Java Database Connection

Introduction

- **Connection to a database is an important aspect of java programming**
- **At the end of this section you will be able to:**
  - connect to a database
  - Create tables
  - Update tables and select from tables

Vocabulary

- **Connection**
  - A session with a database opened by JDBC application program. It represents a connection and a (remote) database.
- **Driver**
  - Software that implements all of the API in the **java.sql** and **javax.sql**

Vocabulary

- **JDBC: Java Database Connectivity**
  - Defines a set of API objects and methods that interact with an underlying database. May be pure java or interact with ODBC
- **ODBC: Open Database Connection**
  - An API defined by Microsoft

Load the JDBC driver

- **The software that knows how to talk to the database; each database (Oracle, mySql, etc.) will have a different driver**
- **Obtain driver from database manufacturer**
  - Add library to your project (do not **unjar**)
  - Point **classpath** to driver jar

Load the JDBC driver

- **Creates a driver and registers it with DriverManager**

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

Establish a connection

- **Connect the appropriate driver to the DBMS**
  - Specify the location of the database:
    - url
  - Specify user information:
    - username
    - password
- **Use the DriverManager, established by the Class.forName(), to create a connection**

Establish a connection

```
Connection con = DriverManager.getConnection(URL, user,
  pass);
```

- **URL: a String giving the location of the database**

  `jdbc:oracle:thin:@HOSTNAME : PORT:SID`

  Example：

  `jdbc:oracle:thin:@app2510.ict.sait.ca:1521:course`

- **User: the user name to sign into the database**
- **Pass: the password associated with the username**

Create a Statement

- **A Statement object is used to send queries and updates to the database**
- **It is created from the Connection as follows:**

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

Execute Statements

- **Statements are used to send SQL commands to the database**
- **DDL statements:**
  - create, alter, or drop a table, insert into a table
  - use the executeUpdate() method
  - statement.executeUpdate(command);
    - command is a string representing a SQL expression

Execute Statements (Query)

- **Query a database**
- **Used to retrieve data from a database**
- **executeQuery(expression)**
  - Expression is a String representing a SQL query
  - “Select * from employee”

Execute Statements (Query)

- **executeQuery(expression) returns a ResultSet**
- **A ResultSet contains the information in the rows that satisfied the query (similar to a cursor)**

Process The Results

Extract the information from the ResultSet

- The next() method in ResultSet sets the next row to the current row
- Create a **while** loop

```
while(rs.next())
  {
  }
```

- Body of the loop will extract information

Process The Results

Useful methods in the ResultSet Class

- getString(columnName)
  - Returns a String value associated with a given column
- getDouble(columnName)
  - Returns a double value associated with a given column

Close / resources

- **Close the Statement**
  - Releases resources associated with the Statement
- **Close the Connection**
  - Releases resources associated with the Connection

Executing a query

```
Statement stmt = null;
String query = "select COF_NAME, SUP_ID, PRICE, " + "SALES, TOTAL " + "from " + dbName + ".COFFEES";
try {
        stmt = con.createStatement();
        ResultSet rs = stmt.executeQuery(query);
        while (rs.next()) {
            String coffeeName = rs.getString("COF_NAME");
            int supplierID = rs.getInt("SUP_ID");
            float price = rs.getFloat("PRICE");
            int sales = rs.getInt("SALES");
            int total = rs.getInt("TOTAL");
            System.out.println(coffeeName + "\t" + supplierID + "\t" + price + "\t" + sales + "\t" + total);
             }
} catch (SQLException e ) {
……
}
finally {
             if (stmt != null)
                          stmt.close();
}
```

Using Prepares Statement

```
PreparedStatement updateSales = null;

String updateString = "update " + dbName + ".COFFEES " + "set SALES = ? where COF_NAME = ?";

try {

        con.setAutoCommit(false);

        updateSales = con.prepareStatement(updateString);

        for (Map.Entry<String, Integer> e : salesForWeek.entrySet()) {

                updateSales.setInt(1, e.getValue().intValue());

                updateSales.setString(2, e.getKey());

                updateSales.executeUpdate();

        }

} catch (SQLException e ) {

        ….

}finaly{

…

}
```

Summary

- Load the JDBC Driver
- Establish a connection
- Create a Statement
- Execute statements
- Process the results
- Close resources