MAKING ORACLE AND JDBC WORK FOR YOU

John King, King Training Resources

Introduction

Java's ability to use Oracle's database is provided by means of JDBC (Java DataBase Connectivity). JDBC was built upon the ODBC standard and allows Java programs to access the commonly used relational database products including Oracle. Standard JDBC provides a mostly vendor-neutral ability to query and update database tables and execute stored procedures. Typically, the driver and connection code is unique to a vendor's database and SQL variations are also possible. Oracle's JDBC drivers and JDBC-oriented packages provide a mechanism that is tailored for the Oracle database and its data. Oracle JDBC also allows programmers to tune the connection to the database in a way that is not possible using generic JDBC. Developers must decide between the need for portability and the need for maximum performance. Where portability between database products is a major concern the "generic" JDBC should be used. If performance is the most important issue, then, Oracle's JDBC classes should be used. This session will introduce the Java syntax necessary to query and update Oracle table data as well as to execute Oracle stored procedures and functions. In addition, many of the more-important Oracle extensions to standard JDBC are explored and explained.

Oracle and Java

Oracle9i and Oracle8i provide a Java Virtual Machine (JVM) built in to the database. Oracle was the first commercial database vendor to build a JVM into the database.

Oracle provides JDBC drivers to allow Java to use the database, these should probably be used rather than the drivers provided by Sun. Additionally, third-party drivers are now available for purchase that offer additional features and improved performance.

The use of JDBC requires the use of Java programming skills limiting its usefulness to those with Java backgrounds. For those developers with limited Java knowledge, the SQLJ option allows programmers to embed SQL statements (similar to Pro*C/C++) into Java programs directly. Oracle provides an SQLJ translator used for converting embedded SQL in Java to calls that behave in a manner similar to JDBC.

JDeveloper is an Integrated Development Environment (IDE) provided by Oracle that may be used to create Java programs using JDBC or SQLJ. JDeveloper was originally based upon Borland's JBuilder product, but, recent releases represent an almost complete rewrite.

Oracle Application Server (OAS) supports Java and JDBC fully and also includes support for JavaBeans and EJB.

Database Connectivity with JDBC

Sun provides JDBC as a tool to access SQL databases from Java. JDBC was originally based upon the ODBC standard successfully pioneered by Microsoft.

Sun provides JDBC drivers for accessing SQL databases directly, or, via a JDBC-ODBC bridge. The JDBC-ODBC bridge is simple, but inefficient for most large-scale applications. Oracle JDBC drivers support JDBC 1.22 fully and are specifically designed to take advantage of the Oracle database. Most of JDBC 2.0 is supported by Oracle JDBC in Oracle9i.

JDBC allows programmers to use SQL to: Connect to a database, Query a database, and Update a database.

JDBC programs are database vendor and platform independent (mostly) unless Oracle-specific features are used. This leaves the Oracle customer with the choice of greater portability or better performance.

JDBC Drivers

Oracle support three types of JDBC drivers: the Oracle Thin JDBC driver, Oracle OCI JDBC driver, and Oracle Server JDBC driver.

The Oracle Thin JDBC driver is the best choice for most applications and applets since it: Does not require Oracle client software, creates its own Oracle Net / Net 8 / Sql*Net connection, and is downloadable (about 900k).

The Oracle OCI JDBC driver is probably best where high performance is required, for instance, when Java code is web server-side in the form of Servlets, JSPs, and Java Data Beans. The OCI connection requires Oracle client installation and uses an existing Oracle Net / Net 8 / Sql*Net connection.

The Oracle Server JDBC driver (also known as the KPRB driver) is specifically designed for code located in the Oracle database server (not the web server). The Oracle Server JDBC driver is used by Java stored procedures and functions and for Enterprise Java Beans. The Oracle Server JDBC driver supports connections between PL/SQL and Java.

Which Driver for Me?

Application developers must choose the best driver for the job.

Use the Thin JDBC driver for: Applets, most applications, web servers without Oracle client software. The Thin JDBC driver is used for TCP/IP connections only.

Use the OCI JDBC driver for Applications requiring the best performance, code running on web servers (where Oracle licenses are available), and connections not using TCP/IP.

Use the Server JDBC driver for programs running in the database server (requires Oracle 8.1.5 or later).

What Database Version?

Various versions of Oracle's database support different levels of JDBC drivers. Thin JDBC drivers and OCI JDBC drivers work with: Oracle 9.0.x, Oracle 8.1.x, Oracle 8.0.x (no support for objects), and Oracle 7.x (no support for objects or LOBs)).

Oracle Server JDBC drivers became available with Oracle 8.1.5 but can access data in Oracle 8.1.4. Oracle Server drivers are not available in Oracle 7 (any release).

Oracle JDBC Features

Oracle's JDBC drivers add several features to the standard Java JDBC including: JDBC 1.22 compliance, support for object-relational data, support for LOB data, Oracle-specific performance features, all Oracle-supported character sets, and use of PL/SQL and Java stored procedures.

However, once a connection has been established, JDBC works the same regardless of driver being used unless Oracle-specific database objects are used.

JDBC Drivers

JDBC drivers for Oracle must be available for compilation and testing, they come in two files: *classes111.zip* (Java 1.1.x) and *classes12.zip* (Java 1.2.x). Two additional sets of drivers may be added for installations using Oracle NLS features: *nls_charset11.zip* (Java 1.1.x NLS characters) and *nsl_charset12.zip* (Java 1.2.x NLS characters). The Zip files are normally included with the database installation and may be found in the directory: <ORACLE_HOME>/jdbc/lib.

Make sure that ONLY ONE SET of these zip files is in your CLASSPATH (1.1.x or 1.2.x) or the JVM might be confused.

Using JDBC

To use JDBC in Java a series of steps is followed:

1. Import java.sql.* and other needed packages
2. Load/register the JDBC driver
3. Connect to the database using JDBC
4. Create a statement object
5. Execute SQL statements and process results
6. Close the result set (technically not required, but safest to avoid memory leak issues)
7. Close the statement
8. Close the connection (disconnect from database)

The next several sections describe the mechanisms used to access the database using JDBC.

Load/register JDBC driver

Prior to using JDBC Java must be informed which JDBC driver will be used. This is one of the two parts of JDBC that is most impacted by switching database vendors.

First the JDBC driver to be used must be made available to the JVM by including it in the CLASSPATH defined for the local environment. It is always a good idea to make sure that the most current driver is used (download from Oracle's web site if necessary).

Java may be notified which driver will by used by either loading or registering of the JDBC driver using one of the following syntax selections:

```
DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
```

or

```
Class.forName("oracle.jdbc.driver.OracleDriver");
```

Connect to the Database

Connections are the other major part of JDBC that is vendor-dependent and must be modified when porting from one database vendor to another. Furthermore, Oracle's connection string varies depending upon the driver in use.

An Oracle DBA's (DataBase Administrator's) help may be required to properly format the connect string and identify the appropriate settings.

Note: It is a bad idea to hard-code userid and password information in code as shown in these examples, be sure to prompt the current user for the necessary information

When finished using the database it is important to close the Connection using the Connection object's *close()* method. Code to close the connection is often placed in a finally block.

Connection Object

The code to connect to the database using the Thin client connection specifies the Host Computer Name, Port Used, and Oracle SID: jdbc:oracle:thin:@**hostcomputer**:**port**:**oraclesid**

```
String url = "jdbc:oracle:thin:@tecra:1521:tec817";
String uID = "scott";  // prompt user for this
String uPswd = "tiger";   // prompt user for this
Connection conn = DriverManager.getConnection url, uID,uPswd);
```

The code to connect to the database using the OCI client connection specifies only the Oracle SID: jdbc:oracle:oci8:@**oraclesid**

```
String url = "jdbc:oracle:oci8:@tec817";
String uID = "scott";     // prompt user for this
String uPswd = "tiger";   // prompt user for this
Connection conn = DriverManager.getConnection url,uId,uPswd);
```

Create a Statement Object

JDBC uses a Statement object to contain the SQL statement to be executed. For maximum portability use generic SQL and the standard Java Statement object. For maximum performance/flexibility, it might be necessary to use Oracle-specific SQL and the OracleStatement object.

When finished using the database it is important to close the Statement using the Statement object's *close()* method. Code to close the connection is often placed in a finally block.

To define a standard Java JDBC Statement object:

```
Statement stmt = conn.createStatement ();
```

To define Oracle-specific Statement objects:

```
OracleStatement stmt = conn.createStatement ();

OraclePreparedStatement stmt = conn.createStatement ();
```

Instantiation of a statement begins the preparation for SQL execution. No data is returned to the program yet.

Result Set

When executing a query, JDBC offers a Result Set object that looks and behaves in ways very reminiscent of a cursor in PL/SQL or Pro*C/C++.

A Result Set object is created and instantiated by executing an SQL query. Immediately after a query, the result set pointer points one record before the first row returned. This is very similar to opening a cursor (but not quite the same).

To process rows use the next method to get the first record and again to get subsequent records. Again, this is similar to the manner in which a cursor fetch is used.

It is not normally necessary to manually close the Result Set, it will be closed automatically when the Statement it is associated with is closed or re-executed using the ResultSet object's *close()* method. However, due to memory-leak issues, it may be safest to close the Result Set (it is certainly not incorrect close the Result Set).

Executing SQL Statements

To execute a query the SQL for the query is used as an input parameter/argument to the construction of a new ResultSet object instance. Once the ResultSet is instantiated, rows resulting from the query will be available to the program for processing.

```
String mySql = "SELECT ENAME,JOB,DEPTNO, "
```

```
          + " to_char(HIREDATE,'MM/DD/YYYY') HIREDATE "
          + " FROM EMP "
          + " WHERE DEPTNO = '" + myParm + "'"
          + " ORDER BY ENAME";
ResultSet rs = stmt.executeQuery(mySQL);
```

To execute an update or stored procedure call a ResultSet object is not necessary.

```
String mySQL = "UPDATE EMP SET SAL = SAL * 1.06 "
             + " WHERE ROWID = '" + saveRowId + "'";
int     nbrRows = stmt.executeUpdate(mySql);
```

ROWID

Queries that supply values that will be used for updates or deletes later should include the Oracle ROWID for each row returned. ROWID provides the fastest possible access to a given row when later updating or deleting takes place. Don't show the ROWID to the user, it can be pretty scary looking.

Process Result Sets

When the SQL executed is a query, the ResultSet is made available upon instantiation of the ResultSet object but no data has yet been returned for the program's use. Data is processed a row at time in the ResultSet using the *next()* method, this is frequently done using a Java while statement (next() will return false when an SQL error occurs or when there are no more rows):

```
while (rs.next()) { /** process SQL row **/ }
```

Once a row has been retrieved using the *next()* method column values are processed independently. Programs may retrieve values one column at a time by name (must be all upper-case letters for Oracle), or, by position in the SQL statement:

```
String myEmpno = rs.getString("EMPNO");

String myEmpno = rs.getString(1);
```

Result Set Processing

Java provides several other methods that might be useful for result set processing including:

afterLast	Moves cursor to the end of ResultSet
beforeFirst	Moves cursor to the front of ResultSet
clearWarnings	Clears all warnings on ResultSet
close	Releases ResultSet object's database and JDBC resources immediately
deleteRow	Deletes the current row
first	Moves cursor to first row in ResultSet
getWarnings	Returns the first warning reported
insertRow	Inserts a row into this ResultSet
last	Moves cursor to last row in ResultSet
next	Moves the cursor down one row
previous	Moves the cursor up one row

Result Set Testing

Java provides methods that may be used for testing ResultSet processing, each returns a Boolean value (true or false). The rowDeleted(), rowInserted(), and rowUpdated() methods may all be used to see if the associated result set processing statement succeeded or failed.

Unlike Pro*C/C++ programs null indicator variables are not used when processing data in Java. The wasNull() method must be tested for each column that might be null.

Result Set Values

The example earlier used the getString() method to retrieve a character-type column from the database. The appropriate method for each column's datatype must be used or an error will result. Some standard methods used to process column values by name or by position:

- getBlob
- getBoolean
- getByte
- getClob
- getDate
- getDouble
- getFloat
- getInt

OracleResultSet Extensions

When using standard Java result set processing, the getXXX methods often require that data be converted before it can be processed properly. This may cause a performance problem when millions of rows are involved. Oracle extensions are available that take into account Oracle specifics is programs use OracleResultSet objects instead of ResultSet objects:

- getArray
- getBfile
- getBlob
- getClob
- getNumber
- getOracleObject
- getRaw
- getRef
- getRowid
- getStruct

JDBC Cleanup

Failing to close database connections can cause problems for the database. Programmers must be sure to close connections manually or they will remain open. It is a good idea to put the Statement close() and Connection close() method calls in a finally block. Java will execute a finally block whether the associated try block ends normally or throws an exception. If desired, a program may include an additional statement, connection close since there is no negative impact from trying to close an already closed statement or connection.

```
finally {
    try {
      // Close result set
      rs.close();
      // Close statement
      stmt.close();
      // Close connection
      conn.close();
    }
```

Using Stored Procedures

Java programs may also use Oracle stored procedures or functions. Using stored procedures is a simply a matter of knowing the procedure/function name, parameters, and return type (for functions). CallableStatement objects or OracleCallableStatement objects are used instead of the normal Statement objects.

When using procedures or functions that require input parameters or arguments, the input parameters must be defined to JDBC prior to calling the stored code.

Input parameters are set using one of the following methods: setString, setArray, setAsciiStream, setBigDecimal, setBinaryStream, setBlob, setBoolean, setByte, setBytes, setCharacterStream, setClob, setDate, setDate, setDouble, setFloat, setInt, setLong, setNull, setObject, setRef, setShort, setString, setTime, setTimestamp, or setUnicodeStream.

In the event of output parameters or function returns the output parameters or function returns are defined using the RegisterOutParameter method.

To execute a stored procedure: Set methods are used to define input paraemeters, the Statement's executeUpdate() method is used to execute the code, and then Get methods are used to retrieve output values. The examples below illustrate the calling of a stored procedure and a stored function.

Calling a Procedure

The code below executes an Oracle Stored Procedure. Note the use of substitution variables for both input and output parameters. Variables are represented by question marks (?) in the PL/SQL code block. Variables are referenced via relative position in the statement.

```
CallableStatement stmt1 = conn.prepareCall("begin addem(?,?,?); end;");
stmt1.setString(1,"123");
stmt1.setString(2,"456");
stmt1.registerOutParameter(3,Types.VARCHAR);
stmt1.executeUpdate();
System.out.println("Value returned is " + stmt1.getString(3));
```

Calling a Function

The code below executes an Oracle Stored Function. Again, note the use of substitution variables for both input and output. Once again, variables are represented by question marks (?) in the PL/SQL code block and are referenced using relative position in the statement.

```
CallableStatement stmt2 = conn.prepareCall("begin ? := times_2(?); end;");
stmt2.registerOutParameter(1,Types.NUMERIC);
stmt2.setString(2,"1234.56");
stmt2.executeUpdate();
java.math.BigDecimal outVal = stmt2.getBigDecimal(1);
System.out.println("Value returned is " + outVal);
```

CallableStatement Methods

Some additional CallableStatement methods not shown in the prior example include:

- getBigDecimal
- getBoolean
- getByte
- getBytes
- getDate
- getDouble
- getFloat
- getInt
- getLong
- getObject
- getShort
- getString
- getTime
- getTimeStamp
- registerOutParameter
- wasNull

OracleCallableStatement Extensions

Once again, Oracle's extensions allow processing of variables without worrying about conversions. The include:

- getArray
- getBfile
- getBlob
- getClob
- getCursor
- getCustomDatum
- getNumber
- getOracleObject
- getRaw
- getRef
- getRowid
- getStruct

Oracle Documentation

Oracle's documentation is voluminous and very useful concerning Java and JDBC use in the database. The following manuals are all useful:

Oracle8i CORBA Developer's Guide and Reference

Oracle8i Enterprise JavaBeans Developer's Guide and Reference

Oracle8i Java Developer's Guide

Oracle8i Java Stored Procedures Developer's Guide

Oracle8i Java Tools Reference

Oracle8i JDBC Developer's Guide and Reference

Oracle8i JPublisher User's Guide

Oracle8i Oracle Servlet Engine Release Notes

Oracle8i Oracle Servlet Engine User's Guide

Oracle8i SQLJ Developer's Guide and Reference

Oracle8i Supplied Java Packages Reference

Oracle JavaServer Pages Developer's Guide and Reference

In addition, Javadoc is available for for Oracle JDBC as part of the database installation, it may be found at: <oraclehome>/jdbc/doc/javadoc.zip

Finally, lots of papers and examples are available on the Oracle Technet website: http://technet.oracle.com

Wrapping it all Up

JDBC access allows full use of Oracle from Java programs in the form most familiar to programmers with Java experience.

All Java access to Oracle may use JDBC's standard features. Oracle-specific features improve performance at the cost of portability and their use should perhaps be limited to those areas where performance advantage may be demonstrated.

Reviewing the steps used to access Oracle via JDBC:

1. Load/register the JDBC driver
2. Connect to the database
3. Create a statement object
4. Execute SQL statements and process results
5. Close the result set (if used), statement, and connection
   (probably from a finally block)

Conclusion

Java is "the thing" in the IT industry today and Oracle is the most commonly available database. Oracle, IBM, Sun, and many others have spent vast sums to make Java the emerging "standard" - an ISO/ANSI standard is still needed before Java can truly claim to be a standard. Oracle allows access to stored database procedures using Java supporting "n-tier" applications.

Java is a third-generation language, only by using an advanced development environment like Oracle JDeveloper, Sun's Forte, IBM VisualAge for Java, Borland JBuilder, or WebGain Cafe can manually writing large amounts of code be avoided.

The industry is moving away from using client-side applications and applets and replacing them with an HTML user interface, servlets, and JSPs Java is a good investment of your time and energy!

Organizations must decide at what level the Oracle Thin and OCI drivers should be used. Also, a decision must be reached as to the use of Oracle-specific extensions to JDBC. (Performance versus Portability)

DBA's must verify that application developers are not only including reasonable SQL in JDBC programs, but, must also make sure that database connections are properly handled.

About the Author

John founded King Training Resources in 1988, he has been working with Oracle products since Oracle Version 4 and has been teaching Oracle-related courses since 1986. He has presented papers at various user group conferences including: IOUG-A Live!, EOUG, UKOUG, ODTUG, SEOUC, and the RMOUG Training Days.

This paper, the presentation slides, and sample SQL may be downloaded from http://www.kingtraining.com.