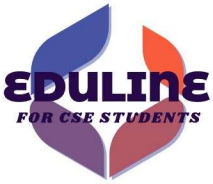
MODULE 5

CHAPTER 2
JDBC



Java DataBase Connectivity (JDBC)

- JDBC stands for Java Database Connectivity, which is a standard Java API for database-independent connectivity between the Java programming language and a wide range of databases.
- The JDBC library includes APIs for each of the tasks mentioned below that are commonly associated with database usage.

- Making a connection to a database
- Creating SQL or MySQL statements
- Executing SQL or MySQL queries in the database
- Viewing & Modifying the resulting records

❖JDBC Architecture

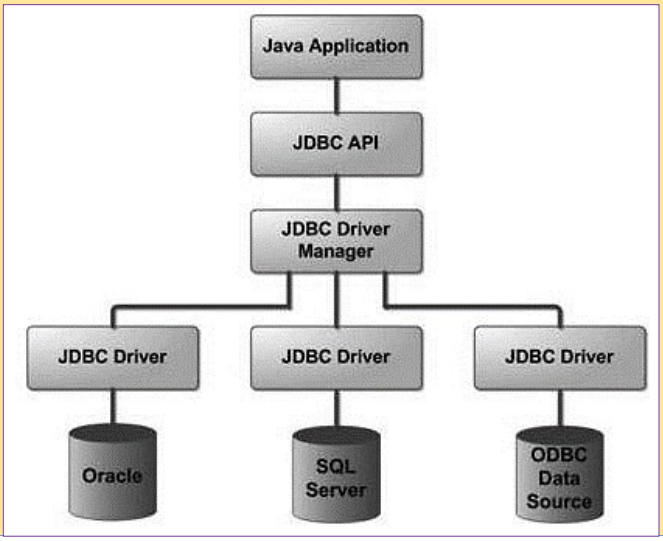
➢JDBC Architecture consists of two layers

- **JDBC API:** This provides the application-to-JDBC Manager connection.
- **JDBC Driver API:** This supports the JDBC Manager-to-Driver Connection.

➢The JDBC API uses a driver manager and database-specific drivers to provide transparent connectivity to heterogeneous databases.

➢The JDBC driver manager ensures that the correct driver is used to access each data source.

- Following is the architectural diagram, which shows the location of the driver manager with respect to the JDBC drivers and the Java application



Java Database Connectivity with 5 Steps

- There are 5 steps to connect any java application with the database using JDBC. These steps are as follows:

1. Register the Driver class
2. Create connection
3. Create statement
4. Execute queries
5. Close connection

- The forName() method is used to register the driver class.
- The getConnection() method of DriverManager class is used to establish connection with the database
- The createStatement() method of Connection interface is used to create statement. The object of statement is responsible to execute queries with the database.
- The executeQuery() method of Statement interface is used to execute queries to the database. This method returns the object of ResultSet that can be used to get all the records of a table.
- By closing connection object statement and ResultSet will be closed automatically. The close() method of Connection interface is used to close the connection.

Java Database Connectivity with MySQL

- To connect Java application with the MySQL database, we need to follow 5 following steps.
- In this example we are using MySql as the database. So we need to know following informations for the mysql database:

1. **Driver class:** The driver class for the mysql database is com.mysql.jdbc.Driver.
2. **Connection URL:** The connection URL for the mysql database is jdbc:mysql://localhost:3306/sonoo where jdbc is the API, mysql is the database, localhost is the server name on which mysql is running, we may also use IP address, 3306 is the port number and sonoo is the database name. We may use any database, in such case, we need to replace the sonoo with our database name.

3. **Username:** The default username for the mysql database is root.
4. **Password:** It is the password given by the user at the time of installing the mysql database. In this example, we are going to use root as the password.

➢ Let's first create a table in the mysql database, but before creating table, we need to create database first.

```
create database sonoo;
use sonoo;
create table emp(id int(10),name varchar(40),age int(3));
```

❖Example to Connect Java Application with mysql database

```
import java.sql.*;
class MysqlCon{
public static void main(String args[]){
try{
Class.forName("com.mysql.jdbc.Driver");
Connection con=DriverManager.getConnection(
"jdbc:mysql://localhost:3306/sonoo","root","root");
//here sonoo is database name, root is username and password
Statement stmt=con.createStatement();
ResultSet rs=stmt.executeQuery("select * from emp");
while(rs.next())
System.out.println(rs.getInt(1)+"  "+rs.getString(2)+"  "+rs.getString(3));
con.close();
}catch(Exception e){ System.out.println(e);}
}
}
```

This example will fetch all the records of emp table.

Creating a sample MySQL database

```
create database SampleDB;
 use SampleDB;
 CREATE TABLE `users` (
   `user_id` int(11) NOT NULL AUTO_INCREMENT,
   `username` varchar(45) NOT NULL,
   `password` varchar(45) NOT NULL,
   `fullname` varchar(45) NOT NULL,
   `email` varchar(45) NOT NULL,
   PRIMARY KEY (`user_id`)
);
```

Connecting to the database

```
String dbURL = "jdbc:mysql://localhost:3306/sampledb";
String username = "root";
String password = "secret";
try {
        Connection conn = DriverManager.getConnection(dbURL, username, password);
      if (conn != null) {
      System.out.println("Connected");
   }
} catch (SQLException ex)
{
   ex.printStackTrace();
}
```


- Once the connection was established, we have a Connection object which can be used to create statements in order to execute SQL queries.
- In the above code, we have to close the connection explicitly after finish working with the database:

```
conn.close();
```

❖**INSERT Statement Example**

Let's write code to insert a new record into the table Users with following details:

username: bill

password: secretpass

fullname: Bill Gates

email: bill.gates@microsoft.com

```
String sql = "INSERT INTO Users (username, password, fullname, email) VALUES (?, ?, ?, ?)";
 PreparedStatement statement = conn.prepareStatement(sql);
 statement.setString(1, "bill");
 statement.setString(2, "secretpass");
 statement.setString(3, "Bill Gates");
 statement.setString(4, "bill.gates@microsoft.com");
 int rowsInserted = statement.executeUpdate();
 if (rowsInserted > 0) {
     System.out.println("A new user was inserted successfully!");
 }
```


❖SELECT Statement Example

```
String sql = "SELECT * FROM Users";

Statement statement = conn.createStatement();
ResultSet result = statement.executeQuery(sql);

int count = 0;

while (result.next()){
    String name = result.getString(2);
    String pass = result.getString(3);
    String fullname = result.getString("fullname");
    String email = result.getString("email");

    String output = "User #%d: %s - %s - %s - %s";
    System.out.println(String.format(output, ++count, name, pass, fullname, email));
}
```

Output

User #1: bill - secretpass - Bill Gates - bill.gates@microsoft.com

❖UPDATE Statement Example

```
String sql = "UPDATE Users SET password=?, fullname=?, email=? WHERE username=?";

PreparedStatement statement = conn.prepareStatement(sql);
statement.setString(1, "123456789");
statement.setString(2, "William Henry Bill Gates");
statement.setString(3, "bill.gates@microsoft.com");
statement.setString(4, "bill");

int rowsUpdated = statement.executeUpdate();
if (rowsUpdated > 0) {
    System.out.println("An existing user was updated successfully!");
}
```


❖DELETE Statement Example

- The following code snippet will delete a record whose username field contains “bill”

```
String sql = "DELETE FROM Users WHERE username=?";

PreparedStatement statement = conn.prepareStatement(sql);
statement.setString(1, "bill");

int rowsDeleted = statement.executeUpdate();
if (rowsDeleted > 0) {
    System.out.println("A user was deleted successfully!");
}
```