Chapter 15
SQL using Java

JDBC Overview

- Call-level interfaces such as JDBC (Java Database Connectivity) are programming interfaces allowing external access to SQL database manipulation and update commands.
- They allow the integration of SQL calls into a general programming environment by providing library routines which interface with the database

JDBC API

- JDBC is a Java API (Application Programming Interface) that documents a standard framework for dealing with tabular and, generally, relational data.

- The JDBC API is an interface specification designed to abstract database applications from the particular database product utilized.
- Revision 3.0 is currently available with the 1.4 release of the JDK.
- Allows developer to concentrate on application instead of learning a vendor API

JDBC Drivers

- A *JDBC driver* is a class that implements the *JDBC Driver* interface and understands how to convert program (and typically SQL) requests for a particular database.

- There are four different driver types
    1. Thin driver
        - Small footprint
        - Good to use with applets
        - Works only with TCP/IP and requires Oracle Net to be running
    2. OCI driver
        - Better performance than the thin driver
        - Suitable for programs deployed on the middle tier, e.g. a web server.
    3. Server-Side Internal Driver
        - Provides direct access to the database
        - Used by the Oracle JVM to communicate with that database.
        - The Oracle JVM is integrated with the database.
    4. Server-Side Thin Driver
        - Also used by the Oracle JVM.
        - Provides access to remote databases.

JDBC and Vendors

- Driver for a particular product (database) is written by the database vendor e.g. Oracle
- Driver complies with the java.sql interfaces defined by Sun
- Oracle's JDBC drivers are available via:
  – http://otn.oracle.com/software/tech/java/sqlj_jdbc/index.html

Basic Steps

1. Establish a Connection
   a. Load a **Driver** compatible with the DB
   b. Make the **Connection** to DB
2. Associate an SQL **Statement** with the **Connection**
3. Execute the SQL **Statement**
4. Process the **ResultSet**
5. Execute further SQL **Statement**s
6. Close the **Connection**

1.Establishing A Connection

- First need Java, JDBC and the database
- The downloaded driver should be on the CLASSPATH (Filesystem in NetBeans) for ease of use

- Next must open a connection between the program(client) and the database(server).
- This involves two steps:
  - Load the vendor specific driver
  - Make the connection

a. Load the vendor specific driver

Why?

– To ensure portability and code reuse, the API was designed to be as independent of the version or the vendor of a database as possible
– Need the driver specific to the DBMS

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

b. Make the connection

- Once the driver is loaded and ready for a connection to be made, you may create an instance of a Connection object using:

```
Connection con =
     DriverManager.getConnection
   ("jdbc:oracle:thin:@codd.cs.camosun.bc.ca:1521:orcl10", "JDBCTEST",
   "password");
```

[Note: This connect string may change.]

1.Establishing A Connection, b. Make the connection (cont.)
("jdbc:oracle:thin:@codd.cs.camosun.bc.ca:1521:orcl10", "username", "passwd");

- The first string is the URL for the database including the protocol *(jdbc)*,
- the vendor *(oracle)*,
- the driver *(thin)*,
- the server *(*codd.cs.camosun.bc.ca*)*,
- the port number *(1521)*,
- and a server instance *(*orcl10*)*.
- The username and passwd are *your* username and password, the same as you would enter into *SQLPLUS* to access your account.

2. Associate SQL Statement

- A JDBC Statement object is used to send your SQL statements to the DBMS
- A JDBC Statement object is associated with an open connection, and not any single SQL Statement.

- An active connection is needed to create a Statement object:

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

- At this point, a Statement object exists, but it does not have an SQL statement to pass on to the DBMS.

3. Execute SQL Statement: CREATE / INSERT / UPDATE

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

stmt.executeUpdate("CREATE TABLE Dogs " +
"(breed VARCHAR2(40), asize VARCHAR2(15), wt REAL)" );

stmt.executeUpdate("INSERT INTO Dogs " +
"VALUES ('Border Collie', 'Medium', 15.00)" );
```

- And

```
String sqlString = "CREATE TABLE Breeders " +
"(name VARCHAR2(40), address VARCHAR2(80), lic INT)" ;
stmt.executeUpdate(sqlString);
```

- When executeUpdate is used to call DDL statements, the return value is always zero.
- Data modification statement executions will return a value greater than or equal to zero, which is the number of tuples affected in the relation.

4. Process Result Set: Executing SELECT statements

- Returns a set of tuples and does not change the state of the database.
- Uses a method called executeQuery, which returns its results as a ResultSet object.

```
String breed, asize ;
float wt ;
ResultSet rs = stmt.executeQuery("SELECT * FROM Dogs");
   while ( rs.next() ) {
        breed = rs.getString("breed");
        size = rs.getString("asize");
        wt = rs.getFloat("wt");
        System.out.println(breed + "s  are of " + asize + "  size & weigh  " + wt + "
   kilos.");
}
```

- The tuples resulting from the query are contained in the variable rs which is an instance of ResultSet.
- Need to be able to access each row and the attributes in each row.
  - The ResultSet provides a cursor, which can be used to access each row in turn.
  - The cursor is initially set *just before* the first row.
  - Each invocation of the method *next* causes it to move to the next row, if one exists and return true, or return false if there is no remaining row.

- We can use the getXXX method of the appropriate type to retrieve the attributes of a row
- For example:

```
breed = rs.getString(1);
wt = rs.getFloat(3);
asize = rs.getString(2);

ResultSet rs = prepareUpdateDogs.executeQuery() ;
```

- JDBC also offers a number of methods to find out position in the result set:
  - getRow
  - isFirst
  - isBeforeFirst
  - isLast
  - isAfterLast

4. Process Result Set :Accessing ResultSet

- By default, cursors scroll forward only and are read only.
- When creating a Statement for a Connection, you can change the type of ResultSet to a more flexible scrolling or updatable model:

```
Statement stmt = con.createStatement( ResultSet.TYPE_FORWARD_ONLY,
   ResultSet.CONCUR_READ_ONLY);
ResultSet rs = stmt.executeQuery("SELECT * FROM Dogs");
```

5. Other SQL Statements: Transactions

- JDBC allows SQL statements to be grouped together into a single transaction
- Transaction control is performed by the Connection object.
- When a connection is created, by default it is in the auto-commit mode.

- Can turn off auto-commit mode for an active connection with :
  con.setAutoCommit(false) ;

- and turn it on again with :
  con.setAutoCommit(true) ;
- When auto-commit is off, no SQL statements will be committed until you call the method commit explicitly:
  con.commit() ;

- Before commit, can invoke rollback() con.rollback() ;

```
con.setAutoCommit(false);
Statement stmt = con.createStatement();
stmt.executeUpdate("INSERT INTO Dogs VALUES('Border Collie', 'Medium',
   15.00)" );
con.rollback();
stmt.executeUpdate("INSERT INTO Dogs VALUES('Great Dane', 'Large', 60.00)"
   );
con.commit();
con.setAutoCommit(true);
```

Handling Errors with Exceptions

- Two levels of error conditions, SQLException and SQLWarning.
  - SQLExceptions
    - are Java exceptions which, if not handled, will terminate the application.
  - SQLWarnings
    - are subclasses of SQLException, but they represent nonfatal errors or unexpected conditions, and as such, can be ignored

Handling Errors with Exceptions

```
try {
   con.setAutoCommit(false) ;
   stmt.executeUpdate("CREATE TABLE Dogs "
        "(breed VARCHAR2(40), asize VARHAR2(15), wt      REAL)" );
   stmt.executeUpdate("INSERT INTO Dogs VALUES          ('Border Collie',
   'Medium', 15.00)" );
   con.commit()
   con.setAutoCommit(true) ;
} catch(SQLException ex) { System.err.println("SQLException: " +
   ex.getMessage()) ;
   con.rollback() ;
   con.setAutoCommit(true) ;
}
```

6. Close the Connection

- May want to first commit any changes before closing:
  con.commit() ;
- To be thorough, should also close any open statement objects:
  stmt.close();
- Finally, close the open connection:
  con.close();

Creating JDBC PreparedStatement

- May be more convenient or efficient to use a PreparedStatement object for sending SQL statements to the DBMS
- Unlike Statement, it is given an SQL statement right when it is created
- This SQL statement is then sent to the DBMS right away, where it is compiled
- In effect, a PreparedStatement is associated as a channel with a connection and a compiled SQL statement

- Advantage offered is that if the same, or similar query with different parameters is used multiple times, the statement can be compiled and optimised by the DBMS just once.
- The *Statement* requires a compilation each time.

- PreparedStatements are also created with a Connection method

```
PreparedStatement prepareUpdateDogs = con.prepareStatement( "UPDATE
   Dogs SET wt = ? WHERE breed = ? AND asize = ?");
```

Creating JDBC PreparedStatement

- Before execute a PreparedStatement, must supply values for the parameters
  - Done by calling one of the setXXX methods defined in the class PreparedStatement
  - Often used methods are
    - setInt,
    - setFloat,
    - setDouble,
    - setString

- Example
  - prepareUpdateDogs.setFloat(1, 15);
  - prepareUpdateDogs.setString(2, "Border Collie");
  - prepareUpdateDogs.setString(3, "Medium");

Executing CREATE / INSERT / UPDATE

- Executing SQL statements in JDBC varies depending on the ``intention'' of the SQL statement.
- DDL (data definition language) statements are all executed using the method executeUpdate

- Using *PreparedStatement*, execute by first plugging in the values of the parameters (as seen earlier), and then invoking the executeUpdate on it.

```
int n = prepareUpdateDogs.executeUpdate() ;
```

Oracle and Java Types

Oracle Type	Java Type
CHAR	string
VARCHAR2	string
DATE	java.sql.Date java.sql.Time java.sql.Timestamp
INTEGER	short int long
NUMBER	float double java.math.BigDecimal

Tutorial

- http://java.sun.com/docs/books/tutorial/jdbc/TOC.html#basics

Summary

- The JDBC API enables Java to access a database
- The Oracle JDBC drivers are used to connect to an Oracle database
- SQL statements may be executed using JDBC