Return to Module Overview Page (https://onlinelearning.berkeley.edu/courses/665040/pages/module-ten)

10.10: The JDBC API

Databases of one form or another have been around since the earliest days of computers. For many years, each brand of database software had its own proprietary API, which made it very difficult to upgrade a database or switch vendors. Over the years, a series of standards have emerged to smoothe over many of these differences. First there was the introduction of the standard Structured Query Language (SQL). SQL is now supported by virtually every relational database product. SQL standardized the contents of database queries, but not the programming API to make the queries. Enter the Open DataBase Connectivity (ODBC) API. ODBC provides a standard API that can be used to send queries to every major database vendor's products. ODBC works via a middle layer of vendor-specific software called a *driver*, which turns standard ODBC method calls into vendor-specific API calls. ODBC has become extremely popular and widespread.

Now there is the Java DataBase Connectivity (JDBC) API. JDBC is inspired by, but simpler than, ODBC, and is far more object-oriented than ODBC as well. JDBC presents a uniform view of databases from the Java programmer's perspective. Thanks to the existence of a JDBC-ODBC *bridge* (a sort of two-ended JDBC/ODBC driver that connects JDBC to the standard ODBC libraries on Win32 and on Solaris), JDBC can be used with dozens of different database products: Sybase, Oracle, Microsoft's SQL Server and Access, and plenty more. Many companies are writing or have written direct JDBC drivers for their database that makes using the bridge unnecessary.

JDBC is very easy to use. To make a query on a database you simply load the JDBC driver class for that database, obtain a Connection object, obtain a Statement object from the Connection object, and call Statement.executeQuery( ). This last function returns a ResultSet object, which represents the tabular result of the query. All of these JDBC classes are in the package java.sql.

```
Class.forName ("sun.jdbc.odbc.JdbcOdbcDriver"); // See Note #1 below

     Connection cxn =
DriverManager.getConnection("jdbc:odbc:Books",
                            "", "");            // Note #2

Statement stmt = cxn.createStatement();

ResultSet rs =
stmt.executeQuery("SELECT FirstName, LastName
                  FROM Authors");              // Note #3
```

The ResultSet object conceptually contains all the rows returned by the query (the data may actually still reside in the database and be fetched as needed.) You can investigate the query result by interrogating ResultSet object.

```
ResultSetMetaData md = rs.getMetaData();    // Note #4
int cols = md.getColumnCount();
```

```
    while (rs.next ())                         // Note #5
    System.out.println(rs.getString(1) + " " +
                       rs.getString(2));
```

Note #1: This line forces the JVM to load the class containing the driver for the database we're interested in. Here we're loading the JDBC/ODBC bridge. When a Driver class is loaded, it automatically registers itself with the DriverManager class.

Note #2: This line opens a connection to the database named in the first argument. JDBC database names are similar to URLs. The first part is always "jdbc," but the later parts depend on the particular driver in use. Here we're opening one of the standard demonstration databases that comes with Microsoft Access. To use this code under Windows 95 or Windows NT, you would need to first install this database using Access, then designate it as an ODBC data source named "Books" using the ODBC Control Panel. The Books database is an example library catalog.

Note #3: The query is in standard SQL: we're requesting a list of every author's first and last names from the database.

Note #4: A ResultSetMetaData object contains lots of useful information about a ResultSet; for example, the names of the columns and, as here, the number of columns.

Note #5: You can't get the number of rows from a ResultSet. You can only traverse the rows one at a time using rs.next( ), as we're doing here. The getString( ) method is just one of many with which you can retrieve column entries; there are methods to retrieve all the standard SQL types. Note that column numbers start with 1, not zero.