td>
<td>4-9</td>
</tr>
<tr>
<td>DOUBLE PRECISION</td>
<td>4-10</td>
</tr>
<tr>
<td>FLOAT</td>
<td>4-10</td>
</tr>
<tr>
<td>LONG VARBINARY</td>
<td>4-12</td>
</tr>
<tr>
<td>LONG VARCHAR</td>
<td>4-6</td>
</tr>
<tr>
<td>MONEY</td>
<td>4-9</td>
</tr>
<tr>
<td>NATIONAL CHARACTER</td>
<td>4-5, 4-6, 4-8</td>
</tr>
<tr>
<td>NATIONAL CHARACTER VARYING</td>
<td>4-6</td>
</tr>
<tr>
<td>NUMBER</td>
<td>4-9</td>
</tr>
<tr>
<td>NUMERIC</td>
<td>4-9</td>
</tr>
<tr>
<td>REAL</td>
<td>4-10</td>
</tr>
<tr>
<td>SMALLINT</td>
<td>4-9</td>
</tr>
<tr>
<td>TIME</td>
<td>4-10</td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td>4-11</td>
</tr>
<tr>
<td>TINYINT</td>
<td>4-9</td>
</tr>
<tr>
<td>VARBINARY</td>
<td>4-11</td>
</tr>
<tr>
<td>VARCHAR</td>
<td>4-5, 4-9</td>
</tr>
</tbody>
</table>

Data types, conversions, 3-3
DATABASE scalar function, 4-52
Date arithmetic expressions, 4-29
DATE data type, 3-3, 4-10
Date format strings, 4-35
Date literals, 4-32
ODBC escape clause, 4-33
Date-time functions, 3-14
ADD_MONTHS, 4-41
CURDATE, 4-51
CURTIME, 4-52
DAYNAME, 4-52
DAYOFMONTH, 4-53
DAYOFWEEK, 4-53
DAYOFYEAR, 4-53
HOUR, 4-57
LAST_DAY, 4-59
MINUTE, 4-64
MONTH, 4-65
MONTHNAME, 4-65
MONTHS_BETWEEN, 4-66
NEXT_DAY, 4-66
NOW, 4-67
QUARTER, 4-70
SECOND, 4-75
SYSDATE, 4-80
SYSTIME, 4-80
SYSTIMESTAMP, 4-80
TIMESTAMPADD, 4-81
TIMESTAMPDIFF, 4-82
TO_CHAR, 4-83
TO_DATE, 4-84
TO_TIME, 4-84
TO_TIMESTAMP, 4-85
WEEK, 4-88
YEAR, 4-88
Date-time literals, 4-32
DAYNAME scalar function, 4-52
DAYOFMONTH scalar function, 4-53
DAYOFWEEK scalar function, 4-53
DAYOFYEAR scalar function, 4-53
DB_NAME scalar function, 4-54
DBA privilege, 5-21, 5-27
DECIMAL data type, 3-3, 4-9
DECODE scalar function, 4-54
Default character set, ASCII, 4-6
DEFAULT clause of CREATE TABLE statement, 5-7
Default national character set, ASCII_SET, 4-8
DEGREES scalar function, 4-55
DELETE privilege, 5-21, 5-27
DELETE statement, 5-16
Delimited identifiers, 3-7, 4-4
Derived tables, 4-15
column aliases for, 4-15
Descending indexes, 5-3
Desktop configuration
specifying alternate location for data dictionary, A-3
Dharma/SQL ODBC driver
adding data sources for, 2-2
DIFFERENCE scalar function, 4-55
Directory option for mdcreate, A-3
DISTINCT argument to SELECT clause, 4-13
DOUBLE data type, 3-3
DOUBLE PRECISION data type, 4-10
Driver manager, ODBC, 1-2
Driver, ODBC, 1-2
DROP INDEX statement, 5-17
DROP SYNONYM statement, 5-18
DROP TABLE statement, 5-19
DROP VIEW statement, 5-20
Dropping database objects, privileges required for, 5-21

E
Environment variables
TPE_DATA_DIR, A-3, A-4
TPEROOT, A-3
Equi-joins, 4-19
Escape clause
{d } for date literals, 4-33
{t } for time literals, 4-34
{ts } for timestamp literals, 4-34
in LIKE predicate, 4-25
ODBC for date literals, 4-33
ODBC for time literals, 4-34
ODBC for timestamp literals, 4-34
ESCAPE clause in LIKE predicate, 4-25
Exact numeric data types, 4-8
EXECUTE privilege, 5-21, 5-27
Execute privileges guaranteed, 3-2
EXISTS predicates, 4-25
EXP scalar function, 4-56
Expressions, 4-26
comparing with relational operators, 4-22
concatenated character, 4-28
conditional, 4-30
date arithmetic, 4-29
numeric arithmetic, 4-28

F
FETCH operations, 3-7
fInfoType, 3-2
FLOAT data type, 3-4, 4-10
FLOOR scalar function, 4-56
FOR UPDATE clause of SELECT statement, 5-31
Form of use of character sets, 4-6
Format strings
date, 4-35
date time values, 4-35
TIME, 4-37
use TO_CHAR function to specify, 4-83
FROM clause, 3-10
FROM clause of query expression, 4-14
{dharma ORDERED}, 4-15
query expressions in, 4-15
FROM user_name clause of REVOKE, 5-27
Functions, 4-37
aggregate functions, 4-37
AVG, 4-38
COUNT, 4-38
MAX, 4-39
MIN, 4-39
SUM, 4-39
List of supported, 3-18
scalar functions, 4-37
ABS, 4-40
ACOS, 4-40
ADD_MONTHS, 4-41
ASCII, 4-41
ASIN, 4-42
ATAN, 4-42
ATAN2, 4-43
CASE, 4-43
CAST, 4-46
CEILING, 4-46
CHAR, 4-47
CHARTOROWID, 4-47
CHR, 4-48
COALESCE, 4-48
CONCAT, 4-49
CONVERT, extension, 4-49
CONVERT, ODBC compatible, 4-50
COS, 4-51
CURDATE, 4-51
CURTIME, 4-52
DATABASE, 4-52
DAYNAME, 4-52
DAYOFMONTH, 4-53
DAYOFWEEK, 4-53
DAYOFYEAR, 4-53
DB_NAME, 4-54
DECODE, 4-54
DEGREES, 4-55
DIFFERENCE, 4-55
EXP, 4-56
FLOOR, 4-56
GREATEST, 4-57
HOUR, 4-57
IFNULL, 4-57
INITCAP, 4-58
INSERT, 4-58
INSTR, 4-59
LAST_DAY, 4-59
LCASE, 4-60
LEAST, 4-60
LEFT, 4-61
LENGTH, 4-61
LOCATE, 4-62
LOG10, 4-62
LOWER, 4-62
LPAD, 4-63
LTRIM, 4-64
MINUTE, 4-64
MOD, 4-64
MONTH, 4-65
MONTHNAME, 4-65
MONTHS_BETWEEN, 4-66
NEXT_DAY, 4-66
NOW, 4-67
NULLIF, 4-67
NVL, 4-67
OBJECT_ID, 4-68
PI, 4-68
POWER, 4-69
PREFIX, 4-69
QUARTER, 4-70
RADIANS, 4-70
date-time, 4–32
numeric, 4–31
time, 4–33
timestamp, 4–34
Loading metadata with mdsql, A–4
LOCATE scalar function, 4–62
Lock types, 3–8
LOG10 scalar function, 4–62
Logical operators, 4–22
LONG VARBINARY data type, 4–12
LONG VARCHAR data type, 4–6
LONGVARCHAR data type, 3–4, 3–10
LOWER scalar function, 4–62
LPAD scalar function, 4–63
LTRIM scalar function, 4–64
MAX aggregate function, 4–39
mdcreate, A–3
arguments, A–3
syntax, A–3
mdsql, A–4
-a password argument, A–4
arguments, A–4
-s script_file argument, A–4
syntax, A–4
-u user_name argument, A–4
Metadata
creating dictionary with mdcreate, A–3
loading with mdsql, A–4
Microsoft ODBC, 3–1
MIN aggregate function, 4–39
MINUS set operator, 4–17
MINUTE scalar function, 4–64
MOD scalar function, 4–64
Modifying
table and view names with RENAME, 5–26
MONEY data type, 4–9
MONTH scalar function, 4–65
MONTHNAME scalar function, 4–65
MONTHS_BETWEEN scalar function, 4–66

N
Names
alias, 4–14, 4–15, 4–27
character set, 4–5, 4–6
column, 4–3, 4–14, 4–15, 4–27
column names in indexes, 5–3
column names in views, 5–14
correlation, 4–14, 4–15, 4–27
index, 5–3
object_id scalar function, 4–68
Octet, 4–6
ODBC
application, 1–2
driver manager, 1–2
drivers, 1–2
ODBC Administrator, 2–2
ODBC data sources, adding, 2–2
Windows (client/server), 2–3
ODBC driver
functions, 3–1
ODBC drivers
for other data sources, 1–2
ODBC escape clause
for date literals, 4–33
for time literals, 4–34
for timestamp literals, 4–34
Open Database Connectivity (ODBC)
administrator utility, 2–2
API functions, 3–11
Operators
concatenation, 4–28
logical, 4–22
relational, 4–22
relational, in quantified predicates, 4–23
set
INTERSECT, 4–17
MINUS, 4–17
UNION, 4–16
OR logical operator, 4–22
ORDER BY clause, 3–7, 3–9
schema, 5–32
table, 4–3, 4–14, 4–15, 4–27, 5–6
table names in indexes, 5–3
view, 4–3, 5–14
NATIONAL CHARACTER data type, 4–5
NATIONAL CHARACTER data type, character
column names in indexes, 5–3
set
INTERSECT, 4–17
MINUS, 4–17
UNION, 4–16
OR logical operator, 4–22
ORDER BY clause, 3–7, 3–9
ORDER BY clause of SELECT statement, 5–30
Outer join predicates, 4–26
Outer joins, 3–11
Output of date-time values, controlling, 4–35
Overriding join optimizations with \{dharma ORDERED\}, 4–15

P
PI scalar function, 4–68
POWER scalar function, 4–69
Precision
   of DECIMAL data type, 4–9
   of FLOAT data type, 4–10
   of MONEY data type, 4–9
   of NUMERIC data type, 4–9
Predicates
   basic, 4–23
   BETWEEN, 4–24
   components of search conditions, 4–21
   CONTAINS, 4–24
   EXISTS, 4–25
   IN, 4–26
   LIKE, 4–25
   NULL, 4–24
   outer join, 4–26
   quantified, 4–23
   relational operators in, 4–22
PREFIX scalar function, 4–69
Privileges
   for dropping database objects, 5–21
   granting, 5–21
   revoking, 5–27
Procedures
   changing default qualifier for with SET SCHEMA, 5–32
   granting EXECUTE privilege, 5–21
   revoking EXECUTE privilege, 5–27
PUBLIC granting of privileges, 5–21
PUBLIC revoking of privileges, 5–27

Q
Qualifying database object names, 5–32
Quantified predicates, 4–23
   query expressions in, 4–23
   QUARTER scalar function, 4–70
Query Expressions, 4–12
   \{dharma ORDERED\} clause, 4–15
   FROM clause, 4–14
   GROUP BY clause, 4–16
   HAVING clause, 4–16
   in basic predicates, 4–23
   in FROM clause of a query expression, 4–15
   in IN predicates, 4–26
   in INSERT statements, 5–24
   in quantified predicates, 4–23
   in search conditions, 4–21
   in SELECT statements, 5–29
   in UPDATE statements, 5–34
   INTERSECT clause, 4–17
   MINUS clause, 4–17
   SELECT clause, 4–13
   select list, 4–13
   table references, 4–14, 4–15
   UNION clause, 4–16
   WHERE clause, 4–15

R
RADIANS scalar function, 4–70
RAND scalar function, 4–71
Read access to all table names, 3–2
Read-only, access limited, 3–6
REAL data type, 3–4, 4–10
REFERENCES privilege, 5–21, 5–27
References to tables in query expressions, 4–14
   derived tables, 4–15
   explicit, 4–14
   joined tables, 4–15
Relational operators, 4–22
   in quantified predicates, 4–23
RENAME statement, 5–26
REPEAT scalar function, 4–72
Repertoire of character sets, 4–6
REPLACE scalar function, 4–71
Reserved word list, 3–6
Reserved words, B–1
RESOURCE privilege, 5–21, 5–27
RESTRICT clause of REVOKE, 5–27
Returned values for SQLGetInfo function, 3–1
REVOKE statement, 5–27
RIGHT scalar function, 4–71
ROLLBACK operation, 3–6
Row identifiers
   CHARTOROWID function, 4–47, 4–73
   ROWID function, 4–72
   ROWID scalar function, 4–72
   ROWIDTOCHAR scalar function, 4–73
   RPAD scalar function, 4–73
   RTRIM scalar function, 4–74

S
-script_file argument, A–4
Scalar expressions. See Expressions
Scalar functions, 4–37
ABS, 4–40
ACOS, 4–40
ADD_MONTHS, 4–41
ASCII, 4–41
ASIN, 4–42
ATAN, 4–42
ATAN2, 4–43
CASE, 4–43
CAST, 4–46
CEILING, 4–46
CHAR, 4–47
CHARTOROWID, 4–47
CHR, 4–48
COALESCE, 4–48
CONCAT, 4–49
conditional expressions subset, 4–30
CONVERT, extension, 4–49
CONVERT, ODBC compatible, 4–50
COS, 4–51
CURDATE, 4–51
CURTIME, 4–52
DATABASE, 4–52
DAYNAME, 4–52
DAYOFMONTH, 4–53
DAYOFWEEK, 4–53
DAYOFYEAR, 4–53
DB_NAME, 4–54
DECODE, 4–54
DEGREES, 4–55
DIFFERENCE, 4–55
EXP, 4–56
FLOOR, 4–56
GREATEST, 4–57
HOUR, 4–57
IFNULL, 4–57
INITCAP, 4–58
INSERT, 4–58
INSTR, 4–59
LAST_DAY, 4–59
LCASE, 4–60
LEAST, 4–60
LEFT, 4–61
LENGTH, 4–61
LOCATE, 4–62
LOG10, 4–62
LOWER, 4–62
LPAD, 4–63
LTRIM, 4–64
MINUTE, 4–64
MOD, 4–64
MONTH, 4–65
MONTHNAME, 4–65
MONTHS_BETWEEN, 4–66
NEXT_DAY, 4–66
NOW, 4–67
NULLIF, 4–67
NVL, 4–67
OBJECT_ID, 4–68
PI, 4–68
POWER, 4–69
PREFIX, 4–69
QUARTER, 4–70
RADIANS, 4–70
RAND, 4–71
REPEAT, 4–72
REPLACE, 4–72
RIGHT, 4–71
ROWID, 4–72
ROWIDTOCHAR, 4–73
RPAD, 4–73
RTRIM, 4–74
SECOND, 4–75
SIGN, 4–75
SIN, 4–75
SOUNDEX, 4–76
SPACE, 4–76
SQRT, 4–77
SUBSTR, 4–77
SUBSTRING, 4–78
SUFFIX, 4–78
SUSER_NAME, 4–79
SYSDATE, 4–80
SYSTIME, 4–80
SYSTIMESTAMP, 4–80
TAN, 4–81
TIMESTAMPADD, 4–81
TIMESTAMPDIFF, 4–82
TO_CHAR, 4–83
TO_DATE, 4–84
TO_NUMBER, 4–84
TO_TIME, 4–84
TO_TIMESTAMP, 4–85
TRANSLATE, extension, 4–85
UCASE, 4–86
UID, 4–86
UPPER, 4–87
USER, 4–87
USER_NAME, 4–88
WEEK, 4–88
YEAR, 4–88
Scale
of DECIMAL data type, 4–9
of MONEY data type, 4–9
of NUMERIC data type, 4–9
Schema
as default table qualifier, 5–32
Search conditions, 4-21
  contain one or more predicates, 4-21
    in HAVING clause of query expressions, 4-16
    in inner joins, 4-18, 4-19
    in WHERE clause of query expressions, 4-15
SECOND scalar function, 4-75
SELECT clause of query expression, 4-13
Select list of query expression, 4-13
SELECT privilege, 5-21, 5-27
SELECT SQL statements
  maximum number of columns allowed, 3-9
  maximum number of tables allowed in FROM clause, 3-10
  relationship between GROUP BY clause and columns in the select list, 3-7
  statements that support positioned operations, 3-11
  whether columns in ORDER BY clause must also be in select list, 3-11
SELECT statement, 5-29
  FOR UPDATE clause, 5-31
  ORDER BY clause, 5-30
  query expressions in, 5-29
Self joins, 4-19
Set operators
  INTERSECT, 4-17
  MINUS, 4-17
  UNION, 4-16
SET SCHEMA statement, 5-32
SIGN scalar function, 4-75
SIN scalar function, 4-75
SMALLINT data type, 3-5, 4-9
Sorting rows with ORDER BY clause, 5-30
SOUNDEX scalar function, 4-76
SPACE scalar function, 4-76
SQL data types
  list of supported, 3-19
SQL keywords, B-1
SQL reserved words, B-1
SQL statements
  CREATE, 3-10
  SELECT, 3-7
SQLGetInfo function
  values returned for, 3-1
SQRRT scalar function, 4-77
Stored procedures
  changing default qualifier for with SET SCHEMA, 5-32
  granting EXECUTE privilege, 5-21
  revoking EXECUTE privilege, 5-27
String literals, 4-32
SUBSTR scalar function, 4-77
SUBSTRING scalar function, 4-78
SUM aggregate function, 4-39
SUFFIX scalar function, 4-78
SUSER_NAME scalar function, 4-79
Synonyms
  changing with RENAME, 5-26
    creating, 5-5
    dropping, 5-18
  in FROM clause of query expression, 4-14
Syntax
  mdcreate, A-3
  mdsql, A-4
SYSDATE scalar function, 4-80
SYSTIME scalar function, 4-80
SYSTIMESTAMP scalar function, 4-80
T
  Table constraints in CREATE TABLE, 5-11
  Table correlation names, 3-5
  Table names, 4-3, 4-14, 4-15, 4-27, 5-6
    changing default qualifier for with SET SCHEMA, 5-32
    changing with RENAME, 5-26
  in indexes, 5-3
  Table references in a query expression, 4-14
    derived tables, 4-15
    explicit, 4-14
    joined tables, 4-15
  Table terms, 3-14
  Tables
    creating, 5-6
    deleting, 5-19
    derived, 4-15
      column aliases for, 4-15
      granting privileges for, 5-21
      revoking privileges for, 5-27
  TAN scalar function, 4-81
  TIME data type, 3-5, 4-10
  Time format strings, 4-37
  Time literals, 4-33
    ODBC escape clause, 4-34
  TIMESTAMP data type, 3-5, 4-11
  Timestamp literals, 4-34
    ODBC escape clause, 4-34
  TIMESTAMPA DD scalar function, 3-14, 4-81
  TIMESTAMPDIFF scalar function, 4-82
  TINYINT data type, 3-5, 4-9
  Titles, for columns in select lists, 4-14
  TO user_name clause of GRANT, 5-21
  TO_CHAR scalar function, 4-83
    specifying date-time format strings in, 4-35
  TO_DATE scalar function, 4-84
  TO_NUMBER scalar function, 4-84
  TO_TIME scalar function, 4-84
TO_TIMESTAMP scalar function, 4–85
TPE_DATADIR environment variable, A–3, A–4
TPEROOT environment variable, A–3
Transaction isolation level, 3–6, 3–15
TRANSLATE scalar function
   extension, 4–85
Triggers
   changing default qualifier for with SET
      SCHEMA, 5–32
Trigonometric functions
   ACOS, 4–40
   ASIN, 4–42
   ATAN, 4–42
   ATAN2, 4–43
   COS, 4–51
   SIN, 4–75
   TAN, 4–81
TYPE clause of CREATE INDEX, 5–4
   required for CONTAINS predicates, 4–24

U
   -u user_name argument, A–4
   UCASE scalar function, 4–86
   UUID scalar function, 4–86
   Unicode character set, 4–6
   UNION set operator, 4–16
   UNION support, 3–15
   Unique indexes, 5–3
   UPDATE privilege, 5–21, 5–27
   UPDATE statement, 5–34
      query expressions in, 5–34
   search conditions in, 4–21
   UPPER scalar function, 4–87
   USER scalar function, 4–87
   USER_NAME scalar function, 4–88

V
   -v argument to mdcreate, A–3
   Value expressions. See Expressions
   Values, specifying default values in CREATE
      TABLE statement, 5–7
   VARCHAR data type, 3–5, 4–11
   View names, 4–3, 5–14
      changing default qualifier for with SET
         SCHEMA, 5–32
      changing with RENAME, 5–26

W
   WEEK scalar function, 4–88
   WHERE clause of query expression, 4–15
   Windows
      adding ODBC data sources (client/ server), 2–3
   WITH GRANT OPTION clause of GRANT, 5–21

Y
   YEAR scalar function, 4–88