JDBC Driver Guide

July 2005

Version 9.1

This manual gives an overview of the JDBC (Java Database Connectivity) interface and the Dharma SDK JDBC Driver. It describes how to set up and use the driver and details the driver's support for the JDBC interface. The JDBC Driver provides access to Dharma SDK environments from applications that support JDBC.

July 2005

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

Documentation Comments

Dharma Systems, Inc.

Brookline Business Center.

#55, Route 13

Brookline, NH 03033

Phone: 603-732-4001

Fax: 603-732-4003

Electronic Mail: support@dharma.com

Web Page: http://www.dharma.com

Contents

Tables

Examples

Introduction

PURPOSE OF THIS MANUAL

This manual gives an overview of the JDBC (Java Database Connectivity) interface and the Dharma JDBC Driver. It describes how to set up and use the driver and details the driver's support for the JDBC interface. The JDBC Driver provides access to the Dharma SDK environments from applications that support JDBC.

AUDIENCE

This manual is directed towards application programmers writing database applications using Dharma SDK. It assumes knowledge of the Java™ programming language.

STRUCTURE

This manual contains the following chapters:

Chapter 1	Introduces the Dharma SDK JDBC driver and describes how it works.
Chapter 2	Describes basic JDBC Driver operations: required software, setup, using the sample program, connecting to databases, and managing transactions.
Chapter 3	Details the information returned by the JDBC driver to the DatabaseMetadata methods and lists the supported SQL and corresponding Java data types.
Appendix A	Contains a glossary of terms you should know.

SYNTAX DIAGRAM 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.

RELATED DOCUMENTATION

Refer to the following manuals for more information:

Dharma SDK SQL Reference Manual	Describes the syntax and semantics of statements and language elements for the Dharma SDK interface.
Dharma SDK User Guide	This manual describes the Dharma Software Development Kit (SDK).
Dharma SDK ISQL Reference Manual	Describes the Dharma SDK interactive SQL utility, ISQL, and other database administration utilities.
Dharma SDK ODBC Driver Guide	Describes Dharma SDK support for the ODBC interface and how to configure the Dharma SDK ODBC Driver.
Dharma SDK .NET Data Provider Guide	Describes how to set up and use the Dharma SDK .NET Data Provider to access Dharma SDK databases from .NET applications.

Java Information

James Gosling & Henry McGilton

"The Java™ Language Environment: A White Paper"

http://java.sun.com/docs/white/langenv/

Mary Campione and Kathy Walrath

The Java™ Tutorial

http://java.sun.com/docs/books/tutorial/index.html

Java™ Platform 1.3 Core API

http://www.javasoft.com:80/products/jdk/1.2/docs/api/packages.html

Gary Cornell and Cay Horstmann

Core Java 1.1 Volume 1: The Fundamentals

Prentice Hall, Upper Saddle River, NJ, 1997

JDBC Information

JavaSoft's JDBC home page

http://java.sun.com/products/jdbc/

George Reese

Database Programming with JDBC and Java

O'Reilly and Associates, Sebastopol, CA, 1997

ODBC Information

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

Chapter 1

Introduction

1.1 OVERVIEW

The Dharma JDBC Driver provides access to Dharma SDK environments from applications that support the JDBC 3.0 API.

JDBC allows applications to connect to any database using the same set of Java interfaces. Those interfaces allow programs to embed standard Structured Query Language (SQL) statements that update and retrieve data in the database.

Because the Java interfaces and SQL syntax are independent of any particular database implementation, JDBC makes it feasible for applications to connect to different database environments without any modification.

1.2 JDBC ARCHITECTURE

JDBC insulates Java applications from variations in database access implementations through the JDBC API, a set of class libraries distributed as a standard part of core Java. Instead of using calls to vendor-specific interfaces, JDBC applications use the JDBC API.

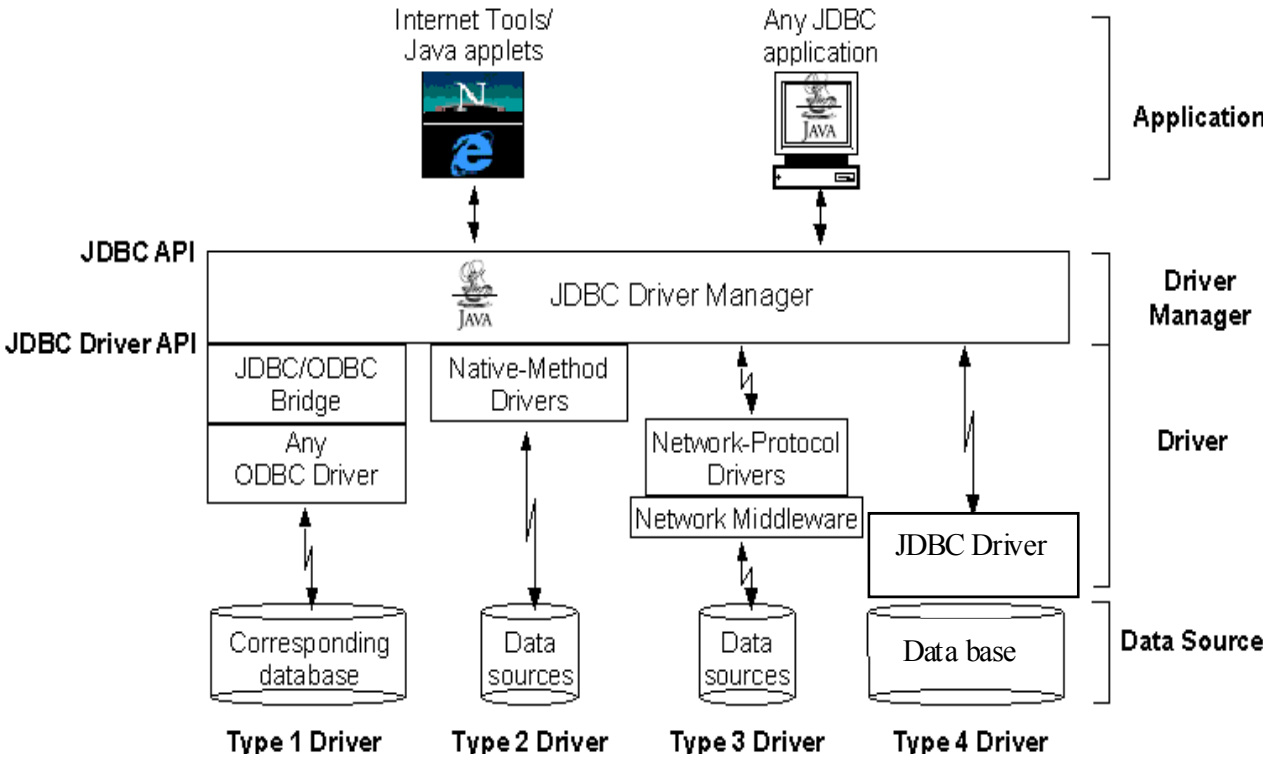
The JDBC API is distributed as the package *java.sql* and is included with the JavaSoft JDK (Version 1.4 or later), so any environment that supports a recent Java compiler can be used to develop JDBC applications.

Calls to the JDBC API are managed by the JDBC driver manager. The JDBC driver manager can support multiple drivers connecting to different databases. When an application tries to connect to a particular database, the driver manager loads the appropriate JDBC driver and routes subsequent calls through the driver.

A JDBC driver is a database-specific software that receives calls from the JDBC driver manager, translates them into a form that the database can process, and returns data to the application.

The following figure shows the different components of the JDBC architecture.

Figure 1-1: JDBC Architecture



1.3 TYPES OF JDBC DRIVERS

JDBC drivers can either be entirely written in Java so that they can be downloaded as part of an applet, or they can be implemented using native methods to bridge to existing database access libraries.

JavaSoft defines four different types of JDBC drivers, as noted in the previous figure and outlined in the following sections.

1.3.1 JDBC-ODBC Bridge Drivers

Type 1 drivers translate calls to JDBC methods into calls to Microsoft Open Database Connectivity (ODBC) functions. Bridge drivers allow JDBC applications immediate access to database connectivity provided by the existing array of ODBC drivers.

Both the JavaSoft JDK and Microsoft Java SDK include JDBC-ODBC bridge drivers.

ODBC architecture requires that the ODBC driver manager and (typically) the ODBC drivers themselves be loaded on each client system. The requirement for software resident on client systems means that JDBC-ODBC bridge drivers will not work with Java applets run from an Internet browser. Browsers do not allow applets to run another program on the client to which they are downloaded. (In general, JDBC-ODBC bridge drivers will not work in environments that restrict Java applications from reading and writing files or running other programs.)

JDBC-ODBC bridge drivers are still useful in corporate networks, or for use by application server code written in Java in a 3-tier architecture. In such an environment, the application server has intermediary software, such as Blue Lobster's Aptivity, that

receives requests from browsers and other Internet applications. The intermediary software in turn calls the JDBC driver manager when it receives a database request.

1.3.2 Native-Method Drivers

Type 2 drivers contain Java code that calls "native" C or C++ methods already implemented by database vendors.

Like an ODBC driver, a native-method driver must be installed on each client or server that uses it, and thus has the same limitations as the JDBC-ODBC bridge drivers. A typical use of native-method drivers is on application servers.

1.3.3 Network-Protocol All-Java Drivers

Type 3 drivers are completely written in Java. They translate JDBC calls into a database-independent network protocol which is in turn translated to a DBMS protocol by middleware on a network server.

This type of driver can thus connect many Java clients to many different databases. The specific protocol used depends on the vendor.

Type 3 drivers are the most flexible since they do not require any driver software resident on client systems and can allow a single driver to provide access to multiple databases.

1.3.4 Native-Protocol All-Java Drivers (Dharma SDK JDBC Driver)

Type 4 drivers are also written completely in Java, but do not rely on middleware. They convert JDBC calls directly into the network protocol used by a particular database. This approach allows a direct call from the client system to the database server. Also, since there is no client-resident software, it also is practical for Internet applications.

Type 4 drivers provide the best performance.

The Dharma JDBC Driver is a Type 4 driver. Sybase is another vendor that offers Type 4 drivers for JDBC access to many of its database products.

1.4 JDBC COMPARED TO ODBC

Generally speaking, JDBC is to Java what Microsoft's Open Database Connectivity (ODBC) interface is to the C language. Both JDBC and ODBC:

- Provide a vendor-independent API that allows the same application to connect to different vendors' databases and retrieve and update data using standard SQL statements.
- Adopt the architecture of imposing a driver manager between applications and vendor-supplied drivers that translate between the standard API and a vendor's proprietary implementation.
- Are based on the X/Open SQL call-level interface specification.

JDBC proponents cite these advantages of JDBC over ODBC:

- JDBC applications enjoy the platform-independence of Java, which lends itself to Internet applications. ODBC applications must, at a minimum, be recompiled to run on a different operating-system/hardware combination.
- JDBC does not require software on each client system, which also recommends it for Internet applications.
- JDBC is much simpler and easier to learn than ODBC.
- JDBC is not primarily targeted for PC application development, which makes for faster implementation outside the Windows environment.

Chapter 2

Basic JDBC Driver Operations

2.1 INTRODUCTION

This chapter describes how to set up and get started using the Dharma SDK JDBC Driver.

2.2 REQUIRED JAVA ENVIRONMENT

You must have a supported Java development or runtime environment on each system that uses the JDBC Driver. The Dharma SDK JDBC Driver requires the following development (or compatible runtime) environments:

- On UNIX: JavaSoft JDK™ Version 1.4 .1_02
- On Windows NT or Windows 2000: JavaSoft JDK™ Version 1.4 or higher

You must have one of these environments or their associated Java runtime environments to use the JDBC Driver. For details on obtaining this software, see the URL *http://java.sun.com/products/index.html*.

2.3 SETTING UP THE JDBC DRIVER: WEB SERVER

In a Web server environment, the JDBC driver and Java applets that use it reside on a host system. No additional driver software is required on the client machine. Client applications must support a Java virtual machine compatible with JavaSoft's JDK Version 1.4 .1_02 (Internet browsers such as Netscape or Internet Explorer meet this requirement.)

Client applications invoke a JDBC applet through a Web page on the server. The browser downloads both the applet and the JDBC Driver from the server (usually in compressed format) and runs the applet. The Java applet opens a database connection (see section "2.5 Connecting to a Database" on page 2-4) and accesses the database using the JDBC API.

A general JDBC security restriction is that applets can only open a database connection from the server from which they are downloaded. That means the host system must be running both the HTTP Web server and the Dharma server *dhdaemon* process. (See the *Dharma SDK UserGuide and Release Notes* for details of managing the *dhdaemon* process).

To set up the JDBC Driver for an applet on a Web server, complete these steps:

- Copy compiled class files for the the applet and Java Archive(.jar) file of the JDBC driver to a directory accessible to the Web page that will invoke the applet.
- Compress all the applet class files into a single .jar file.

- Create the Web page that will invoke the applet.

2.3.1 Copying JDBC Driver and Applet Class Files

On both Windows NT/2000 and UNIX, the jar file for the JDBC Driver is installed in the directory mentioned during installation. Copy the jar file from that directory to a directory accessible to the Web page. Do the same for the applet's class file.

For example:

```
systpe@isis% cd $webroot
/vol6/webroot
systpe@isis% mkdir test
systpe@isis% cd test
/vol6/webroot/test
systpe@isis% cp -i $TPEROOT/DharmaDriver.jar .
systpe@isis% cp -i /applet_test/DhJDBCApplet.class .
```

2.3.2 Compressing Class Files Into Java Archive Files

This step is optional but recommended. JAR files greatly reduce the number of connections a browser must make to the Web server to download required classes. For example, compress class files into JAR file DhJDBCTest.jar file as following:

```
systpe@isis% pwd
/vol6/webroot/test
systpe@isis% jar -cvf DhJDBCTest.jar *.class
adding: ClRqTypes.class (in=1287) (out=736) (deflated 42%)
adding: CntlIface.class (in=696) (out=375) (deflated 46%)
adding: CntlIfaceCS.class (in=2143) (out=1116) (deflated 47%)
adding: CntlIfaceSS.class (in=2133) (out=1056) (deflated 50%)
.
.
.
systpe@isis% ls -al *.jar
-rw-r--r--   1 systpe   staff     132534 Sep  2 16:58 DhJD-
BCTest.jar
```

2.3.3 Creating a Web Page That Invokes the Applet

At a minimum, the page must include the APPLET tag that invokes the applet.

For example, the following page includes little else but the APPLET tag. The example's APPLET tag specifies the *DhJDBCApplet.class* and *DhJDBCTest.jar* files from the preceding sections, as well as class-name and connection parameters to pass to the applet.

```
systpe@isis% pwd
/vol6/webroot/test
```

```
systpe@isis% more jtest.htm
<html>
        <head>
                <title>Test</title>
        </head>
        <body>
        <p>
        Here, in all its glory, is the DhJDBCApplet test applet!
                <center>
                       <applet code="DhJDBCApplet.class"
                        archive="DhJDBCTest.jar" width=500 height=400>
                        <param name=Driver value="dharma.jdbc.DharmaDriver">
                        <param name=URL value="jdbc:dharma:T:isis:jdbcdb">
                        <param name=User value="systpe">
                        <param name=Password value="dummy">
                        </applet>
                </center>
        </body>
</html>
```

2.4 SETTING UP THE JDBC DRIVER: APPLICATION SERVER

In an application server environment, the system on which the JDBC application runs also has the JDBC driver installed. This configuration provides good performance when users are on the same system or can execute the JDBC application across a network.

To set up the JDBC Driver, you must have access to a system (UNIX or Windows) where the Dharma SDK libraries and executable files have been installed, as described in the *Dharma SDK Guide and Release Notes*.

2.4.1 Setting Environment Variables

Whether the JDBC Driver jar file resides locally or on network-served disks, you must set the CLASSPATH environment variable to point to the class files.

On both Windows and UNIX, the CLASSPATH environment variable must point to the directory containing the *DharmaDriver.jar* file.

Windows
On Windows , you must set the CLASSPATH environment variable:

```
C:\>set classpath


JREHOME=C:\JRE1.4
CLASSPATH=%TPEROOT%\DharmaDriver.jar;%JREHOME%\lib\rt.jar
```

(For the environment variables to persist across different processes, set them using the Window Control Panel's *System* utility, and set them as system variables.)

UNIX
Make sure the CLASSPATH variable includes the directory containing the *.class* files for the Dharma JDBC Driver.

For example:

```
% setenv CLASSPATH ".:$TPEROOT/DharmaDriver.jar:${JREHOME}/lib/
rt.jar"
```

2.5 CONNECTING TO A DATABASE

JDBC applications must perform two steps to connect to a database:

1. Load the JDBC driver
2. Connect to the driver

2.5.1 Load the JDBC Driver Using *Class.forName*

The *Class.forName* method takes as its argument the fully-qualified class name for the JDBC Driver. If it finds the class, the method loads and links the class, and returns the Class object representing the class.

The fully-qualified class name for the Dharma SDK JDBC Driver is *dharma.jdbc.DharmaDriver*. To load the JDBC Driver, use it as the argument to the *Class.forName* method:

```
// Load the driver
Class.forName ("dharma.jdbc.DharmaDriver");
```

2.5.2 Connect to the JDBC Driver Using *DriverManager.getConnection*

To connect to a Dharma SDK database through the JDBC Driver, an application specifies:

- A database connection string in the form of a JDBC URL
- User authentication detail (user name and password)

Applications specify this information as arguments to the *DriverManager.getConnection* method.

2.5.2.1 Java URL Connection String

DriverManager.getConnection requires at least one argument, a character string specifying a database connection URL. For the Dharma SDK JDBC Driver, the URL takes the following form:

`jdbc:dharma:T:host_name:db_name:port:`*optional connection information*

The URL string has the following components:

jdbc:dharma:T	An identifying protocol and subprotocol string for the Dharma JDBC Driver.
:host_name	Name of the server system where the database resides.
:db_name	Name of the database.

:port	The port number associated with the JDBC server on the host system. In most cases this component is optional. Java applets that are hosted on servers that do not use the default port number of 1990 must use this component to specify the correct port number. See the *Installation Guide and Release Notes* for details on setting the port number for the JDBC server.
:optional connection information	The *optional connection information* component of the URL is optional. If it is specified, the port information must also be specified in order for the string to be correctly parsed. The maximum length of this component is 200 characters. It may contain blank spaces but not the colon(**:**) character. The contents of this string are implementer dependent.

For example, the default URL in the sample application is *jdbc:dharma:T:isis:testdb*. When passed to *DriverManager.GetConnection*, this URL specifies that the Dharma JDBC Driver be used to connect to the database *testdb* on the server named *isis*.

2.5.2.2 User Authentication Detail

DriverManager.GetConnection accepts three variants of user authentication detail:

- User name and password passed as two character string arguments:

```
Connection con = DriverManager.getConnection (url, "fred"
"fredspasswd" );
```

- User name and password passed as a single *Properties* object:

```
Connection con = DriverManager.getConnection (url, prop );
```

Note that the JDBC Driver expects the keys of the *Properties* object to be named user and password when it processes the object. Application code must use those names when it populates the *Properties* object:

```
prop.put("user", userid);
prop.put("password", passwd);
```

- User name and password omitted. The JDBC Driver connects to the database with a blank username and null password:

```
Connection con = DriverManager.getConnection (url);
```

2.5.3 An Example Connection

The following example shows a code excerpt that illustrates loading the driver and connecting to the default server and database. The following example uses the form of *DriverManager.GetConnection* that takes authentication information as a single *Properties* object.

Example 2-1: Loading the JDBC Driver and Connecting to a Database

```
String url    = "jdbc:dharma:T:isis:testdb";
 String userid = "fred";
 String passwd = "fredspasswd";
```

```
// Load the driver
Class.forName ("dharma.jdbc.DharmaDriver");

// Attempt to connect to a driver.  Each one
// of the registered drivers will be loaded until
// one is found that can process this URL.
java.util.Properties  prop = new java.util.Properties();
prop.put("user", userid);
prop.put("password", passwd);

Connection con = DriverManager.getConnection (url, prop);
```

2.5.4 Connection Pooling Support in JDBC

The IBM's Websphere 4.0 Application server uses the JDBC APIs DataSource, PooledConnection and ConnectionPoolDataSource to connect to any data source. These APIs are part of the javax.sql package. It is mandatory to support them when working with IBM's Websphere 4.0 middleware.

2.5.4.1 javax.sql Package

The javax.sql package provides the APIs for server side data source access and processing. This is included in the Java 2 SDK version 1.4, Standard Edition.

The javax.sql package provides the following:

- The DataSource interface as an alternative to the DriverManager for establishing a connection with a data source
- Connection pooling

2.5.4.2 Using A DataSource Object To Make A Connection

The javax.sql package provides the preferred way to make a connection with a data source. The DriverManager class, the original mechanism, is still valid, and code using it will continue to run. However, the newer DataSource mechanism is preferred because it offers many advantages over the DriverManager mechanism.

The following are the main advantages of using a DataSource object to make a connection:

- Applications do not need to hard code a driver class.
- Changes can be made to a data source's properties, which means that it is not necessary to make changes in application code when something about the data source or driver changes.
- Connection pooling and distributed transactions are available through a DataSource object that is implemented to work with the middle-tier infrastructure. Connections made through the DriverManager do not have connection pooling or distributed transaction capabilities.

A particular DataSource object represents a particular physical data source, and each connection the DataSource object creates is a connection to that physical data source.

A DataSource object can be implemented to work with the middle tier infrastructure so that the connections it produces will be pooled for reuse. An application that uses such a DataSource implementation will automatically get a connection that participates in connection pooling. A DataSource object can also be implemented to work with the middle tier infrastructure so that the connections it produces can be used for distributed transactions without any special coding.

2.5.4.3 Connection Pooling

Connections made via a DataSource object that is implemented to work with a middle tier connection pool manager will participate in connection pooling. This can improve performance dramatically because creating new connections is very expensive. Connection pooling allows a connection to be used and reused, thus reducing the number of new connections that need to be created.

Connection pooling is totally transparent. It is done automatically in the middle tier of a J2EE Environment, and hence from an application's viewpoint, no change in code is required. An application simply uses the DataSource.getConnection method to get the pooled connection and uses it the same way it uses any Connection object.

2.5.4.4 Implementation in Dharma JDBC Driver

In order to support working with IBM's Websphere 4.0 Application server, the following interfaces from the package javax.sql have been implemented.

- javax.sql.DataSource

This is a factory for connections to the physical data source that this DataSource object represents. An object that implements the DataSource interface will typically be registered with a naming service based on the Java Naming and Directory (JNDI) API.

- javax.sql.PooledConnection interfaces

An object that provides hooks for connection pool management. A PooledConnection object represents a physical connection to a data source. The connection can be recycled rather than being closed when an application is finished with it, thus reducing the number of connections that need to be made.

- javax.sql.ConnectionPoolDataSource

This is a factory for PooledConnection objects. An object that implements this interface will typically be registered with a naming service that is based on the Java Naming and Directory Interface (JNDI).

A new directory, TPEROOT/src/jdbcx, has been introduced. This directory has the following source files:

- DharmaDataSource.java
- DharmaConnectionPoolDataSource.java
- DharmaPooledConnection.java
- DharmaObjectFactory.java

2.6 MANAGING TRANSACTIONS EXPLICITLY TO IMPROVE PERFORMANCE

By default, new connections in JDBC applications are in autocommit mode.

In autocommit mode every SQL statement executes in its own transaction:

- After successful completion, the JDBC Driver automatically commits the transaction.
- If the statement execution fails, the JDBC Driver automatically rolls back the transaction.

Note: In autocommit mode, the JDBC Driver does not issue a commit after SELECT and CALL statements. The driver assumes these statements generate result sets and relies on the application to explicitly commit or roll back the transaction after it processes any result set and closes the statement.

You can change the transaction mode to manual commit by calling the *Connection.setAutoCommit* method. In manual commit mode, applications must commit a transaction by using the *Connection.commit* method. Similarly, applications must explicitly roll back a transaction by invoking the *Connection.rollback* method.

You will improve the performance of your programs by setting autocommit to false after creating a Connection object with the *Connection.setAutoCommit* method:

```
      Connection con = DriverManager.getConnection ( url, prop);
.
.
.
     con.setAutoCommit(false);
```

Chapter 3

JDBC Conformance Notes

The Dharma SDK JDBC Driver is JDBC 3.0 compliant and provides access to Dharma SDK environments from applications and application servers that support JDBC 3.0.

3.1 SUPPORTED DATA TYPES

The Dharma JDBC Driver supports standard JDBC mapping of JDBC types corresponding Java types.

In the JDBC methods *CallableStatement.getXXX* and *PreparedStatement.setXXX* methods, *XXX* is a Java type:

- For *setXXX* methods, the driver converts the Java type to the JDBC type shown in the following table before sending it to the database.
- For *getXXX* methods, the driver converts the JDBC type returned by the database to the Java type shown in Table 3–2 before returning it to the *getXXX method*.

Table 3-1: Mapping Between Java and JDBC Data Types

Java Type	JDBC type	Java Type	JDBC type
String	VARCHAR or LONGVARCHAR	float	REAL
java.math.BigDecimal	NUMERIC	double	DOUBLE
boolean	BIT	byte[]	VARBINARY or LONG-VARBINARY
byte	TINYINT	java.sql.Date	DATE
short	SMALLINT	java.sql.Time	TIME
int	INTEGER	java.sql.Timestamp	TIMESTAMP
long	BIGINT		

Table 3-2: Mapping Between JDBC and Java Data Types

JDBC type	Java type	JDBC type	Java type
CHAR	String	REAL	float
VARCHAR	String	FLOAT	double

Table 3-2: Mapping Between JDBC and Java Data Types

JDBC type	Java type	JDBC type	Java type
LONGVAR-CHAR	String	DOUBLE	double
NUMERIC	java.math.BigDecimal	BINARY	byte[]
DECIMAL	java.math.BigDecimal	VARBINARY	byte[]
BIT	boolean	LONGVARBINARY	byte[]
TINYINT	byte	DATE	java.sql.Date
SMALLINT	short	TIME	java.sql.Time
INTEGER	int	TIMESTAMP	java.sql.Timestamp
BIGINT	long		

3.2 RETURN VALUES FOR *DATABASEMETADATA* METHODS

Applications call methods of the *DatabaseMetaData* class to retrieve details about the JDBC support provided by a specific driver.

The following table lists each method of the *DatabaseMetadata* class and shows what the Dharma JDBC Driver returns when an applications calls the method. For details on the format and usage of each method, see the Java Platform Core API documentation.

The following example shows an excerpt from the sample program that illustrates calling methods of *DatabaseMetadata*.

Example 3-1: Getting Driver Information Through DatabaseMetadata Methods

```
     Connection con = DriverManager.getConnection ( url, prop);
.
.
.
            // Get the DatabaseMetaData object and display
            // some information about the connection

            DatabaseMetaData dma = con.getMetaData ();

            o.println("\nConnected to " + dma.getURL());
            o.println("Driver       " +
               dma.getDriverName());
            o.println("Version      " +
               dma.getDriverVersion());
```

Many of the methods return lists of information as an object of type *ResultSet*. Use the normal *ResultSet* methods such as *getString* and *getInt* to retrieve the data from the result sets.

Table 3-3: Return Values for *DatabaseMetaData* Methods

Method	Description	Returns
allProceduresAreCallable()	Can all the procedures returned by getProcedures be called by the current user?	True
allTablesAreSelectable()	Can all the tables returned by getTable be SELECTed by the current user?	False
dataDefinitionCausesTransactionCommit()	Does a data definition statement within a transaction force the transaction to commit?	False
dataDefinitionIgnoredInTransactions ()	Is a data definition statement within a transaction ignored?	False
doesMaxRowSizeIncludeBlobs()	Did getMaxRowSize() include LONGVARCHAR and LONGVARBINARY blobs?	False
getBestRowIdentifier(String, String, String, int, boolean)	Get a description of a table's optimal set of columns that uniquely identifies a row.	(result set)
getCatalogs()	Get the catalog names available in this database.	"Driver not capable"
getCatalogSeparator()	What's the separator between catalog and table name?	"" (blank)
getCatalogTerm()	What's the database vendor's preferred term for "catalog"?	"" (blank)
getColumnPrivileges(String, String, String, String)	Get a description of the access rights for a table's columns.	(result set)
getColumns(String, String, String, String)	Get a description of table columns available in a catalog.	(result set)
getCrossReference(String, String, String, String, String, String)	Get a description of the foreign key columns in the foreign key table that reference the primary key columns of the primary key table (describe how one table imports another's key.) This should normally return a single foreign key/primary key pair (most tables only import a foreign key from a table once.) They are ordered by FKTABLE_CAT, FKTABLE_SCHEM, FKTABLE_NAME, and KEY_SEQ.	(result set)

Table 3-3: Return Values for *DatabaseMetaData* Methods

Method	Description	Returns
getDatabaseProductName()	What's the name of this database product?	"Dharma JDBC SDK"
getDatabaseProductVersion()	What's the version of this database product?	"09.00.0000"
getDefaultTransactionIsolation()	What's the database's default transaction isolation level? The values are defined in java.sql.Connection.	TRANSACTION_SERIALIZABLE
getDriverMajorVersion()	What's this JDBC driver's major version number?	9
getDriverMinorVersion()	What's this JDBC driver's minor version number?	0
getDriverName()	What's the name of this JDBC driver?	"dharma.jdbc.DharmaDriver"
getDriverVersion()	What's the version of this JDBC driver?	"09.00.0000"
getExportedKeys(String, String, String)	Get a description of the foreign key columns that reference a table's primary key columns (the foreign keys exported by a table).	(result set)
getExtraNameCharacters()	Get all the "extra" characters that can be used in unquoted identifier names (those beyond a-z, A-Z, 0-9 and _).	null
getIdentifierQuoteString ()	What's the string used to quote SQL identifiers? This returns a space " " if identifier quoting isn't supported.	"""
getImportedKeys(String, String, String)	Get a description of the primary key columns that are referenced by a table's foreign key columns (the primary keys imported by a table).	(result set)
getIndexInfo(String, String, String, boolean, boolean)	Get a description of a table's indices and statistics.	(result set)
getMaxBinaryLiteralLength()	How many hex characters can you have in an inline binary literal?	2000
getMaxCatalogNameLength()	What's the maximum length of a catalog name?	32
getMaxCharLiteralLength()	What's the max length for a character literal?	2000
getMaxColumnNameLength()	What's the limit on column name length?	32

Table 3-3: Return Values for *DatabaseMetaData* Methods

Method	Description	Returns
getMaxColumnsInGroupBy()	What's the maximum number of columns in a "GROUP BY" clause?	0 (no limit)
getMaxColumnsInIndex()	What's the maximum number of columns allowed in an index?	100
getMaxColumnsInOrderBy()	What's the maximum number of columns in an "ORDER BY" clause?	0 (no limit)
getMaxColumnsInSelect()	What's the maximum number of columns in a "SELECT" list?	0 (no limit)
getMaxColumnsInTable()	What's the maximum number of columns in a table?	500
getMaxConnections()	How many active connections can we have at a time to this database?	10
getMaxCursorNameLength()	What's the maximum cursor name length?	32
getMaxIndexLength()	What's the maximum length of an index (in bytes)?	0 (no limit)
getMaxProcedureNameLength()	What's the maximum length of a procedure name?	32
getMaxRowSize()	What's the maximum length of a single row?	0 (no limit)
getMaxSchemaNameLength()	What's the maximum length allowed for a schema name?	32
getMaxStatementLength()	What's the maximum length of a SQL statement?	35000
getMaxStatements()	How many active statements can we have open at one time to this database?	0 (no limit)
getMaxTableNameLength()	What's the maximum length of a table name?	32
getMaxTablesInSelect()	What's the maximum number of tables in a SELECT?	250
getMaxUserNameLength()	What's the maximum length of a user name?	32

Table 3-3: Return Values for *DatabaseMetaData* Methods

Method	Description	Returns
getNumericFunctions()	Get a comma separated list of math functions.	ABS, ACOS, ASIN, ATAN, ATAN2, CEILING, COS, COT, DEGREES, EXP, FLOOR, LOG, LOG10, MOD, PI, POWER, RADIANS, RAND, ROUND, SIGN, SIN, SQRT, TAN, TRUNCATE
getPrimaryKeys(String, String, String)	Get a description of a table's primary key columns.	(result set)
getProcedureColumns(String, String, String, String)	Get a description of a catalog's stored procedure parameters and result columns.	(result set)
getProcedures(String, String, String)	Get a description of stored procedures available in a catalog.	(result set)
getProcedureTerm()	What's the database vendor's preferred term for "procedure"?	"procedure"
getSchemas()	Get the schema names available in this database.	(result set)
getSchemaTerm()	What's the database vendor's preferred term for "schema"?	"Owner"
getSearchStringEscape()	This is the string that can be used to escape '_' or '%' in the string pattern style catalog search parameters.	"\"
getSQLKeywords()	Get a comma separated list of all a database's SQL keywords that are NOT also SQL92 keywords.	null

Table 3-3: Return Values for *DatabaseMetaData* Methods

Method	Description	Returns
getStringFunctions()	Get a comma separated list of string functions.	ASCII, LTRIM, CHAR, DIFFERENCE, INSERT, LCASE, LEFT, REPEAT, REPLACE, SOUNDEX, SPACE, SUBSTRING, UCASE, RTRIM, CONCAT, LENGTH, LOCATE
getSystemFunctions()	Get a comma separated list of system functions.	SQL_FN_SYS_USERNAME, SQL_FN_SYS_IFNULL, SQL_FN_SYS_DBNAME
getTablePrivileges(String, String, String)	Get a description of the access rights for each table available in a catalog.	(result set)
getTables(String, String, String, String [])	Get a description of tables available in a catalog.	(result set)
getTableTypes()	Get the table types available in this database.	SYNONYM, SYSTEM TABLE, TABLE, VIEW
getTimeDateFunctions()	Get a comma separated list of time and date functions.	CURDATE, CURTIME, DAYOFMONTH, DAYOFWEEK, DAYOFYEAR, MONTHNOW, QUARTER, WEEK, YEAR, HOUR, MINUTE, SECOND,
getTypeInfo()	Get a description of all the standard SQL types supported by this database.	(result set)
getURL()	What's the url for this database?	(the URL)
getUserName()	What's our user name as known to the database?	(the userid)

Table 3-3: Return Values for *DatabaseMetaData* Methods

Method	Description	Returns
getVersionColumns(String, String, String)	Get a description of a table's columns that are automatically updated when any value in a row is updated.	(result set)
isCatalogAtStart()	Does a catalog appear at the start of a qualified table name? (Otherwise it appears at the end)	True
isReadOnly()	Is the database in read-only mode?	False
nullPlusNonNullIsNull()	Are concatenations between NULL and non-NULL values NULL? A JDBC-Compliant driver always returns true.	True
nullsAreSortedAtEnd()	Are NULL values sorted at the end regardless of sort order?	False
nullsAreSortedAtStart()	Are NULL values sorted at the start regardless of sort order?	False
nullsAreSortedHigh()	Are NULL values sorted high?	False
nullsAreSortedLow()	Are NULL values sorted low?	True
storesLowerCaseIdentifiers()	Does the database treat mixed case unquoted SQL identifiers as case insensitive and store them in lower case?	True or False Depends on the identifier case specified during the creation of database.
storesLowerCaseQuotedIdentifiers()	Does the database treat mixed case quoted SQL identifiers as case insensitive and store them in lower case?	True or False Depends on the quoted identifier case specified during the creation of database.
storesMixedCaseIdentifiers()	Does the database treat mixed case unquoted SQL identifiers as case insensitive and store them in mixed case?	False
storesMixedCaseQuotedIdentifiers()	Does the database treat mixed case quoted SQL identifiers as case insensitive and store them in mixed case?	False
storesUpperCaseIdentifiers()	Does the database treat mixed case unquoted SQL identifiers as case insensitive and store them in upper case?	True or False Depends on the identifier case specified during the creation of database.
storesUpperCaseQuotedIdentifiers()	Does the database treat mixed case quoted SQL identifiers as case insensitive and store them in upper case?	False

Table 3-3: Return Values for *DatabaseMetaData* Methods

Method	Description	Returns
supportsAlterTableWithAddColumn()	Is "ALTER TABLE" with add column supported?	True
supportsAlterTableWithDropColumn()	Is "ALTER TABLE" with drop column supported?	True
supportsANSI92EntryLevelSQL()	Is the ANSI92 entry level SQL grammar supported? All JDBC-Compliant drivers must return true.	True
supportsANSI92FullSQL()	Is the ANSI92 full SQL grammar supported?	False
supportsANSI92IntermediateSQL()	Is the ANSI92 intermediate SQL grammar supported?	False
supportsCatalogsInDataManipulation()	Can a catalog name be used in a data manipulation statement?	False
supportsCatalogsInIndexDefinitions()	Can a catalog name be used in an index definition statement?	False
supportsCatalogsInPrivilegeDefinitions()	Can a catalog name be used in a privilege definition statement?	False
supportsCatalogsInProcedureCalls()	Can a catalog name be used in a procedure call statement?	False
supportsCatalogsInTableDefinitions()	Can a catalog name be used in a table definition statement?	False
supportsColumnAliasing()	Is column aliasing supported? If so, the SQL AS clause can be used to provide names for computed columns or to provide alias names for columns as required.	True
supportsConvert()	Is the CONVERT function between SQL types supported?	True
supportsConvert(int, int)	Is CONVERT between the given SQL types supported?	True or False Depends on the types being converted.
supportsCoreSQLGrammar()	Is the ODBC Core SQL grammar supported?	True
supportsCorrelatedSubqueries()	Are correlated subqueries supported? A JDBC-Compliant driver always returns true.	True
supportsDataDefinitionAndDataManipulationTransactions ()	Are both data definition and data manipulation statements within a transaction supported?	True
supportsDataManipulationTransactionsOnly()	Are only data manipulation statements within a transaction supported?	False

Table 3-3: Return Values for *DatabaseMetaData* Methods

Method	Description	Returns
supportsDifferentTableCorrelationNames()	If table correlation names are supported, are they restricted to be different from the names of the tables?	False
supportsExpressionsInOrderBy()	Are expressions in "ORDER BY" lists supported?	True
supportsExtendedSQLGrammar()	Is the ODBC Extended SQL grammar supported?	True
supportsFullOuterJoins()	Are full nested outer joins supported?	False
supportsGroupBy()	Is some form of "GROUP BY" clause supported?	True
supportsGroupByBeyondSelect()	Can a "GROUP BY" clause add columns not in the SELECT provided it specifies all the columns in the SELECT?	True
supportsGroupByUnrelated()	Can a "GROUP BY" clause use columns not in the SELECT?	False
supportsIntegrityEnhancementFacility()	Is the SQL Integrity Enhancement Facility supported?	True
supportsLikeEscapeClause()	Is the escape character in "LIKE" clauses supported? A JDBC-Compliant driver always returns true.	True
supportsLimitedOuterJoins()	Is there limited support for outer joins? (This will be true if supportFullOuterJoins is true.)	True
supportsMinimumSQLGrammar()	Is the ODBC Minimum SQL grammar supported? All JDBC-Compliant drivers must return true.	True
supportsMixedCaseIdentifiers()	Does the database treat mixed case unquoted SQL identifiers as case sensitive and as a result store them in mixed case? A JDBC-Compliant driver will always return false.	False
supportsMixedCaseQuotedIdentifiers()	Does the database treat mixed case quoted SQL identifiers as case sensitive and as a result store them in mixed case? A JDBC-Compliant driver will always return true.	True
supportsMultipleResultSets()	Are multiple ResultSets from a single execute supported?	False
supportsMultipleTransactions ()	Can we have multiple transactions open at once (on different connections)?	True

Table 3-3: Return Values for *DatabaseMetaData* Methods

Method	Description	Returns
supportsNonNullableColumns()	Can columns be defined as non-nullable? A JDBC-Compliant driver always returns true.	True
supportsOpenCursorsAcross-Commit()	Can cursors remain open across commits?	False
supportsOpenCursorsAcross-Rollback()	Can cursors remain open across rollbacks?	False
supportsOpenStatement-sAcrossCommit()	Can statements remain open across commits?	True
supportsOpenStatement-sAcrossRollback()	Can statements remain open across rollbacks?	True
supportsOrderByUnrelated()	Can an "ORDER BY" clause use columns not in the SELECT?	True
supportsOuterJoins()	Is some form of outer join supported?	True
supportsPositionedDelete()	Is positioned DELETE supported?	True
supportsPositionedUpdate()	Is positioned UPDATE supported?	True
supportsSchemasInDataManipulation()	Can a schema name be used in a data manipulation statement?	True
supportsSchemasInIndexDefinitions()	Can a schema name be used in an index definition statement?	True
supportsSchemasInPrivilegeDefinitions()	Can a schema name be used in a privilege definition statement?	True
supportsSchemasInProcedure-Calls()	Can a schema name be used in a procedure call statement?	True
supportsSchemasInTableDefinitions()	Can a schema name be used in a table definition statement?	True
supportsSelectForUpdate()	Is SELECT for UPDATE supported?	True
supportsStoredProcedures()	Are stored procedure calls using the stored procedure escape syntax supported?	True
supportsSubqueriesInComparisons()	Are subqueries in comparison expressions supported? A JDBC-Compliant driver always returns true.	True
supportsSubqueriesInExists()	Are subqueries in 'exists' expressions supported? A JDBC-Compliant driver always returns true.	True
supportsSubqueriesInIns()	Are subqueries in 'in' statements supported? A JDBC-Compliant driver always returns true.	True

Table 3-3: Return Values for *DatabaseMetaData* Methods

Method	Description	Returns
supportsSubqueriesInQuantifieds()	Are subqueries in quantified expressions supported? A JDBC-Compliant driver always returns true.	True
supportsTableCorrelationNames()	Are table correlation names supported? A JDBC-Compliant driver always returns true.	True
supportsTransactionIsolationLevel(int)	Does the database support the given transaction isolation level?	True
supportsTransactions ()	Are transactions supported? If not, commit is a no-op and the isolation level is TRANSACTION_NONE.	True
supportsUnion()	Is SQL UNION supported?	True
supportsUnionAll()	Is SQL UNION ALL supported?	True
usesLocalFilePerTable()	Does the database use a file for each table?	False
usesLocalFiles()	Does the database store tables in a local file?	False
JDBC2.0		
deletesAreDetected(int)	Indicates whether or not a visible row delete can be detected by calling ResultSet.rowDeleted().	False
getConnection()	Retrieves the connection that produced this metadata object.	The connection that produced this metadata object
getUDTs(String, String, String, int[])	Gets a description of the user-defined types defined in a particular schema.	Empty ResultSet object
insertsAreDetected(int)	Indicates whether or not a visible row insert can be detected by calling ResultSet.rowInserted().	False
othersDeletesAreVisible(int)	Indicates whether deletes made by others are visible.	False
othersInsertsAreVisible(int)	Indicates whether inserts made by others are visible.	False
othersUpdatesAreVisible(int)	Indicates whether updates made by others are visible.	False
ownDeletesAreVisible(int)	Indicates whether a result set's own deletes are visible.	False
ownInsertsAreVisible(int)	Indicates whether inserts made by others are visible.	False
ownUpdatesAreVisible(int)	Indicates whether a result set's own updates are visible.	False

Table 3-3: Return Values for *DatabaseMetaData* Methods

Method	Description	Returns
supportsBatchUpdates()	Indicates whether the driver supports batch updates.	True
supportsResultSetType(int)	Does the database support the given result set type?	True if result set type is FORWARD_ONLY
supportsResultSetConcurrency(int, int)	Does the database support the concurrency type in combination with the given result set type?	True if result set type is FORWARD_ONLY and if concurrency type is CONCUR_READ_ONLY
updatesAreDetected(int)	Indicates whether or not a visible row update can be detected by calling the method ResultSet.rowUpdated.	False
JDBC3.0		
supportsSavepoints()	Does the database support savepoints	False
supportsNamedParameters()	Does the database support named parameters to callable statements	False
supportsMultipleOpenResults()	Indicates whether it is possible to have multiple ResultSet objects returned from a Callable Statement object simultaneously	False
supportsGetGeneratedKeys()	Indicates whether auto-generated keys can be retrieved after a statement has been executed	False
getSuperTypes(String, String, String)	Gets a description of user-defined type hierarchies defined in a particular schema in this database	Empty ResultSet object
getSuperTables(String, String, String)	Gets a description of tables defined in a particular schema in this database	Empty ResultSet object
getAttributes(String, String, String, String)	Gets a description of the given attribute of the given type for a user-defined type that is available in the given schema and catalog	Empty ResultSet object
supportsResultSetHoldability(int)	Does the database support the given result set holdability	True if result set holdability is ResultSet.CLOSE_CURSORS_AT_COMMIT, otherwise False
getResultSetHoldability()	Gets the default holdability of this ResultSet object	Always returns ResultSet.CLOSE_CURSORS_AT_COMMIT
getDatabaseMajorVersion()	Gets the major version number of the underlying database	returns 9

Table 3-3: Return Values for *DatabaseMetaData* Methods

Method	Description	Returns
getDatabaseMinorVersion()	Gets the minor version number of the underlying database	returns 0
getJDBCMajorVersion()	Gets the major JDBC version number of this driver	returns 3
getJDBCMinorVersion()	Gets the minor JDBC version number of this driver	returns 0
getSQLStateType()	Indicates whether the SQLStates returned by SQLException.getSQLState is X/Open SQL CLI or SQL99	returns sqlState99
locatorsUpdateCopy()	Indicates whether updates made to LOB are made on a copy or directly to the LOB	“Driver does not support this” Exception
supportsStatementPooling()	Does the database support statement pooling	False

3.3 ERROR MESSAGES

The error messages generated by the driver, along with associated SQLSTATE and Dharma error code values, are documented in the *Dharma SDK SQL Reference Manual*.

Appendix A

Glossary

A.1 TERMS

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.

alias
A temporary name for a table or column specified in the FROM clause of an SQL query expression. Also called correlation name. Derived tables and search conditions that join a table with itself must specify an alias. Once a query specifies an alias, references to the table or column must use the alias and not the underlying table or column name.

applet
A special kind of Java program whose compiled class files a Java-enabled browser can download from the Internet and run.

ASCII
(American Standard Code for Information Interchange) A 7-bit character set that provides 128 character combinations.

bytecode
Machine-independent code generated by the Java compiler and executed by the Java interpreter.

candidate key
Another term for unique key.

cardinality
Number of rows in a result table.

Cartesian product
Also called cross-product. In a query expression, the result table generated when a FROM clause lists more than one table but specifies no join conditions. In such a case, the result table is formed by concatenating every row of every table with all other rows in all tables. Typically, Cartesian products are not useful and are slow to process.

client
Generally, in client/server systems, the part of the system that sends requests to servers and processes the results of those requests.

collation
The rules used to control how character strings in a character set compare with each other. Each character set specifies a collating sequence that defines relative values of each character for comparing, merging and sorting character strings. In addition, storage systems may define additional collations that SQL statements specify with the COLLATE clause in column definitions, column references, and character-string references.

column alias
An alias specified for a column. See alias.

constraint
Part of an SQL table definition that restricts the values that can be stored in a table. When you insert, delete, or update column values, the constraint checks the new values against the conditions specified by the constraint. If the value violates the constraint, it generates an error. Along with triggers, constraints enforce referential integrity by insuring that a value stored in the foreign key of a table must either be null or be equal to some value in the matching unique or primary key of another table.

correlation name
Another term for alias.

cross product
Another term for Cartesian product.

data dictionary
Another term for system catalog.

data source
See ODBC data source.

derived table
A virtual table specified as a query expression in the FROM clause of another query expression.

driver manager
See JDBC driver manager and ODBC driver manager.

field handle
In the storage interfaces, a handle that identifies storage for data stored in columns defined with the SQL LONG VARCHAR or LONG VARBINARY data type. Implementations create field handles when the SQL engine calls the tpl_hdl_t::tpl_insert routine. (This is in contrast to conventional data-type columns, for which the SQL engine passes actual values to the insert routine.) Similarly, for fetch routines, implementations return field handles instead of the actual long data values.

flat-file storage system / storage manager
A storage system and storage manager supplied with Dharma SDK . It provides an example of an implementation of the storage interfaces. In addition, implementations can use it as a simple repository that can be used for storing system catalog tables. Implementations that do not support table creation, for instance, can use the flat-file storage system for system catalog tables.

foreign key
A column or columns in a table whose values must either be null or equal to some value in a corresponding column (called the primary key) in another table. Use the REFERENCES clause in the SQL CREATE TABLE statement to create foreign keys.

form of use
The storage format for characters in a character set. Some character sets, such as ASCII, require one byte (octet) for each character. Others, such as Unicode, use two bytes, and are called multi-octet character sets.

handle
In the storage interfaces, a temporary identifier for database objects. Storage managers generate handles when the SQL engine calls routines to open tables, indexes, table scans, and index scans, or to access long data-type columns. The SQL engine uses the handle on subsequent calls to scan, fetch, insert, and update operations. More generally, a handle is a memory pointer associated with a temporary object which does not last across a user level SQL session. Compare with identifier.

identifier
In the storage interfaces, a persistent object that identifies database elements such as tables and indexes. Storage managers generate identifiers when the database element is created. The identifier is stored in the appropriate system table along with other information that describes the object. Types of identifiers include table, index, tuple, tuples, procedure, and trigger. Compare with handle.

index handle
In the storage interfaces, a handle that identifies an index open for updating. Implementations generate index handles when the SQL engine calls rss_hdl_t::ix_hdl_ctor.

Java snippet
See snippet.

JDBC
Java Database Connectivity: a part of the Java language that allows applications to embed standard SQL statements and access any database that implements a JDBC driver.

JDBC driver
Database-specific software that receives calls from the JDBC driver manager, translates them into a form that the database can process, and returns data to the application.

JDBC driver manager
A Java class that implements methods to route calls from a JDBC application to the appropriate JDBC driver for a particular JDBC URL.

join
A relational operation that combines data from two tables.

input parameter
In a stored procedure specification, an argument that an application must pass when it calls the stored procedure. In an SQL statement, a parameter marker in the statement string that acts as a placeholder for a value that will be substituted when the statement executes.

interface
In Java, a definition of a set of methods that one or more objects will implement. Interfaces declare only methods and constants, not variables. Interfaces provide multiple-inheritance capabilities.

main-memory storage system / storage manager
A storage system and storage manager supplied with Dharma SDK . It provides a mechanism for implementations to store data in memory instead of on disk. By using the main-memory storage system for volatile data such as temporary sort tables and dynamic indexes, implementations can improve performance of many queries, such as joins.

manager
A main component of the SQL engine. In particular, the term storage manager refers to a completed implementation of the storage interfaces that provides access to an underlying storage system.. Besides one or more storage managers, the SQL engine includes several managers, including the SQL statement manager, parser, and optimizer.

metadata
Data that details the structure of tables and indexes in the proprietary storage system. The SQL engine stores metadata in the system catalog.

octet
A group of 8 bits. Synonymous with byte, and often used in descriptions of character-set encoding format.

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. Dharma 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.

optimizer
Within the SQL engine, the manager that analyzes costs and statistics associated with the statement and converts the relational algebra tree to the most efficient form for execution. The optimizer stores the trees for later use.

output parameter
In a stored procedure specification, an argument in which the stored procedure returns a value after it executes.

package
A group of related Java classes and interfaces, like a class library in C++. The Java development environment includes many packages of classes that procedures can import. The Java runtime system automatically imports the java.lang package. Stored procedures must explicitly import other classes by specifying them in the IMPORT clause of a CREATE PROCEDURE statement.

parameter marker
A question mark (?) in a procedure call or SQL statement string that acts as a placeholder for an input or output parameter supplied at runtime when the procedure executes. The CALL statement (or corresponding ODBC or JDBC escape clause) use parameter markers to pass parameters to stored procedures, and the SQLIStatement, SQLPStatement, and SQLCursor objects use them within procedures.

postfix notation
Notation in which the numbers precede the operation. For example, 2 + 2 is expressed as 2 2 +, and 10 - 3 * 4 would be 10 3 4 * -. If a storage manager supports processing of expressions, the SQL engine passes them to the storage manager using postfix notation.

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.

procedure body
In a stored procedure, the Java code between the BEGIN and END keywords of a CREATE PROCEDURE statement.

procedure result set
In a stored procedure, a set of data rows returned to the calling application. The number and data types of columns in the procedure result set are specified in the RESULT clause of the CREATE PROCEDURE statement. The procedure can transfer data from an SQL result set to the procedure result set or it can store data generated internally. A stored procedure can have only one procedure result set.

procedure specification
In a CREATE PROCEDURE statement, the clauses preceding the procedure body that specify the procedure name, any input and output parameters, any result set columns, and any Java packages to import.

procedure variable
A Java variable declared within the body of a stored procedure, as compared to a procedure input parameter or output parameter, which are declared outside the procedure body and are visible to the application that calls the stored procedure.

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, and can be used in some expressions and search conditions.

referential integrity
The condition where the value stored in a database table's foreign key must either be null or be equal to some value in another table's the matching unique or primary key. SQL provides two mechanisms to enforce referential integrity: constraints specified as part of CREATE TABLE statements prevent updates that violate referential integrity, and triggers specified in CREATE TRIGGER statements execute a stored procedure to enforce referential integrity.

repertoire
The set of characters allowed in a character set .

result set
In a stored procedure, either an SQL result set or a procedure result set.

More generally, another term for result table.

result table
A virtual table of values derived from columns and rows of one or more tables that meet conditions specified by an SQL query expression.

row identifier
Another term for tuple identifier.

scan handle
In the storage interfaces, a handle that identifies an index or table open for scan operations. Implementations generate scan handles when the SQL engine calls rss_hdl_t::ix_scanhdl_ctor or rss_hdl_t::tpl_scan_hdl_ctor.

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.

snippet
In a stored procedure, the sequence of Java statements between the BEGIN and END keywords in the CREATE PROCEDURE (or CREATE TRIGGER) statement. The Java statements become a method in a class the SQL engine creates and submits to the Java compiler.

SQL diagnostics area
A data structure that contains information about the execution status (success, error or warning conditions) of the most recent SQL statement. The SQL-92 standard specified the diagnostics area as a standardized alternative to widely varying implementations of the SQLCA. Dharma SDK supports both the SQLCA and the SQL diagnostics area. The SQL GET DIAGNOSTICS statement returns information about the diagnostics area to an application, including the value of the SQLSTATE status parameter.

SQL engine
The core component of the Dharma SDK environment. The SQL engine receives requests from applications, processes them, and returns results. The SQL engine calls the storage interfaces to convey requests to an underlying storage system.

SQLCA
SQL Communications area: A data structure that contains information about the execution status (success, error or warning conditions) of the most recent SQL statement. The SQLCA includes an SQLCODE field. The SQLCA provides the same information as the SQL diagnostics area, but is not compliant with the SQL-92 standard. Dharma SDK supports both the SQLCA and the SQL diagnostics area.

SQLCODE
An integer status parameter whose value indicates the condition status returned by the most recent SQL statement. An SQLCODE value of zero means success, a positive value means warning, and a negative value means an error status. SQLCODE is superseded by SQLSTATE in the SQL-92 standard. Applications declare either SQLSTATE or SQLCODE, or both. SQL returns the status to SQLSTATE or SQLCODE after execution of each SQL statement.

SQL result set
In a stored procedure, the set of data rows generated by an SQL statement (SELECT and, in some cases, CALL).

SQLSTATE
A 5-character status parameter whose value indicates the condition status returned by the most recent SQL statement. SQLSTATE is specified by the SQL-92 standard as a replacement for the SQLCODE status parameter (which was part of SQL-89). SQLSTATE defines many more specific error conditions than SQLCODE, which allows applications to implement more portable error handling. Applications declare either SQLSTATE or SQLCODE, or both. SQL returns the status to SQLSTATE or SQLCODE after execution of each SQL statement.

storage environment
The combination of storage systems which have implemented the storage interfaces. One possible combination of storage systems in an implementation is the Dharma-supplied flat-file and main-memory storage system, with a proprietary database containing user data.

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. Once filled in for a particular storage system, the completed storage interfaces are called a storage manager.

storage manager
A completed implementation of Dharma SDK storage interfaces. A storage manager receives calls from the SQL engine through the storage interfaces and accesses the underlying proprietary storage system to retrieve and store data.

storage system
The proprietary database system that underlies a storage manger. Dharma SDK provides an SQL interface to a storage system through the SQL engine and its storage interfaces.

stored procedure
A snippet of Java source code embedded in an SQL CREATE PROCEDURE statement. The source code can use all standard Java features as well as use Dharma SDK -supplied Java classes for processing any number of SQL statements.

stub interfaces
Another term for storage interfaces. Also called simply stubs.

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.

dharma
The default owner name for all system tables in a Dharma SDK database. Users must qualify references to system tables as systpe.tablename.

table handle
In the storage interfaces, a handle that identifies a table open for non-scan operations. Implementations generate scan handles when the SQL engine calls rss_hdl_t::tpl_hdl_ctor.

table space
A mechanism 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.

tid
Another term for tuple identifier.

transaction
A group of operations whose changes can be made permanent or undone only as a unit. Once implementations add the ability to change data in the proprietary storage system, they must also implement transaction management to protect against data corruption.

trigger
A special type of stored procedure that helps insure referential integrity for a database. Like stored procedures, triggers also contain Java source code (embedded in a CREATE TRIGGER statement) and use Dharma SDK Java classes. However, triggers are automatically invoked ("fired") by certain SQL operations (an insert, update, or delete operation) on the trigger's target table.

trigger action time
The BEFORE or AFTER keywords in a CREATE TRIGGER statement. The trigger action time specifies whether the actions implemented by the trigger execute before or after the triggering INSERT, UPDATE, or DELETE statement.

trigger event
The statement that causes a trigger to execute. Trigger events can be SQL INSERT, UPDATE, or DELETE statements that affect the table for which a trigger is defined.

triggered action
The Java code within the BEGIN END clause of a CREATE TRIGGER statement. The code implements actions to be completed when a triggering statement specifies the target table.

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.

Unicode
A superset of the ASCII character set that uses two bytes for each character rather than ASCII's 7-bit representation. Able to handle 65,536 character combinations instead of ASCII's 128, Unicode includes alphabets for many of the world's languages. The first 128 codes of Unicode are identical to ASCII, with a second-byte value of zero.

unique key
A column or columns in a table whose value (or combination of values) must be unique. Use the UNIQUE clause of the SQL CREATE TABLE statement to create unique keys. Unique keys are also called candidate keys.

URL
In general, a Universal Resource Locator used to specify protocols and locations of items on the Internet. In JDBC, a database connection string in the form jdbc:subprotocol:subname. The Dharma JDBC Driver format for database URLs is jdbc:dharma:T:host_name:db_name.

utility class
A set of utility functions that a storage manager uses to assemble and disassemble data elements passed through the storage interfaces.

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.

virtual table
A table of values that is not physically stored in a database, but instead derived from columns and rows of other tables. SQL generates virtual tables in its processing of query expressions: the FROM, WHERE, GROUP BY and HAVING clauses each generate a virtual table based on their input.

virtual machine
The Java specification for a hardware-independent and portable language environment. Java language compilers generate code that can execute on a virtual machine. Implementations of the Java virtual machine for specific hardware and software platforms allow the same compiled code to execute without modification.

Index

U

V

W