USING JAVA AND JDBC IN AN INTRODUCTORY DATABASE COURSE

Thomas B. Gendreau
Computer Science Department
University of Wisconsin - La Crosse
gendreau@csfac.uwlax.edu

Abstract

JDBC is one of the standard packages available with Java. It provides classes and interfaces that allow application program/database interaction with a high degree of database vendor independence. The features of JDBC enable an application program to connect to remote databases, execute SQL queries or stored procedures on the remote database, and process the results of queries a row at a time. The SQL statements are represented as strings in the Java program. A string representing a SQL expression can be hardcoded into the program or built during execution based on user input. JDBC also includes features that allow the program to query the database to determine metadata such as the names and structure of tables in the database. Access to metadata objects allows very flexible application programs to be created. This paper gives an overview of the features of JDBC and discusses its use in an introductory database course.

Introduction

At the University of Wisconsin - La Crosse CS 364:Introduction to Database Management Systems is the first course in database systems offered to computer science and information systems majors. The course includes traditional topics such as writing SQL queries, ER modeling, and normalization. Three years ago a client/server database programming component was added to the course. In this part of the course, students are taught database application programming using features of Java and JDBC. This paper gives an overview of the features of JDBC that have been used in CS 364 and discusses some projects that have been used in CS 364.

JDBC Basics

There following code segment uses many of the basic features of JDBC. The code segment connects to the database and outputs the contents of the Author table to standard output. (Assignments used in CS 364 always use GUIs but in order to focus on the JDBC features in this paper we use code segments with simpler user interfaces)

```
Connection con;
Statement stmt;
ResultSet rs;
try {
        DriverManager.registerDriver( new oracle.jdbc.driver.OracleDriver());
        con = DriverManager.getConnection(
                        "jdbc:oracle:thin:@cs364.intra.uwlax.edu:1521:cs364",
                        "login", "password");

        stmt = con.createStatement();

        rs = stmt.executeQuery("select * from Author");
        while (rs.next())
        {
                System.out.println(rs.getString(1) + " " + rs.getString(2));
        }
        stmt.close();
}

catch (SQLException e)
{
        //code not shown
}
```

Before any interaction between the client program and the database server can happen, the client program must create a connection to the server. To make the connection, an appropriate driver must be loaded and a call to DriverManager.getConnection must be

made. The drivers are database server dependent. The above code works for an Oracle 8 server. A MySQL server was used the first year JDBC was used in CS 364. The following code segment was used to make a connection to that server.

```
Class.forName("org.gjt.mm.mysql.Driver");
con =DriverManager.get Connection
   ("jdbc:mysql://nabokov.cslab.uwlax.edu:3306/Libraries",
      "login","password");
```

The format of the url string passed to the driver varies by driver and database server. In general the url provides a reference to the machine on which the server runs, a port number to connect to (sometimes implicit), and a database instance name. More information on drivers can be found in [1] and at java.sun.com.

Before a program executes an SQL expression some type of statement object must be created. The simplest statement class is Statement. The above code creates a Statement object referenced by the name stmt. The methods executeQuery and executeUpdate can be invoked on a Statement object. executeQuery is used to execute an SQL select statement. The parameter to executeQuery is a string that should be a syntactically correct SQL expression. The SQL strings can be hardcoded, as in the above example, but more frequently the strings are built during execution based on user input. For example, a user interface could be built that lets a user choose tables, attributes and conditions (think of how simple queries can be built in Access) and based on the user's choices an SQL string is built for the query. (In CS 364 programming assignments students frequently make use of the StringBuffer class to incrementally build the SQL expressions). It is important to realize that Java does not recognize SQL. If an SQL string has syntax errors, this fact will be recognized by the server at runtime and an SQLException will be raised by the program.

A call to executeQuery returns a ResultsSet. Through the ResultSet the program can access the rows returned from the select statement. In order to access the rows (including the first row) the method next must be invoked on the ResultSet object. The next method returns true as long as another row is available. To access attributes from the current row, methods such as getString or getInt can be invoked on the ResultSet. Attributes can be referenced by position, as in the above code, or by name. For example, the call rs.getString(2) could have been replaced by the call rs.getString("authname").

Other Types of Statements

There are two other types of statement classes: PreparedStatements and CallableStatements. PreparedStatements are used to create SQL expression templates that can be parameterized. For example the following PreparedStatement can be used to find the titles of books written by a particular author. The author's name used in the query is not given until the setString method is called.

```
PreparedStatement ps = con.preparedStatement
   ("select B.title from Book B, Author A where B.aid = A.aid and" +
         "A.authname = ?");
ps.setString(1,"Smith");
ResultSet rs = ps.executeQuery();
```

Note, associated with each PreparedStatement is a particular SQL expression, so a string is not passed to the executeQuery method. Before the executeQuery is called, values must be given to each of the parameters. The above query has only one parameter but queries can have many parameters. Each parameter is represented by a "?". Parameters are given values by position using the various setXXX methods. If the same query with different parameters is executed many times, it may be faster to use a PreparedStatement instead of a Statement. With PerparedStatements the template is sent to the server when the statement is created and part of the work of executing the query can be done before the parameters are set. This work will not have to be repeated for each set of parameters.

CallableStatements are used to invoke stored procedures. Stored procedures can provide some benefits in terms of throughput because smaller amounts of data are passed between the client application and the database server. Stored procedures are usually written in a database server specific language. For example, Oracle stored procedures are usually written in a language called PL/SQL. Through CallableStatements java programs can pass parameters to stored procedures and can get return values from stored procedures. In the following code segment assume the stored procedure, called P1, expects a string as a value parameter (IN in PL/SQL syntax) and returns a VARCHAR as a result parameter (OUT in PL/SQL syntax). The call to registerOutParameter indicates that the second parameter is a result parameter.

```
CallableStatement cs = con.prepareCall("{call P1(?,?)}");
cs.setString(1,"mystring");
cs.registerOutParameter(2, java.sql.Types.VARCHAR);
cs.executeQuery();
System.out.println(cs.getString(2));
```

ResultSet Options

The ResultSet used in the first example is the simplest type of ResultSet. It is referred to as a forward-only ResultSet. With a forward-only ResultSet a program can only move in one direction through the results of a query. This is a serious limitation in the creation of flexible GUIs. For example, a forward-only ResultSet makes it hard (or inefficient) to implement a program to allow a user to see the results of a query in a window in which the user can navigate forward and backward. Other options for ResultSets are to make the ResultSets scroll-insensitive or scroll-sensitive. A scroll-insensitive ResultSet allows the program to move forward and backward through the results of a query. It also enables the program to move to specific rows in the result. Calling the ResultSet insensitive means that changes made to the underlying tables that were part of the query will not be seen in

the ResultSet. A ResultSet that is scroll-sensitive can move around the results like a scroll-insensitive ResultSet and it can see the effects of changes to the results based on changes to the underlying tables that were part of the query. The limitations on the kinds of changes that can be seen are database server dependent. For example, see the discussion of keyset-driven semantics in [2].

ResultSets can also be distinguished based on whether the ResultSet is read-only or updatable. Through an updatable ResultSet the underlying table(s) of the query can be modified through the ResultSet. The limitations on the types of queries that could produce updatable ResultSets depend on the restrictions the server places on updatable SQL views. Usually the restriction is that the query uses only one table and the attributes in the result of the query include the primary key of the table.

The type of ResultSet that can be created by executing a query on a statement is determined when the statement object is created. For example the following code segment creates a Statement from which a scroll-sensitive/updatable ResultSet can be created.

```
Statement st = con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
                    ResultSet.CONCUR_UPDATABLE);
```

MetaData

Metadata is information about features of the database server, the database schema or the structure of a ResultSet. The use of metadata information is necessary for creating flexible applications. In the case of an application written to communicate with a particular database instance, metadata may not be required but if the application is designed to work with many database instances and schemas, then metadata is essential. The following code segment creates a DatabaseMetaData object.

```
DatabaseMetaData md = con.getMetaData();
```

Given a DatabaseMetaData object a program can query about properties of the database server such as the data types supported by the server. A DatabaseMetaData object can also be used to query about the structure of a particular database instance. For example an application could be written in which the input includes the database instance to which the user wants to connect. After the program is given the database instance information, the program could create a connection to the database and create a DatabaseMetaData object. With the DatabaseMetaData object the program can determine the tables in the database instance schema. The following code segment invokes getTables on a DatabaseMetaData object to produce a ResultSet from which the tables in the database can be found. There will be one row in the ResultSet for each table in the database instance.

```
ResultSet rs = con.getMetaData().getTables("",dbname, "%",null);
```

See java.sun.com for details of the attributes in each row of the result. For example, attribute 3 is the name of the table. The interpretation of some of the parameters to getTables and other DatabaseMetaData methods is database server dependent. For example the above call was made to an Oracle server and since Oracle does not use catalogs the first parameter is a null string.

Another type of metadata is ResultSetMetaData. This type of metadata enables a program to find characteristics about a particular ResultSet. For example, if a query is created at runtime based on user input, the type and number of attributes in the result is not known when the program is created. To process the results of the query at runtime, the program needs to determine the types and number of attributes in the result. The following code segment can be used to dump the contents of a table to standard output. The table name is given as a command line argument. The code segment assumes all the types of the columns are compatible with getString (most types such a VARCHAR, INTEGER, etc. are compatible with getString). The code segment could be made more flexible by using the method getColumnType to find the type of a column and thus to determine the getXXX method to use to get the value of the column.

```
ResultSet rs = st.executeQuery("select * from " + args[0]);
int numCols = rs.getMetaData().getColumnCount();
while (rs.next())
{
        for (int cols = 1; cols <= numCols; cols++)
                System.out.print(rs.getString(cols) + " ");
        System.out.println();
}
```

Support for Transactions

After a connection is made to the database server the execution of each individual SQL expression is treated as a separate transaction. If the application requires the execution of a group of SQL expressions to be treated as a single transaction, the following code segment can be executed.

```
con.setAutoCommit(false);
```

After this line of code is executed, database updates will not be committed until an explicit call to commit is made. If during the execution of the sequence of SQL expressions that are part of the transaction an event occurs that requires the transaction to stop, then the method rollback can be called. Support for transactions, such as the isolation level, is dependent on the implementation of transactions on the database server.

Student Programming Projects

Before students enroll in CS 364 they must complete a year-long course in software development. The language used in the course is Java. The JDBC part of CS 364 is taught after material on the basics of the relational data model, SQL, and ER modeling. About three weeks is spent discussing database application programming and JDBC. When the JDBC lecture material is completed, the students are assigned a development project. The students have about 4 weeks to complete the project. During the last week of the semester, students demonstrate the project, one-on-one, for the instructor.

Most of the projects involve implementing a client interface to perform operations on a database from the textbook. These databases have included a library database [3], a video store database [4], and a customer/agent database [5]. In the most recent project, students were given a problem description for a seed company database. Students created an ER design of the database and then implemented an application to perform operations on the database.

Each project includes a list mandatory operations the students are required to implement. Students also specify additional operations they want to implement. Projects are evaluated based of the quality of the implementation and on the number of operations supported by the implementation. Operations that have been implemented include borrowing a book from a library, placing an order for a customer, returning a video, or checking the status of an outstanding seed order. Some projects have included a general query evaluation tool that required use of DatabaseMetadata. All the JDBC features described earlier in the paper have been used in some CS 364 project with the exception of CallableStatement and transactions (these have been used with undergraduates in a second database course).

Since the drivers for both Oracle and MySQL are freely available and since many students have computers with Internet access available at home, many students developed their programs at home. This was not only convenient for the students but also gave them a better sense of the flexibility of JDBC and distributed natured of the software they developed.

Database Servers

CS 364 has used two different database servers. The first year JDBC was taught, a MySQL server running on Linux on G3 Apple hardware was used. The JDBC component was a late addition to the course content and MySQL was the most easily accessible server software available. MySQL was easy to install and maintain. The semester it was used CS 364 had 95 students and MySQL supported 95 small database instances on a desktop class machine.

In subsequent semesters an Oracle 8 server running on Windows 2000 has been used. Oracle is a much larger more complex product than MySQL. There are numerous reasons why we started using Oracle. Some of them include the increase use of Oracle at UW-L

for administrative functions, the availability of Oracle through a state wide licensing agreement, the desire of the IS department to use Oracle in IS 411 (CS 364 is a prerequisite to IS 411), and the sense that both CS students and IS students would benefit for experience with a database server that is so widely used in industry.

In the past I have maintained an Oracle server but currently the Information Technology department at UW-L maintains an Oracle server for instruction purposes. An Oracle database server is a very complex product that can require a lot of system administration experience. With appropriate support I think Oracle is a good choice. However, if a department has limited resources and limited system administration experience, MySQL might be a better choice since it is easier to maintain. When I used MySQL it had limited support for transactions and nested queries but like most software it continues to evolve and even the version I used three years ago had sufficient power for an introductory course.

Conclusion

Using JDBC in CS 364 has significantly improved the course. Students build on their Java experience and on the relational database ideas taught in the course to develop applications that have many of the characteristics of commercial database applications. I think this experience motivates students and helps them retain the basic database skills and knowledge taught in the course.

References

1. Speegle, Gregory D. (2002). *JDBC: Practical Guide for Java Programmers*. Morgan Kaufmann.

2. Lewis, Philip M., Bernstein A., and Kifer, M. (2002), *Databases and Transaction Processing*. Addison-Wesley.

3. Johnson, James L. (1997). *Database: Models, Languages, Design*. Oxford University Press.

4. Riccardi, Greg (2001). *Principles of Database Systems with Internet and Java Applications*. Addison-Wesley.

5. O'Neil, Patrick and O'Neil, Elizabeth (2001). *Database: Principles, Programming, and Performance*. Morgan Kaufmann.

Acknowledgements

CS 364 serves both Computer Science and Information Systems students. For Information Systems students the course is a prerequisite to IS 411. I want to thank Bill

Wehrs for many good conversations we have had about the content of CS 364 and about making the content of CS 364 and IS 411 work to the benefit of the students. Also I want to thank John Tillman, Bob van Abel, Heath Ahnen, Mike McGargle, Sandy Suchla, and Barry Sommers, for providing IT resources and support needed to manage the Oracle server.