Databases with Java

JDBC

- “An API that lets you access virtually any tabular data source from the Java programming language”

Database Drivers

- Think of a database as just another device connected to your computer
- like other devices it has a driver program to relieves you of having to do low level programming to use the database
- the driver provides you with a high level api to the database
- Make sure you install the driver along with the DBMS. In class we installed connectorJ for MySQL.
- In order to use this driver in your code, you can add the driver in eclipse by adding an external jar file containing the library. As we did in class by adding the jar file of connectorJ in Project properties of our Database project.

Basic steps to use a database in Java

- 1.Establish a **connection**
- 2.Create JDBC **Statements**
- 3.Execute **SQL** Statements
- 4.GET **ResultSet**
- 5.**Close** connections

1. Establish a connection

- Establish a connection with the DB

  `Connection c = DriverManager.getConnection(url, uid, password);`

  – Searches the registered Drivers until it finds the match.
  – The `Driver` class then establishes the connection, returns a `connection` object with the help of the DriverManager class.

- For Oracle:

String url = "jdbc:oracle:thin:@oracleprod:1521:OPROD”;
String username = “Write Username Here“;
String pwd = “Write password Here“;
Connection con = DriverManager.getConnection(url, username, pwd);

- Similarly for MySQL:

String url = "jdbc:mysql://localhost:3306/BCS15“;
String username = “Write Username Here“;
String pwd = “Write password Here“;
Connection con =DriverManager.getConnection(url, username, pwd);

2. Create JDBC statement(s)

- `java.sql.Statement`
  - Most basic class. It performs all the SQL statements
- Statement stmt = con.createStatement() ;
- Creates a Statement object for sending SQL statements to the database
  - `executeQuery( String )`
  - `executeUpdate( String )`
  - `execute( String )`

Executing SQL Statements

- You can create a Table e.g.

```
String createLehigh = "Create table Lehigh " + "(SSN Integer not null,
                                Name VARCHAR(32), " + "Marks Integer)";
stmt.executeUpdate(createLehigh);
```

- You can insert values in a table e.g.

```
String insertLehigh = "Insert into Lehigh values" + "(123456789,abc,100)";
stmt.executeUpdate(insertLehigh);
```

Get ResultSet

`java.sql.ResultSet`

One or more rows of data returned by a query

```
Statement st = c.createStatement();
ResultSet rs = st.executeQuery("…");
```

Methods: `next(),getString(column),getInt(..),` `last(), getRow()`

```
String queryLehigh = "select * from Lehigh";

ResultSet rs = Stmt.executeQuery(queryLehigh);

while (rs.next()) {
    int ssn = rs.getInt("SSN");
    String name = rs.getString("NAME");
    int marks = rs.getInt("MARKS");
}
```

Close connection

- stmt.close();
- con.close();

Sample program: Using the Select Statement

```
import java.sql.*;
public class Test1 {

public static void main(String[] args) {

try {
Connection conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/BCS15","root","ALI");
Statement stm = conn.createStatement();
String str = new String("select * from batch");
ResultSet rs=stm.executeQuery(str);
while(rs.next()){
System.out.println(rs.getString(1)+" " + rs.getString(2)+" "+rs.getString(3)+" "+rs.getString(4));
}

rs.close();
stm.close();
conn.close();
} catch (SQLException e) {
e.printStackTrace();
}
}
}
```

Sample program: Inserting Values in a Table

```
import java.sql.*;
public class Test2 {
    public static void main(String[] args) {
        try {
        Connection conn =
    DriverManager.getConnection("jdbc:mysql://localhost:3306/BCS15","root
    ","ALI");
                Statement stm = conn.createStatement();
                String str = new String("insert into batch values
    (17,'BCS',27,'2021-28')");
                stm.executeUpdate(str);
                stm.close();
                conn.close();
        } catch (SQLException e) {e.printStackTrace();}
    }
}
```

JDBC – DB Access Classes (Cont.)

- `java.sql.ResultSetMetaData`
  - Provides extra information about (data about data) the ResultSet object
  - `getColumnCount(), getColumnName(column)`

  ```
  ResultSetMetaData md = rs.getMetaData();
  ```
- `java.sql.DatabaseMetaData`
  - Provides extra information about the database for a given connection object
  - What tables exist, what username is being used, is the DB read-only, what are the primary keys for a table, etc.

  ```
  DatabaseMetaData dmd = c.getMetaData();
  ```
  - Lets developers write apps that are DB-independent