JDBC (1/11/22 - 3/12/22)

1. **What is Jar File ?**
    - **Ans**
        - **Jar means Java archieve(compress).**
        - **It is file based on zip file format which is used to compress many files into one single file.**
        - **Jar file needed to import the properties based on the requirement.**
        - **Contents of Jar File**
            - **.java**
                - .java contains source code statement in it.
            - **.class**
                - .class contains byte code statement in it.
            - **.config file**
                - .config contains only configuration data in it.
                - Example for .config
                    1. OS of our system
                    2. about option in our mobile phone.
                    3. Setup file of any software can be example for config file
                - Types of .config File
                    - .xml (Extensible Markup Language)
                        - .xml configures a resource with the help of custom tags or user defined tags
                    - .properties
                        - It is use to provides a set of properties in the form of key and values pair.
                        - where key must be unique(different)
        - NOTE:
            - All the 6000+ inbuilt class of java are in one single Jar file called rt.jar
            - where rt stands for runtime.
            1. How many inbuilt classes present in java ?
            - Ans
                - 6000+
4. **Steps to create Jar File ?**
    - **Ans**

      Step 1:- Right click on Project →select Export option.

      Step 2:- Open Java folder —>select Jar file click on next

Step 3:- Browser for the appropriate location to place the Jar file and name the JAR and click on save and finish.

4. **Step to build the Java path ?**

▼ **Ans**

- **There are 2different ways to build a Java path to import the properties from the jar file namely**

▼ **Externally**

1. **Right click on Project and select Properties option.**
2. **Select Java build path and click on libraries tab.**
3. **Click on add External Jar and Select the Jar file from the appropriate location and click on apply and close.**

▼ **Internally**

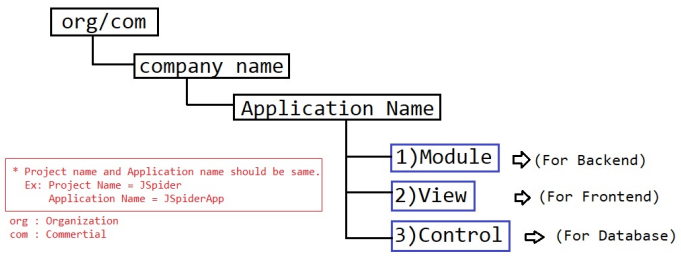
1. **Right click on Project and create a new folder by name called lib.**
2. **Add the respective Jar file into the lib folder.**
3. **Right click on Project and Select Properties option.**
4. **Select Java build path and click on library tab**
5. **Click on add Jar and Select the Jar file from the lib folder of respective project and click on open, apply and close.**

▼ Note

- **It is always good practise to import the properties from a Jar file Internally.**

4. **Standard package / Folder Structure ?**

▼ Ans



- In order to differentiate the folder and packages we nee Standard package structure.

org/com —> Company Name —> Application Name —> Module1, Mpodule2, , , ,etc

- **In case of standard package structure, we use . separator**
- **In case of standard folder structure, we use / as a separator**

org/com —> Company Name —> Application Name —> 1)Module 2)View 3)Control

1) Module is for Back-end.

2) View is for Front-end.

3) Control is for Data-base.

5. **Steps to create basic JDBC project ?**

▼ Ans

Step 1:- Open eclipse in Java perspective.

Step 2:- Open Navigator Mode

Step 3:- Right click within Navigator Mode and create new java project and name it.

Step 4:- Select src folder and control n to create a new package folder structure.

Step 5:- Select Application Name and hit control+N to create the new class

▼ **Note**

- **If we delete bin folder of .class file , we can regeneration it by using Project button + clean button.**

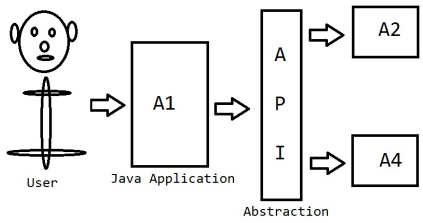
6. **What is Abstractions ?**

▼ Ans

- **Abstractions**
  - **Abstractions means hiding the internal implementation and providing the functionality to the user with the help of Interface is know as Abstraction.**
  - In case of abstraction, the implementation are hidden in order to reduce the complexity of viewing things or to make the implementation simple and neat for the user.
  - The output of the abstraction is loose coupling which can be achieved with the help of Interface.
- **Interface**
  - It is a **media to communicate between User and any Device.**
- **Loose Coupling**
  - Change in the implementation which does not affect the user is know as Loose Coupling.
- **Tight Coupling**
  - Change in the implementation which affect to the user is know as Tight Coupling.
- **Real Time Example for Abstraction is Loose coupling and Tight coupling.**

7. **What is API ?**

▼ **Ans**



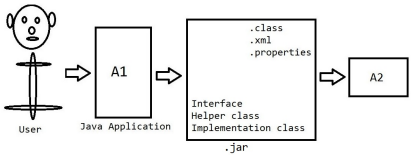
- API Stands for **Application Programing Interface**
- **API is used for "Inter Application Communication" that means one application can communicate with another application with help of an API to achieve loose coupling.**
- **The backbone of API is Abstraction.**
- **The output of Abstraction is loose coupling which can be achieve with help of Interface.**
- **Change in the implementation does not affect the user is know as loose coupling.**

▼ **Example for API**

1. **Apache POI**
2. **Jaxcel**
3. **JDBC**
4. **Servalets etc**

8. **What are the contents of an API ?**

▼ **Ans**



- **The contents of API are Interfaces, helper classes and Implementation class which is given in the form Jar file.**
- **Whenever any API is given in the form Jar file then such API are developed by using Java as a programming language.**

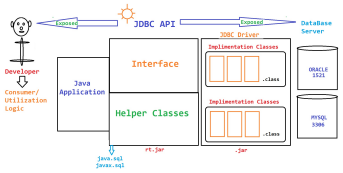
9. **What are the Types of API ?**

▼ Ans

There are two different form of API are present namely :-

▼ **I Form of API**

▼ **II Form of API or (JDBC Architecture)**



- For us to communicate with any database serve and perform any database operation, we need two different components namely JDBC API and JDBC Driver.
- JDBC API is used to achieve loose coupling between Java application and the data the database server
- JDBC Driver helps us to get connected to a particular data base server or vendor.

▼ JDBC API

- **JDBC API was given by SunMrosystem to achieve loose coupling between Java application and database server.**
- **JDBC API contains Interface and Helper class in the form of Jar file.**
- **JDBC API is distributed into 2different packages namely**

1. **java.sql**
2. **javax.sql**

- **Few of the Interface of JDBC API are**

1. **Driver**
2. **Connection**
3. **Statement**
4. **Preparedstatement**
5. **Callablestatement**
6. **ResultSet ,etc.**

- **JDBC API contains only one Helper class in it by name called DriverManager.**

▼ JDBC Driver

- **JDBC Driver is an Implementation of JDBC API.**
- **JDBC Driver contains Implementation classes in form of Jar file.**
- **JDBC Driver is always specific to a particular data base server or vendor.**
- **JDBC Driver are provided by the respective database server or vendor**

9. **What is Consumer and Utilization Logic ?**

▼ Ans

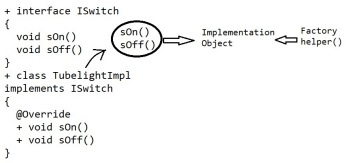
- **It is the most important logic written by the user to access the functionalities from the implementation Object.**
- **The consumer or utilization logic is always written only after knowing the implementation.**

10. **What is Factory or Helper method ?**

▼ Ans

- **It is used to create an implementation object for an Interface.**

▼ **Example**



▼ **Example 1**

```
//Connection Interface
public interface ISwitch
{
  void on();
  void off();
}

//Implementation Class for MY SQL
public class TubeLightImpl implements ISwitch
{
  @Override
  public void on()
  {
    System.out.println("TubeLight ON");
  }
```

```
  @Override
  public void off()
  {
    System.out.println("TubeLight OFF");

  }
}

//Implementation Class for ORACLE
public class LedLightImple implements ISwitch
{
  @Override
  public void on()
  {
    System.out.println("LedLight ON");
  }
  @Override
  public void off()
  {
    System.out.println("LedLight OFF");

  }
}

//DriverManager
public class LightFactory
{
  public static ISwitch getLight(String type)
  {
    if(type.equalsIgnoreCase("tubelight"))
    {
      return new TubeLightImpl();
    }
    else if(type.equalsIgnoreCase("ledlight"))
    {
      return new LedLightImple();
    }
    else
    {
      System.out.println("No Light Found!!");
      return null;
    }
  }
}

//Our class
public class Test
{
  public static void main(String[] args)
  {
    System.out.println("Enter the Type of Light");
    Scanner sc=new Scanner(System.in);
    String type=sc.next();sc.close();

    ISwitch sw=LightFactory.getLight(type);
    if(sw!=null)
    {
      sw.on();
      sw.off();
    }
  }
}

-------------OUTPUT-----------------------
Enter the Type of Light
TubeLight
TubeLight ON
TubeLight OFF

Enter the Type of Light
LedLight
LedLight ON
LedLight OFF
```

```
Enter the Type of Light
FocusLight
No Light Found!!
```

▼ Example 2

```
------------------Interface----------------------
public interface Test
{
  void display();   //abstrct method
}
------------------Implementation Class-----------
public class TestImpl implements Test
{
  @Override
  public void display()
  {
    System.out.println("Hi");
  }
}
------------------Helper Class-------------------
public class Helper
{
  static Test getObject()
  {
    return new TestImpl();
  }
}
------------Utilization Class(Our Class)----------
public class MyClass
{
  public static void main(String[] args)
  {
    Test t1=Helper.getObject();
    t1.display();
  }
}
```

▼ Example 3

```
------------------Interface----------------------
public interface state
{
 void display();
}
------------------Interface----------------------
public interface manu
{
  void setInt();
  void setString();

  state getObject();
}
------------------Implementation Class-----------
public class manuimpl implements manu
{
  @Override
  public void setInt()
  {
    System.out.println("setInt()");
  }
  @Override
  public void setString()
  {
    System.out.println("setString()");
  }
  @Override
  public state getObject()
```

```
  {
    return new stateimpl();
  }
}
------------------Helper Class-------------------
public class helperClass
{
  static manu getObject()
  {
     return new manuimpl() ;
  }
}
------------Utilization Class(Our Class)----------
public class ctest
{
  public static void main(String[] args)
  {
  manu m=helperClass.getObject();
  m.setInt();
  m.setString();
  state s=m.getObject();
  s.display();
  }
}
```

11. **What is URL ?**

    ▼ Ans

    - URL stands for **Uniform Resource Locater**
    - **It is the Path or Address which used to access an Application or Resource. (Web resource)**

12. **What is Port number ?**

    ▼ Ans

    - **Port number is the one which helps us to get connect to a particular server. Port number are provided by the respective servers which are always unique.**
        1. Oracle —> 1521
        2. My SQL —> 3306
        3. MS SQL —> 1433
        4. Derby —> 1527

           ▼ Note:

           - **There are 65,535 ports, from there 1,024 are reserved.**

13. **What is Class Loading ? How many ways can we load the class ?**

    ▼ **Ans**

    - **Class Loading means loading the .class files into the JVM memory is know as Class Loading.**
    - **A class can be generally loaded in 2 difference ways namely :-**
        1. **By calling any of the members of class which can either be constructor, method and blocks.**

           ▼ Example 1

```
+ class Student
{
  + int id;
  + String name;
  +String gender;
```

```
  Student g=new Student();
  g.id=420;
  g.name="Pruthvi";
  g.gender="male";
}
```

▼ Example 2

```
class trainer
{
  void scold()
  {
    System.out.println("J2EE");
  }
}
class Test
{
    public static void main(Strng[] arg)
    {
      trainer G=new trainer();
          G.scold();
    }
}
---------------OUTPUT-------------------------------
J2EE
```

2. **By using static method called forName() method which is declare inside a class by name called Class present in java.lang package.**
   - **Whenever we use forName() method, It throws a checked Exception called ClassNotFoundException.**
   - **forName() method is always returns loaded class based on its fully qualified class name(FQCN).**

   ▼ Syntax

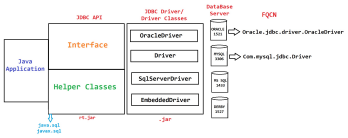
   Class.forName(”FQCN”);

   ▼ Example 1

```
package org.jsp.demoApp
public class trainer
{
  static
      {
          System.out.println("J2EE");
      }
}
package org.jsp.demoApp
public class Test
{
    public static void main(String arg[])
    {
      try
        {
          Class.forName("org.jsp.demoApp.trainer");
        }
      catch (ClassNotFoundException e)
      {
          e.prinstackTrace();
      }
    }
}
```

14. **Different driver classes provided by respective database server or vendor ?**

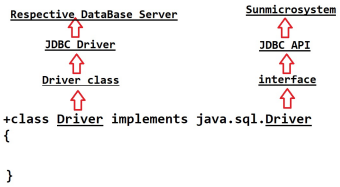
▼ Ans



- **All the Driver classes which are part of JDBC driver are provided by respective database server or vendor in the form of Jar file.**
- **All the driver classes must mandatory implements java.sql.Driver Interface which is a part of JDBC API.**

15. **Why JDBC driver is an implementation of JDBC API ?**

▼ Ans



- **Since all the driver classes which are part of JDBC driver must mandatory implements java.sql.Driver Interface which is a part of JDBC API to achieve loose coupling between Java application and Database server.**

3. **What is JDBC ?**

▼ Ans

- JDBC means → **Java Data Base Connectivity**
- **JDBC is a specification which is given in form of Abstraction API to achieve loose coupling between Java application and Database server.**
  - **Specification means “detailed description”.**

▼ **Advantage JDBC**

- **We can achieve loose coupling between Java application and the database server**
- **Platform Independent**

▼ **Specification of JDBC**

There are 3 different specification are there with respect to JDBC:-

1. **All the Driver classes must contain one static block in it.**
2. **All the Driver classes must mandatorily Implements java.sql.Driver Interface which is the part of JDBC API.**
3. **All the the Driver classes must mandatorily registered with DriverManager using a static method called registerDriver().**

▼ **Example**

```
public class Driver extends NonRegisteringDriver implements Driver
{
  static
  {
    try
    {
      DriverManager.registerDrier(new Driver());
    }
    catch(SQLException e)
    {
      throw new RunTimeException("Can't register Driver!")
    }
  }
}
```

4. **Steps of JDBC ?**

▼ Ans

There are 6 Standard steps are there in JDBC namely.

▼ **Load and Register the Driver. (**where driver refers to driver classes which are given by respective database server**)**

1. **In this step, We need to Load and Register the driver classes provided by respective database servers or vendor.**
2. **The driver classes can be Load and Register in two different way**

▼ Manually

- **In case of manually an object of driver class has to be created and then register with DriverManager by using a static method called registerDriver().**
- It is not a good practice, since it causes tight coupling between Java Application and database.

▼ **Syntax**

```
Driver d=new Driver();
DriverManager.registerDriver(d);
```

▼ By using a static forName() method

- B**y using a static method called forName() method, we have to Load and Register the driver classes which are provided by respective database server or vendor.**

▼ **Syntax**

```
Class.forName("com.mysql.jdbc.Driver");
```

▼ **Example**

```
public class LoadDemo
{
   public static void main(String[] arg)
   {
     try
        {
          Class.forName("com.mysql.jdbc.Driver");
          System.out.println("Driver class is loaded and Registered");
        }
```

```
        catch(ClassNotFoundException e)
        {
          e.printStackTrace();
        }
    }
}
```

Note: If we want to load and register Driver classes, we should set path to .jar file given by respective database.

▼ **Establish a connection with the database server or vendor.**

1. **In this step, We need to Establish the connection between Java application and the database server by using getConnection() method.**
2. **getConnection() is a static factory or helper method which is declared inside DriverManager class which is a part of JDBC API which is used to Create and Return an Implementation object of Connection Interface based on URL. Hence the return type of getConnection() method is Connection Interface.**

   ▼ Syntax

   ```
   java.sql.Connection con=DriverManager.getConnection("URL");
   ```

3. **There are 3 Overloaded Variants of getConnection() methods are present.**
   a. getConnection(String URL);
   b. getConnection(String URL , Properties info);
   c. getConnection(String URL , String user , String passwrd);

   ▼ JDBC URL for MySql

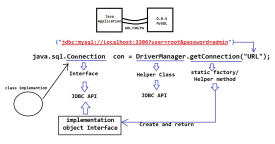
   ```
   URL - "jdbc:mysql://Localhost:3306?user=root&password=admin"
   ```

   ▼ JDBC URL for Oracle

4. **When ever we use either of this overloaded variants of getConection() methods, It throws checked Exception called “SQLException”.**

▼ Note: Interview Point of view

- Whether you should tell “Establish a connection with the database server” or “Establish a connection between Java application and the database server”. But don’t tell “Establish a connection ~~between~~ the database server”.



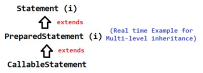
▼ **Create a Statement or Platform.**

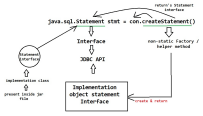
1. **In this step, We need to create a statement or platform in order to execute the SQL queries or statements.**

2. **A Platform can either be created by using Statement, PreparedStatements or CallableStatement Interface which are part of JDBC API.**

▼ Syntax

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





▼ **Execute the SQL queries or SQL statements.**

1. **In this step, We need to Execute different types of SQL queries or statements for that we have 3 different types of methods are there to execute SQL Queries,**
   - **execute() method**
   - **executeUpdate() method**
   - **executeQuery() method**
2. **All these are abstract methods declared inside Statement Interface which is a part of JDBC API and Implementations are provided by respective database server or vendor as a part of JDBC driver. But, All these methods can be used either with respect to Statement or PreparedStatement Interface. Beacuse of IS-A RelationShip.**

   ▼ **execute()**

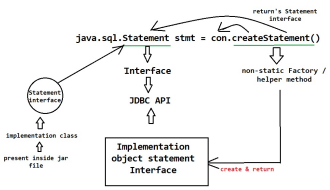
   1. **It is a generic method, Since it is used to execute any type of SQL queries or statement. Hence the return type of execute() method is boolean.**
   2. execute() method returns the boolean true value in case of DQL and boolean false value in case of DDL (or) DML.

   - Method Signature :-

     public boolean execute(”generic SQL queries”);

   - Syntax :-

     bolean tr = stmt.execute(”generic SQL queries”);



   ▼ **executeUpdate()**

1. It is a specialize method which is used to execute DML quires or statements. Hence the return type of executeUpdate() method is integer.
2. When ever we try to execute DDL and DQL queries using executeUpdate() method. It throws a Checked Exception called SQL Exception.

- Method Signature :-

  public int executeUpdate("only DML queries");

- Syntax :-

  int a = stmt.executeUpdate("only DML DQL");

▼ Write a code Insert a Single Record into Database server using Statement Interface ?

- (Before run the code create table in database and Set path to the respective database jar file.)

```
----------------MY SQL----------------------------------
import java.sql.*;
public class Insert_record
{
  public static void main(String[] args)
  {
    Connection con = null;
    Statement stmt = null;
    try
    {
      Class.forName("com.mysql.jdbc.Driver");
      con = DriverManager.getConnection("jdbc:mysql://Localhost:3306?user=root&password=admin");
      stmt = con.createStatement();
      stmt.executeUpdate("INSERT INTO BTM.STUDENT VALUES (1,'ABC',99.99)");
      System.out.println("Data Inserted");
    }
    catch(ClassNotFoundException | SQLException e)
    {
      System.out.println(e);
    }
    finally
    {
      if(stmt!=null)
      {
        try
        {
          stmt.close();
        }
        catch(SQLException e)
        {
          System.out.println(e);
        }
      }
      if(con!=null)
      {
        try
        {
          con.close();
        }
        catch(SQLException e)
        {
          System.out.println(e);
        }
      }
      System.out.println("Resorce are closed");
    }
  }
}
----------------------OUTPUT-------------------------------------
Data Inserted
Resorce are closed
----------------------ORACLE---------------------------------
import java.sql.*;
```

```
public class Test2
{
  public static void main(String[] args)
  {
    Connection con=null;
    Statement stmt=null;
    try
    {
      Class.forName("oracle.jdbc.driver.OracleDriver");
      con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl","scott","tiger");
      stmt=con.createStatement();
      stmt.executeUpdate("INSERT INTO MANU VALUES (1,'JVHH',13.36)");
      System.out.println("Record is Insered");
    }
    catch(ClassNotFoundException|SQLException e)
    {
      e.printStackTrace();
    }
    finally
    {
      if(stmt!=null)
      {
        try
        {
          stmt.close();
        }
        catch (SQLException e)
        {
          System.out.println(e);
        }
      }
      if(con!=null)
      {
        try
        {
          con.close();
        }
        catch (SQLException e)
        {
          System.out.println(e);
        }
      }
    }
  }
}
```

▼ Write a code Update a Record into Database server using Statement Interface ?

- (Before run the code create table in database and Set path to the respective database jar file.)

```
import java.sql.*;
public class Update_record
{
  public static void main(String[] args)
  {
    Connection con = null;
    Statement stmt = null;
    try
    {
      Class.forName("com.mysql.jdbc.Driver");
      con = DriverManager.getConnection("jdbc:mysql://Localhost:3306?user=root&password=admin");
      stmt = con.createStatement();
      stmt.executeUpdate("UPDATE BTM.STUDENT SET NAME='CBA' WHERE ID=1");
      System.out.println("Data Updated");
    }
    catch(ClassNotFoundException | SQLException e)
    {
      System.out.println(e);
    }
    finally
    {
```

```
      if(stmt!=null)
      {
        try
        {
          stmt.close();
        }
        catch(SQLException e)
        {
          System.out.println(e);
        }
      }
      if(con!=null)
      {
        try
        {
          con.close();
        }
        catch(SQLException e)
        {
          System.out.println(e);
        }
      }
      System.out.println("Resorce are closed");
    }
  }
}
----------------output-------------------------------
Data Updated
Resorce are closed
```

▼ Write a code Delete a Record into Database server using Statement Interface ?

- (Before run the code create table in database and Set path to the respective database jar file.)

```
import java.sql.*;
public class Delete_record
{
  public static void main(String[] args)
  {
    Connection con = null;
    Statement stmt = null;
    try
    {
      Class.forName("com.mysql.jdbc.Driver");
      con = DriverManager.getConnection("jdbc:mysql://Localhost:3306?user=root&password=admin");
      stmt = con.createStatement();
      stmt.executeUpdate("DELETE FROM BTM.STUDENT WHERE ID=1");
      System.out.println("Data Deleted");
    }
    catch(ClassNotFoundException | SQLException e)
    {
      System.out.println(e);
    }
    finally
    {
      if(stmt!=null)
      {
        try
        {
          stmt.close();
        }
        catch(SQLException e)
        {
          System.out.println(e);
        }
      }
      if(con!=null)
      {
        try
        {
          con.close();
```

```
        }
        catch(SQLException e)
        {
          System.out.println(e);
        }
      }
      System.out.println("Resorce are closed");
    }
  }
}
--------------------OUTPUT-------------------------
Data Deleted
Resorce are closed
```

▼ Write a code Insert a Multiple Record into Database server using Statement Interface ?

- (Before run the code create table in database and Set path to the respective database jar file.)

```
import java.sql.*;
public class Insert_multiple_record
{
  public static void main(String[] args)
  {
    Connection con = null;
    Statement stmt = null;
    String qry1 = "INSERT INTO BTM.STUDENT VALUES (1,'ABC',99.99)";
    String qry2 = "INSERT INTO BTM.STUDENT VALUES (2,'DEF',99.99)";
    String qry3 = "INSERT INTO BTM.STUDENT VALUES (3,'GHI',99.99)";
    String qry4 = "INSERT INTO BTM.STUDENT VALUES (4,'JKL',99.99)";
    try
    {
      Class.forName("com.mysql.jdbc.Driver");
      con = DriverManager.getConnection("jdbc:mysql://Localhost:3306?user=root&password=admin");
      stmt = con.createStatement();
      stmt.executeUpdate(qry1);
      stmt.executeUpdate(qry2);
      stmt.executeUpdate(qry3);
      stmt.executeUpdate(qry4);
      System.out.println("Multiple Data Inserted");
    }
    catch(ClassNotFoundException | SQLException e)
    {
      System.out.println(e);
    }
    finally
    {
      if(stmt!=null)
      {
        try
        {
          stmt.close();
        }
        catch(SQLException e)
        {
          System.out.println(e);
        }
      }
      if(con!=null)
      {
        try
        {
          con.close();
        }
        catch(SQLException e)
        {
          System.out.println(e);
        }
      }
      System.out.println("Resorce are closed");
    }
  }
```

```
}
----------------OUTPUT-------------------------------
Multiple Data Inserted
Resorce are closed
```

▼ **Process the Resultant data. (optional)**

▼ **Close all the Costly Resources.**

- **Costly resources means resources which makes uses of system properties in the form of stream are known as Costly Resources.**
- **In case of JDBC all the Interface are consider to be costly resources. So, It is always a good practice to close all the costly resources. Since it decrease the performance of an application.**
- **All the costly resource must be closed in finally block using "if" condition to avoid NullPointerException.**
  - Pointing towards an object which is not present. It throws an exception called **Null Pointer Exception**.

▼ **Note:**

- All the Interfaces of JDBC API are considered to be costly resource with respect to JDBC.
- The Interface of JDBC API are Driver, Connection, Statement, PreparedStatement, CallableStatement, etc

5. **Why we want to Load and Register the Driver ?**

▼ Ans

- **To access the functionalities and Implementation classes from the driver class. So that we should Load and Register with driver class.**

6. java.sql.**DriverManager**

▼ Ans

- **It is a Helper class which is part JDBC API and It contains 2 static methods.**
  1. getConnection()
  2. registerDriver()

7. java.sql.**Connection**

▼ Ans

- **It is an Interface which is the part of JDBC API and the Implementations are provided by the respective database server and vendors as a part of JDBC Driver.**
- **createStatement() , preparedStatement() and prepareCall(). All these are abstract factory or helper methods declared inside Connection Interface.**
- **It is always a good practice to close Connection Interface inside finally block. Since, It decrease the performance of an application.**

8. java.sql.**Statement**

▼ Ans

- **It is an Interface, which is a part of JDBC API and Implementations are provided by respective database server or vendor as a part of JDBC Driver**

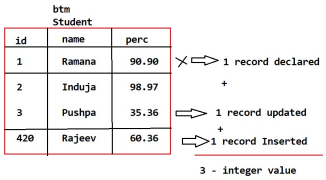
- **createStatement() is an abstract factory or helper method declared inside Connection Interface which is part of JDBC API and Implementation is provided by respective database server or vendor as a part of JDBC Driver which is used to create and return an Implementation object for Statement Interface which is also a part of JDBC API. Hence return type of createStatement() method is Statement Interface.**
- **execute() , executeUpdate() , executeQuery() and getResultSet(). All these are abstract factory or helper methods declared inside Statement Interface.**

9. **What is the Outcome of any executed DML queries or statements ?**

▼ Ans

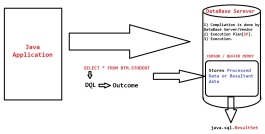
- **The outcome of any executed DML queries is 0 to N integer value which gives the total number of records are affected in the database server. When outcome is '0' then there is no records present or no change in the database.**
- String qry1 = "INSERT INTO BTM.STUDENT VALUES (420,'RAJEEV',60.36)");
- String qry2 = "UPDATE BTM.STUDENT SET NAME='MOHAN' WHERE ID=3;
- String qry3 = "DELETE FROM BTM.STUDENT WHERE ID=1;
- 0 ⇒

  String qry = "DELETE FROM BTM.STUDENT WHERE ID=5;



10. **Fetching the data from Database Server using ResultSet Interface ?**

▼ Ans



- **Whenever we execute DQL queries or statements we may get result which is referred as Processed or Resultant data.**
- **The processed or resultant data which is stored in the cursor or buffer memory which can be fetched with the help of ResultSet Interface which is a part of JDBC API.**
- **The structure of cursor or buffer memory may differ from the structure of database.**

▼ Note:

- **column name is also known as column label.**
- **column number is also known as column index**

11. **java.sql.ResultSet**

▼ Ans

- **It is an Interface which is a part of JDBC API and Implementations are provided by respective to database servers or vendors as a part of JDBC driver.**
- getXXX() is an overloaded abstract method declared inside ResultSet Interface which is the part of JDBC API and Implementations are provided by respective database server or vendor which are used to fetch the processed or resultant data from cursor or buffer memory. Hence the return type of getXXX() methods are respective datatypes.

▼ **There are 2 overloaded variants of getXXX() are present namely.**

1. **public XXX getXXX(int column number)**
2. **public XXX getXXX(String column name)**

- **If datatype is integer:**
  - public int getInt(int column number)
  - public int getInt(String column name)
- **If datatype is String :**
  - public String getString(int column number)
  - public String getString(String column name)
- **If datatype is double :**
  - public double getDouble(int column number)
  - public double getDouble(String column name)

- **By default ResultSet Interface does not pointing to any record in the cursor / buffer memory.**

▼ **next()**

- **next() method is used to check whether the next record is present in the cursor memory or not. It returns a boolean value called true or false but not the records.**
- **Whenever it can be used wherever there are minimum number of records are present in the cursor of buffer memory**

  Method Signature : - public boolean next()

▼ **absolute()**

- **absolute() method is used to check whether a particular record is present in the cursor memory or not based on a parameter called Integer Row Number and it returns a boolean value called true or false but not the record.**
- **absolute() Method can be used whenever there are a number of records present in the cursor memory or buffer memory.**

  Method Signature : - public boolean absolute(int rownumber)

▼ **executeQuery()**

- **It is a specialised method which is used to execute only DQL queries or statements.**
- **The outcome of DQL is processed or resultant data which is stored in the cursor or buffer memory which can be fetch with the help of getXXX() methods declared inside ResulSet Interface which is a part of JDBC API. Hence the return type of executeQuery() is ResultSet Interace.**

Method Signature : - public ResultSet execueQuery(”Only DQL”)

- **Whenever we try to execute DDL or DML query using executeQuery() method it throws Checked Exception called SQLException.**

12. **In how many ways can we create an implementation object for ResultSet Interface ?**

▼ Ans

- **There are 2 different ways in which we can create an implementation object of ResultSet Interface.**

▼ 1). By using **getResultSet**() method declared inside Statement Interface.

```
public val = stmt.execute(”DQL Query”);
if(val)
{
    java.sql.ResultSet rs=stmt.getResultSet();
}
```

▼ 2). By using **executeQuery()** method declared inside Statement Interface.

```
ResultSet rs=stmt.executeQuery(”Only DQL”);
```

13. **What is Processed or Resultant data ?**

▼ Ans

- **Whenever we execute DQL queries or statements we may get the result which is referred as Processed or Resultant data.**

14. **What is Cursor or Buffer memory ?**

▼ Ans

- **Cursor or Buffer memory is a storage area which stores the Processed or Resultant data whenever we execute DQL queries or statements. The structure of Cursor or Buffer memory may differ from the structure of database.**

15. **How can we fetch the Processed or Resultant data ?**

▼ Ans

- **The processed or resultant data which is stored in the cursor or buffer memory which can be fetch with the help of getXXX() methods declraed inside ResultSet Interface which is a part of JDBC API.**

16. **Can we deals with multiple records by using Statement Interface?**

▼ Ans

- **Yes, But it is not good practice to use Statement Interface for dealing with multiple records. Because, Compilation takes place each time along with the execution, the performance of application decreases. Hence, We use PreparedStatement Interface which increases the performance of an application. Because it supports the concept of Placeholder to take dynamic values at the runtime by the user and also it’s supports the concept of “Execute Plan” that means Compile once and execute many times.**
  - **Statement stmt = con.createStatement();**
  - **stmt.execute(qry) —> [compilation + Execution]**

17. **In General, Is Statement Interface is faster in performance or PreparedStatement Interface is faster ?**

▼ **Ans**

In General, PreparedStatement Interface is faster in performance compared to Statement Interface for two important reasons namely.

1. **Since PreapredStatement Interface supports the concept of Palceholder to take dynamic values at the runtime by the user.**
2. **Since PreparedStatement Interface supports the concept of Compile once and Execute many times (Execute plan).**

18. **What is Placeholder ?**

▼ **Ans**

- **Placeholder is a parameter which takes the dynamic values at the runtime by the User.**
- **In case of JDBC placeholder is represented as ?**

▼ **declaration of placeholder**

- String Qry1="INSERT INTO BTM.STUDENT VALUES(?,?,?)";
- String Qry3="UPDATE BTM.STUDENT SET NAME=? WHERE ID=?";
- String Qry4="DELETE FROM BTM.STUDENT WHERE ID=?";
- String Qry4="SELECT * FROM BTM.STUDENT WHERE ID=?";

19. **What are the Rules to set the data for a Placeholder ?**

▼ Ans

There are 2 different rules are there in order to set data for a Placeholder. They are:-

1. **We have to set the data for a Placeholder before the Execution of SQL Queries by using setXXX() method.**
2. **The number of data must exactly match the number of Placeholders.**

20. java.sql.**PreparedStatement Interface ?**

▼ Ans

- **It is an Interface which is a part of JDBC API and the implementations are provided by the respective database server or vendor as a part of JDBC driver.**
- **PreparedStatement Interface is a sub Interface of statement Interface.**
- **PreparedStatement Interface supports the concept of Placeholder to take dynamic values at the runtime by the user.**
- **PreparedStatement Interface supports the concept of Compile once and Execute many times (Execute plan).**
- **In case of PreparedStatement Interface, the query is passed at the time of implementation object creation of PreparedStatement Interface but not in the execute() method. Hence PreparedStatement Interface are also kown as Pre-Compiled statement.**

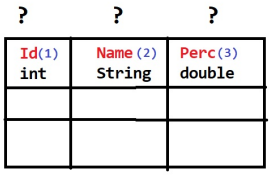
- **prepareStatement() is an abstract factory or helper method declared inside Connection Interface which is a part of JDBC API and Implementation are provided by respective database server or vendor as part of JDBC driver which is used to Create and Return an Implementation object of PreparedStatement Interface which is also a part of JDBC API. Hence return type of prepareStatement() method is PreparedStatement Interface.**
- **setXXX() is an abstract method declared inside PreparedStatement Interface which is a part JDBC API and implementations are provided by respective database server or vendor as a part JDBC driver which is used to set data for Placeholders. Hence return type for setXXX() method is void.**

▼ **Syntax**

- **public void setXXX(int placeholdernumber/placeholderIndex , XXX data)**
- **public void setInt(int placeholdernumber/placeholderIndex , int data)**
- **public void setString(int placeholdernumber/placeholderIndex , String data)**
- **public void setDouble(int placeholdernumber/placeholderIndex , double data)**

▼ **Example**

- setInt(1,420)
- setString(2,"Vijay")
- setDouble(3,72.60)



21. **What are return type for getXXX() method and setXXX() methods ?**

▼ Ans

- **By default return type getXXX() methods are respective datatypes, where as the return type of setXXX() method is void.**

22. **Why are PreparedStatement Interface is also known as Pre-Complied statement ?**

▼ Ans

- **Since the query is already compiled at the time of implementation object creation of PreparedStatement Interface. Hence the name called Pre-Compiled statement.**

23. **What is Role of Placeholder in case of PreparedStatement and CallableStatement Interface ?**

▼ Ans

- **In case of PreparedStatement Interface Placeholder is used to take the dynamic values at the runtime by the User.**
- **In case of CallableStatement Interface Placeholder is used to take the values for IN and OUT parameter.**

String qry="SELECT USERNAME FROM BTM.USER WHERE NAME=? AND PASSWORD=?"

24. Whenever the number of data exceeds the number of placeholder. What is the output ?

▼ Ans

- OUTPUT: SQLException

String qry=”INSERT INTO BTM.STUDENT VALUES (?,?,?)”

pstmt.setInt(1,10);

pstmt.setString(2,”Manu”);

pstmt.setString(3,”82.25”);

pstmt.setString(4,”96.25”);

25. Whenever the number of data exceeds the number of placeholder. What is the output ?

▼ Ans

- OUTPUT: It updates lastly Entered value. means 96.25.

String qry=”INSERT INTO BTM.STUDENT VALUES (?,?,?)”

pstmt.setInt(1,10);

pstmt.setString(2,”Manu”);

pstmt.setString(3,”82.25”);

pstmt.setString(3,”96.25”);

26. **What are the different types of Driver with respect to JDBC ?**

▼ Ans

- **There are 4 different types of JDBC Driver are available.**
    1. Type-1 : **JDBC-ODBC Bridge.**
    2. Type-2 : **Native API Driver.**
    3. Type-3 : **Network Protocol Driver** (Middleware Driver)
    4. Type-4 : **Database Protocol Driver** (pure Java Driver or Thin Driver)
- We are using Type-4 (Database Protocol Driver (pure Java Driver or Thin Driver))

27. **Write a code to Insert multiple records into the database server by using PreparedStatement Interface along with Placeholder ?**

▼ Ans

▼ Program 1

```
import java.sql.*;
public class InsertMultipleRecords
{
  public static void main(String[] args)
  {
    Connection con = null;
    PreparedStatement pstmt = null;
    String qry = "INSERT INTO BTM.STUDENT VALUES(?,?,?)";//DML
    try
    {
      Class.forName("com.mysql.jdbc.Driver");
      System.out.println("Driver Class Loaded and Register");
      con = DriverManager.getConnection("jdbc:mysql://localhost:3306?user=root&password=admin");
      System.out.println("Connection Establish with DBS");
      pstmt = con.prepareStatement(qry);
      System.out.println("Platform Created");
      //Set the data for the placeholder before Execution
```

```
      pstmt.setInt(1, 1);
      pstmt.setString(2, "Vjay");
      pstmt.setDouble(3, 35.96);
      //Execute the SQL query
      pstmt.executeUpdate();

      pstmt.setInt(1, 2);
      pstmt.setString(2, "Ajay");
      pstmt.setDouble(3, 56.96);
      //Execute the SQL query
      pstmt.executeUpdate();

      pstmt.setInt(1, 3);
      pstmt.setString(2, "meghu");
      pstmt.setDouble(3, 99.96);
      //Execute the SQL query
      pstmt.executeUpdate();

      System.out.println("Records are Inserted Successfully");
    }
    catch(ClassNotFoundException | SQLException e)
    {
      e.printStackTrace();
    }
    finally
    {
      if (pstmt!=null)
      {
        try
        {
          pstmt.close();
        }
        catch (SQLException e)
        {
          e.printStackTrace();
        }
      }
      if (con!=null)
      {
        try
        {
          con.close();
        }
        catch (SQLException e)
        {
          e.printStackTrace();
        }
      }
    }
  }
}
---------------OUTPUT-------------------------
Driver Class Loaded and Register
Connection Establish with DBS
Platform Created
```

▼ Program 2 (Using Scanner)

```
import java.sql.*;
import java.util.*;
public class InsertMultipleRecords
{
  public static void main(String[] args)
  {
    Scanner sc=new Scanner(System.in);
    Connection con = null;
    PreparedStatement pstmt = null;
    String qry = "INSERT INTO BTM.STUDENT VALUES(?,?,?)";//DML
    try
    {
      Class.forName("com.mysql.jdbc.Driver");
```

```
      System.out.println("Driver Class Loaded and Register");
      con = DriverManager.getConnection("jdbc:mysql://localhost:3306?user=root&password=admin");
      System.out.println("Connection Establish with DBS");
      pstmt = con.prepareStatement(qry);
      System.out.println("Platform Created");
      //Set the data for the placeholder before Execution
      while (true)
      {
        System.out.println("Enter ID :=");
        int ID=sc.nextInt();
        pstmt.setInt(1, ID);

        System.out.println("Enter the Name :=");
        String Name=sc.next();
        pstmt.setString(2, Name);

        System.out.println("Enter the Percentage :=");
        double Perc=sc.nextDouble();
        pstmt.setDouble(3, Perc);

        //Execute the SQL query
        pstmt.executeUpdate();

        System.out.println("Record Inserted");
        System.out.println("add more Data ??? Yes or No");
        String s=sc.next();
        if(s.equalsIgnoreCase("no"))
          break;
      }
      sc.close();
      System.out.println("Scanner is Cloased!!!");
      System.out.println("OK Prends Boiii");
    }
    catch(ClassNotFoundException | SQLException e)
    {
      e.printStackTrace();
    }
    finally
    {
      if (pstmt!=null)
      {
        try
        {
          pstmt.close();
        }
        catch (Exception e)
        {
          e.printStackTrace();
        }
      }
      if (con!=null)
      {
        try
        {
          con.close();
        }
        catch (Exception e)
        {
          e.printStackTrace();
        }
      }
    }
  }
}
-----------------OUTPUT-------------------------------
Driver Class Loaded and Register
Connection Establish with DBS
Platform Created
Enter ID :=
4
Enter the Name :=
diamond
Enter the Percentage :=
99.99
```

```
Record Inserted
add more Data ??? Yes or No
yes
Enter ID :=
5
Enter the Name :=
Madakari
Enter the Percentage :=
90.36
Record Inserted
add more Data ??? Yes or No
no
Scanner is Cloased!!!
OK Prends Boiii
```

28. **Write a code to Fetch all the records from cursor or buffer memory by using Resultset Interface ?**

▼ Ans

```
import java.sql.*;
public class FetchDemo
{
  public static void main(String[] args)
  {
    Connection con = null;
    PreparedStatement pstmt = null;
    ResultSet rs = null;
    String qry = "SELECT * FROM btm.student";//DQL

    try
    {
      Class.forName("com.mysql.jdbc.Driver");
      con = DriverManager.getConnection("jdbc:mysql://localhost:3306?user=root&password=admin");
      pstmt = con.prepareStatement(qry);
      rs = pstmt.executeQuery();
      while(rs.next())
      {

        int id = rs.getInt(1);
        String name = rs.getString("NAME");
        double perc = rs.getDouble(3);
        System.out.println("ID "+id);
        System.out.println("Name "+name);
        System.out.println("Percenetage "+perc);
        System.out.println("----------------------");
      }
    }
    catch(ClassNotFoundException | SQLException e)
    {
      e.printStackTrace();
    }
    finally
    {
      if(rs!=null)
      {
        try
        {
          rs.close();
        }
        catch (SQLException e)
        {
          e.printStackTrace();
        }
      }
      if(pstmt!=null)
      {
        try
        {
          pstmt.close();
        }
        catch (SQLException e)
```

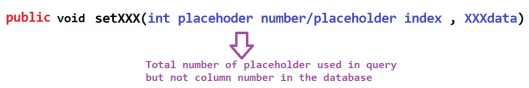
```
        {
          e.printStackTrace();
        }
      }
      if(con!=null)
      {
        try
        {
          con.close();
        }
        catch (SQLException e)
        {
          e.printStackTrace();
        }
      }
    }
  }
}
--------------------OUTPUT------------------
ID 1
Name Manu
Percenetage 35.35
-----------------------
ID 2
Name Megharaj
Percenetage 80.99
-----------------------
ID 3
Name Anand
Percenetage 90.99
-----------------------
ID 4
Name diamond
Percenetage 99.99
-----------------------
ID 5
Name Madakari
Percenetage 90.36
-----------------------
```

29. **Write a code to Fetch particular record from cursor or buffer memory where Id=placeholder ?**

▼ Ans

```
import java.sql.*;
import java.util.Scanner;
public class FetchParticularrRecord2
{
  public static void main(String[] args)
  {
    Connection con = null;
    PreparedStatement pstmt = null;
    ResultSet rs = null;
    String qry = "SELECT * FROM BTM.STUDENT WHERE ID=?";//DQL
    Scanner sc=new Scanner(System.in);
    System.out.println("Enter ID??");
    int Id=sc.nextInt();sc.close();
    try
    {
      Class.forName("com.mysql.jdbc.Driver");
      con = DriverManager.getConnection("jdbc:mysql://localhost:3306?user=root&password=admin");
      pstmt = con.prepareStatement(qry);
      //Set the data for the placeholder before Execution
      pstmt.setInt(1, Id);
      //Execute SQL Query
      rs=pstmt.executeQuery();
      //Check For Record in cursor or Buffer Memory
      if(rs.next())
      {
        String Name=rs.getString(2);
        double Perc=rs.getDouble(3);
```

```
        System.out.println("Name="+Name+" Perc="+Perc);
        System.out.println("Data Fetched");
      }
      else
      {
        System.err.println("No dat Found Name:= "+Id);
      }
    }
    catch(ClassNotFoundException | SQLException e)
    {
      e.printStackTrace();
    }
    finally
    {
      if (rs!=null)
      {
        try
        {
          rs.close();
        }
        catch (SQLException e)
        {
          e.printStackTrace();
        }
      }
      if (pstmt!=null)
      {
        try
        {
          pstmt.close();
        }
        catch (SQLException e)
        {
          e.printStackTrace();
        }
      }
      if (con!=null)
      {
        try
        {
          con.close();
        }
        catch (SQLException e)
        {
          e.printStackTrace();
        }
      }
    }
  }
}
----------OUTPUT----------------------------------------
Enter ID??
1
Name=Manu Perc=35.35
Data Fetched
```



30. **Write a code to Fetch particular record from cursor or buffer memory where name=placeholder ?**

▼ Ans

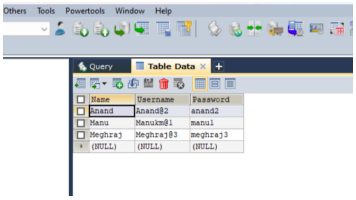
```
import java.sql.*;
import java.util.*;
public class FetchParticularRecord
```

```
{
  public static void main(String[] args)
  {
    Connection con = null;
    PreparedStatement pstmt = null;
    ResultSet rs = null;
    String qry = "SELECT * FROM BTM.STUDENT WHERE NAME=?";//DQL
    Scanner sc=new Scanner(System.in);
    System.out.println("Enter Name??");
    String Name=sc.next();sc.close();
    try
    {
      Class.forName("com.mysql.jdbc.Driver");
      con = DriverManager.getConnection("jdbc:mysql://localhost:3306?user=root&password=admin");
      pstmt = con.prepareStatement(qry);
      //Set the data for the placeholder before Execution
      pstmt.setString(1, Name);
      //Execute SQL Query
      rs=pstmt.executeQuery();
      //Check For Record in cursor or Buffer Memory
      if(rs.next())
      {
        int Id=rs.getInt(1);
        double Perc=rs.getDouble(3);
        System.out.println("Id="+Id+" Perc="+Perc);
        System.out.println("Data Fetched");
      }
      else
      {
        System.err.println("No dat Found Name:= "+Name);
      }
    }
    catch(ClassNotFoundException | SQLException e)
    {
      e.printStackTrace();
    }
    finally
    {
      if (rs!=null)
      {
        try
        {
          rs.close();
        }
        catch (SQLException e)
        {
          e.printStackTrace();
        }
      }
      if (pstmt!=null)
      {
        try
        {
          pstmt.close();
        }
        catch (SQLException e)
        {
          e.printStackTrace();
        }
      }
      if (con!=null)
      {
        try
        {
          con.close();
        }
        catch (SQLException e)
        {
          e.printStackTrace();
        }
      }
    }
  }
}
```

```
----------OUTPUT----------------------------------------
Enter Name??
manu
Id=1 Perc=35.35
Data Fetched
```

31. **Code Login Validation using standard steps of JDBC ?**

▼ Ans




```
import java.util.*;
import java.sql.*;
public class LoginValidation
{
  public static void main(String[] args)
  {
    Connection con=null;
    PreparedStatement pstmt=null;
    ResultSet rs=null;
    String Qry="SELECT USERNAME FROM BTM.USER WHERE NAME=? AND PASSWORD=?";
    Scanner sc=new Scanner(System.in);
    System.out.println("Enter Name:= ");
    String Name=sc.next();
    System.out.println("Enter password:= ");
    String password=sc.next();
    sc.close();
    try
    {
      Class.forName("com.mysql.jdbc.Driver");
      con = DriverManager.getConnection("jdbc:mysql://localhost:3306?user=root&password=admin");
      pstmt = con.prepareStatement(Qry);
      pstmt.setString(1,Name);
      pstmt.setString(2,password);
      rs=pstmt.executeQuery();
      if(rs.next())
      {
        String Username=rs.getString(1);
        System.out.println("Welcome "+Username);
      }
      else
      {
        System.err.println("Ivalid Name/password");
      }
    }
    catch(ClassNotFoundException | SQLException e)
    {
      e.printStackTrace();
    }
    finally
    {
      if (rs!=null)
      {
        try
        {
          rs.close();
        }
        catch (SQLException e)
        {
          e.printStackTrace();
        }
```

```
      }
      if (pstmt!=null)
      {
        try
        {
          pstmt.close();
        }
        catch (SQLException e)
        {
          e.printStackTrace();
        }
      }
      if (con!=null)
      {
        try
        {
          con.close();
        }
        catch (SQLException e)
        {
          e.printStackTrace();
        }
      }
    }
  }
}
----------------OUTPUT(1)-------------------------------------
Enter Name:=
manu
Enter password:=
manu1
Welcome Manukm@1
----------------OUTPUT(2)------------------------------
Enter Name:=
manu
Enter password:=
manu2
Ivalid Name/password
```

32. **All JDBC Operation Using Dao (Data Access Object) ?**

▼ **Ans**

▼ **org.jsp.dto**

▼ **Student.java**

```
package org.jsp.dto;

public class Student {
  private int id;
  private String name, dept;
  private double perc;

  public int getId() {
    return id;
  }

  public void setId(int id) {
    this.id = id;
  }

  public String getName() {
    return name;
  }

  public void setName(String name) {
    this.name = name;
  }

  public String getDept() {
```

```
    return dept;
  }

  public void setDept(String dept) {
    this.dept = dept;
  }

  public double getPerc() {
    return perc;
  }

  public void setPerc(double perc) {
    this.perc = perc;
  }
}
```

▼ **org.jsp.dao**

▼ **StudentDao.java**

```
package org.jsp.dao;

import java.sql.*;
import java.util.*;

import org.jsp.dto.Student;

public class StudentDao {
  static Connection connection;
  PreparedStatement preparedStatement;
  public static final String DRIVER = "com.mysql.jdbc.Driver";
  public static final String URL = "jdbc:mysql://localhost:3306/btm";
  public static final String USER = "root";
  public static final String PASSWORD = "admin";
  {
    try {
      Class.forName(DRIVER);
      connection = DriverManager.getConnection(URL, USER, PASSWORD);
    } catch (ClassNotFoundException | SQLException e) {
      e.printStackTrace();
    }
  }

  public void saveStudent(Student student) {
    try {
      preparedStatement = connection.prepareStatement("insert into student values(?,?,?,?)");
      preparedStatement.setInt(1, student.getId());
      preparedStatement.setString(2, student.getName());
      preparedStatement.setString(3, student.getDept());
      preparedStatement.setDouble(4, student.getPerc());
      preparedStatement.executeUpdate();
      System.out.println("Resource opened");
    } catch (SQLException e) {
      e.printStackTrace();
    } finally {
      closeResources(connection, preparedStatement, null);
      System.out.println("Resource Closed");
    }
  }

  public void updateStudent(Student student) {
    try {
      preparedStatement = connection.prepareStatement("update student set name=?, dept=?, perc=? where id=?");
      preparedStatement.setInt(4, student.getId());
      preparedStatement.setString(1, student.getName());
      preparedStatement.setString(2, student.getDept());
      preparedStatement.setDouble(3, student.getPerc());
      preparedStatement.executeUpdate();
      System.out.println("Resource opened");
    } catch (SQLException e) {
      e.printStackTrace();
```

```
    } finally {
      closeResources(connection, preparedStatement, null);
      System.out.println("Resource Closed");
    }
  }

  public void deleteStudent(int id) {
    String query = "delete from student where id=?";
    try {
      preparedStatement = connection.prepareStatement(query);
      preparedStatement.setInt(1, id);
      preparedStatement.executeUpdate();
      System.out.println("Resource opened");
    } catch (SQLException e) {
      e.printStackTrace();
    } finally {
      closeResources(connection, preparedStatement, null);
      System.out.println("Resource Closed");
    }
  }

  public Student getStudent(int id) {
    String query = "select * from student where id=?";
    Student student = new Student();
    ResultSet resultSet = null;
    try {
      preparedStatement = connection.prepareStatement(query);
      preparedStatement.setInt(1, id);
      resultSet = preparedStatement.executeQuery();
      if (resultSet.next()) {
        student.setId(resultSet.getInt(1));
        student.setName(resultSet.getString(2));
        student.setDept(resultSet.getString(3));
        student.setPerc(resultSet.getDouble(4));
      } else
        student = null;
      System.out.println("Resource opened");
    } catch (SQLException e) {
      e.printStackTrace();
    } finally {
      closeResources(connection, preparedStatement, resultSet);
      System.out.println("Resource Closed");
    }
    return student;
  }

  public List<Student> getAllStudent() {
    String query = "select * from student";
    List<Student> students = new ArrayList<Student>();
    ResultSet resultSet = null;
    try {

      preparedStatement = connection.prepareStatement(query);
      resultSet = preparedStatement.executeQuery();
      while (resultSet.next()) {
        Student student = new Student();
        student.setId(resultSet.getInt(1));
        student.setName(resultSet.getString(2));
        student.setDept(resultSet.getString(3));
        student.setPerc(resultSet.getDouble(4));
        students.add(student);
      }
      System.out.println("Resource opened");
    } catch (SQLException e) {
      e.printStackTrace();
    } finally {
      closeResources(connection, preparedStatement, resultSet);
      System.out.println("Resource Closed");
    }
    return students;
  }

  public void closeResources(Connection con, PreparedStatement pst, ResultSet rs) {
    try {
```

```
      if (con != null)
        con.close();
      if (pst != null)
        pst.close();
      if (rs != null)
        rs.close();
    } catch (SQLException e) {
      e.printStackTrace();
    }
  }
}
```

▼ org.jsp.controller

▼ SaveStudent.java

```
package org.jsp.controller;

import java.util.Scanner;
import org.jsp.dao.StudentDao;
import org.jsp.dto.Student;

public class SaveStudent {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    System.out.println("Enter the Student ID to Save the Data : ");
    int id = sc.nextInt();
    System.out.println("Enter the Student Name : ");
    String name = sc.next();
    System.out.println("Enter the Student Department : ");
    String dept = sc.next();
    System.out.println("Enter the Student Percentage : ");
    double per = sc.nextDouble();
    sc.close();

    Student student = new Student();
    student.setId(id);
    student.setName(name);
    student.setDept(dept);
    student.setPerc(per);
    StudentDao dao = new StudentDao();
    dao.saveStudent(student);
    System.out.println("Record is Saved");
  }
}
```

▼ UpdateStudent.java

```
package org.jsp.controller;

import java.util.Scanner;
import org.jsp.dao.StudentDao;
import org.jsp.dto.Student;

public class UpdateStudent {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    System.out.println("Enter the Student ID to Save the Data : ");
    int id = sc.nextInt();
    System.out.println("Enter the Student Name : ");
    String name = sc.next();
    System.out.println("Enter the Student Department : ");
    String dept = sc.next();
    System.out.println("Enter the Student Percentage : ");
    double per = sc.nextDouble();
    sc.close();

    Student student = new Student();
```

```
    student.setId(id);
    student.setName(name);
    student.setDept(dept);
    student.setPerc(per);
    StudentDao dao = new StudentDao();
    dao.updateStudent(student);
    System.out.println("Record is Updated");
  }
}
```

▼ GetStudentByID.java

```
package org.jsp.controller;

import java.util.Scanner;

import org.jsp.dao.StudentDao;
import org.jsp.dto.Student;

public class GetStudentByID {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    System.out.println("Enter the Student ID to Fetch the Details : ");
    int id = sc.nextInt();
    sc.close();

    StudentDao studentDao = new StudentDao();
    Student student = studentDao.getStudent(id);
    if (student != null) {
      System.out.println("Student ID : " + student.getId());
      System.out.println("Student Name : " + student.getName());
      System.out.println("Student Dept : " + student.getDept());
      System.out.println("Student Perc : " + student.getPerc());
      System.out.println("---------------------------------");
    } else {
      System.out.println("Invalid ID");
    }
  }
}
```

▼ GetAllStudent.java

```
package org.jsp.controller;

import java.util.List;
import org.jsp.dao.StudentDao;
import org.jsp.dto.Student;

public class GetAllStudent {
  public static void main(String[] args) {
    StudentDao studentDao = new StudentDao();
    List<Student> students = studentDao.getAllStudent();
    for (Student s : students) {
      System.out.println("Student ID : " + s.getId());
      System.out.println("Student Name : " + s.getName());
      System.out.println("Student Dept : " + s.getDept());
      System.out.println("Student Perc : " + s.getPerc());
      System.out.println("---------------------------------");
    }
  }
}
```

▼ DeleteStudent.java

```
package org.jsp.controller;
```

```
import java.util.Scanner;
import org.jsp.dao.StudentDao;
import org.jsp.dto.Student;

public class DeleteStudent {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    System.out.println("Enter the Student ID to Delete the Details : ");
    int id = sc.nextInt();
    sc.close();

    StudentDao studentDao = new StudentDao();
    Student student = studentDao.getStudent(id);
    if (student != null) {
      new StudentDao().deleteStudent(id);
      System.out.println("Student ID=" + id + " is Deleted");
    } else {
      System.out.println("Student ID is not Found !");
    }
  }
}
```

▼ **TableCreationQuery(Oracle)**

```
CREATE TABLE  "STUDENT"
   ("ID" INT(10),
  "NAME" VARCHAR(40) NOT NULL,
  "DEPT" VARCHAR(40) NOT NULL,
  "PERC" DECIMAL(4,2) NOT NULL,
   CONSTRAINT "ID_PK" PRIMARY KEY ("ID")
   )
/
```

▼ **TableCreationQuery(MySQL)**

```
CREATE TABLE  "STUDENT"
   ("ID" NUMBER(10),
  "NAME" VARCHAR2(40) NOT NULL,
  "DEPT" VARCHAR2(40) NOT NULL,
  "PERC" NUMBER(4,2) NOT NULL,
   CONSTRAINT "ID_PK" PRIMARY KEY ("ID")
   )
/
```

---

1. Create Stored Procedure ?

▼ Ans

▼ How to open MySQL in CMD and Select the Data and Execute Queries

1. Type → “mysql -u root -p”
2. Type → “admin”
3. Type → “show databases”
4. Type → “use btm”
5. Type → “show tables”
6. Type → “select * from student”

▼ How to create procedure

1. Type → “delimiter /” (For changing termination symbol to /)

2. Type → “create procedure proc99()”
3. Type → “begin”
4. Type → “select * from student;”
5. Type → “end”
6. Type → “/” (for termination)

▼ How to see procedure

- Go to MySQL Yog and See

▼ How to call procedure

1. Type → “”

▼ How to call procedure through JDBC

2. CallableStatement Usage?

▼ Ans

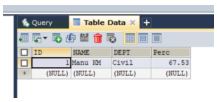
- CallableStatement interface is used to call the **stored procedures and functions**.
- We can have business logic on the database by the use of stored procedures and functions that will make the performance better because these are precompiled.
- Suppose you need the get the age of the employee based on the date of birth, you may create a function that receives date as the input and returns age of the employee as the output.

▼ Difference between **stored procedures and functions**

▼ Create stored procedure in CMD

```
C:\Users\manuk>mysql -u rot -p
Enter password: *****
ERROR 1045 (28000): Access denied for use

C:\Users\manuk>mysql -u root -p
Enter password: *****
Welcome to the MySQL monitor.  Commands e
Your MySQL connection id is 17
Server version: 5.5.27 MySQL Community Se

Copyright (c) 2000, 2011, Oracle and/or i

Oracle is a registered trademark of Oracl
affiliates. Other names may be trademarks
owners.

Type 'help;' or '\h' for help. Type '\c'

mysql> use btm;
Database changed
mysql> delimiter /
mysql> create procedure manu123()
    -> begin
    -> select * from student;
    -> end
    -> /
Query OK, 0 rows affected (0.00 sec)
```

▼ Student table from MySQL



▼ JDBC Code Using CallableStatement Interface

```
import java.sql.*;
public class Fetch
{
  public static void main(String[] args)
  {
    Connection con = null;
    CallableStatement cstmt = null;
    ResultSet rs = null;

    try
    {
      Class.forName("com.mysql.jdbc.Driver");
      con = DriverManager.getConnection("jdbc:mysql://localhost:3306?user=root&password=admin");
      cstmt = con.prepareCall("{call btm.manu123()}");
      boolean b=cstmt.execute();
      if(b)
      {
        rs=cstmt.getResultSet();
        if(rs.next())
```

```
        {
          int id = rs.getInt("ID");
          String name = rs.getString("NAME");
          String dept = rs.getString("DEPT");
          double perc = rs.getDouble("perc");
          System.out.println("ID "+id);
          System.out.println("Name "+name);
          System.out.println("Dept "+dept);
          System.out.println("Percenetage "+perc);
          System.out.println("----------------------");
        }
      }
    }
    catch(ClassNotFoundException | SQLException e)
    {
      e.printStackTrace();
    }
    finally
    {
      if(rs!=null)
      {
        try
        {
          rs.close();
        }
        catch (SQLException e)
        {
          e.printStackTrace();
        }
      }
      if(cstmt!=null)
      {
        try
        {
          cstmt.close();
        }
        catch (SQLException e)
        {
          e.printStackTrace();
        }
      }
      if(con!=null)
      {
        try
        {
          con.close();
        }
        catch (SQLException e)
        {
          e.printStackTrace();
        }
      }
    }
  }
}
---------------------------OUTPUT-------------------------
ID 1
Name Manu KM
Dept Civil
Percenetage 67.53
----------------------
```

3. DDL Operations ?

▼ Ans

▼ Create Table Using execute() with Statement Interface (Oracle)

```
import java.sql.*;
public class Test2
{
```

```
  public static void main(String[] args)
  {
    Connection con=null;
    Statement stmt=null;
    String qry="CREATE TABLE STUDENT(ID NUMBER(10),NAME VARCHAR(20),PERCENT NUMBER(7,2))";
    try
    {
      Class.forName("oracle.jdbc.driver.OracleDriver");
      con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl","scott","tiger");
      stmt=con.createStatement();
      stmt.execute(qry);
      System.out.println("Table is created");
    }
    catch (ClassNotFoundException|SQLException e)
    {
      e.printStackTrace();
    }
    finally
    {
      if(stmt!=null)
      {
        try
        {
          stmt.close();
        }
        catch (SQLException e2)
        {
          e2.printStackTrace();
        }
      }
      if(con!=null)
      {
        try
        {
          con.close();
        }
        catch (SQLException e3)
        {
          e3.printStackTrace();
        }
      }
    }
  }
}
```

▼ Create Table Using executeUpdate() with Statement Interface (Oracle)

```
import java.sql.*;
public class Test2
{
  public static void main(String[] args)
  {
    Connection con=null;
    Statement stmt=null;
    String qry="CREATE TABLE STUDENT1(ID NUMBER(10),NAME VARCHAR(20),PERCENT NUMBER(7,2))";
    try
    {
      Class.forName("oracle.jdbc.driver.OracleDriver");
      con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl","scott","tiger");
      stmt=con.createStatement();
      stmt.executeUpdate(qry);
      System.out.println("Table is created");
    }
    catch (ClassNotFoundException|SQLException e)
    {
      e.printStackTrace();
    }
    finally
    {
      if(stmt!=null)
      {
```

```
        try
        {
          stmt.close();
        }
        catch (SQLException e2)
        {
          e2.printStackTrace();
        }
      }
      if(con!=null)
      {
        try
        {
          con.close();
        }
        catch (SQLException e3)
        {
          e3.printStackTrace();
        }
      }
    }
  }
}
```

▼ Create Table Using executeQuery with Statement Interface (Oracle)

```
import java.sql.*;
public class Test2
{
  public static void main(String[] args)
  {
    Connection con=null;
    Statement stmt=null;
    String qry="CREATE TABLE STUDENT7(ID NUMBER(10),NAME VARCHAR(20),PERCENT NUMBER(7,2))";
    try
    {
      Class.forName("oracle.jdbc.driver.OracleDriver");
      con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl","scott","tiger");
      stmt=con.createStatement();
      stmt.executeQuery(qry);
      System.out.println("Table is created");
    }
    catch (ClassNotFoundException|SQLException e)
    {
      e.printStackTrace();
    }
    finally
    {
      if(stmt!=null)
      {
        try
        {
          stmt.close();
        }
        catch (SQLException e2)
        {
          e2.printStackTrace();
        }
      }
      if(con!=null)
      {
        try
        {
          con.close();
        }
        catch (SQLException e3)
        {
          e3.printStackTrace();
        }
      }
    }
```

```
  }
}
```

▼ Create Table Using execute() with PreparedStatement Interface (Oracle)

```
import java.sql.*;
public class Test2
{
  public static void main(String[] args)
  {
    Connection con=null;
    PreparedStatement pstmt=null;
    String qry="CREATE TABLE STUDENT3(ID NUMBER(10),NAME VARCHAR(20),PERCENT NUMBER(7,2))";
    try
    {
      Class.forName("oracle.jdbc.driver.OracleDriver");
      con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl","scott","tiger");
      pstmt=con.prepareStatement(qry);
      pstmt.execute();
      System.out.println("Table is created");
    }
    catch (ClassNotFoundException|SQLException e)
    {
      e.printStackTrace();
    }
    finally
    {
      if(pstmt!=null)
      {
        try
        {
          pstmt.close();
        }
        catch (SQLException e2)
        {
          e2.printStackTrace();
        }
      }
      if(con!=null)
      {
        try
        {
          con.close();
        }
        catch (SQLException e3)
        {
          e3.printStackTrace();
        }
      }
    }
  }
}
```

▼ Create Table Using executeUpdate() with PreparedStatement Interface (Oracle)

```
import java.sql.*;
public class Test2
{
  public static void main(String[] args)
  {
    Connection con=null;
    PreparedStatement pstmt=null;
    String qry="CREATE TABLE STUDENT4(ID NUMBER(10),NAME VARCHAR(20),PERCENT NUMBER(7,2))";
    try
    {
      Class.forName("oracle.jdbc.driver.OracleDriver");
      con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl","scott","tiger");
      pstmt=con.prepareStatement(qry);
```

```
      pstmt.executeUpdate();
      System.out.println("Table is created");
    }
    catch (ClassNotFoundException|SQLException e)
    {
      e.printStackTrace();
    }
    finally
    {
      if(pstmt!=null)
      {
        try
        {
          pstmt.close();
        }
        catch (SQLException e2)
        {
          e2.printStackTrace();
        }
      }
      if(con!=null)
      {
        try
        {
          con.close();
        }
        catch (SQLException e3)
        {
          e3.printStackTrace();
        }
      }
    }
  }
}
```

▼ Create Table Using executeQuery() with PreparedStatement Interface (Oracle)

```
import java.sql.*;
public class Test2
{
  public static void main(String[] args)
  {
    Connection con=null;
    PreparedStatement pstmt=null;
    String qry="CREATE TABLE STUDENT5(ID NUMBER(10),NAME VARCHAR(20),PERCENT NUMBER(7,2))";
    try
    {
      Class.forName("oracle.jdbc.driver.OracleDriver");
      con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl","scott","tiger");
      pstmt=con.prepareStatement(qry);
      pstmt.executeQuery();
      System.out.println("Table is created");
    }
    catch (ClassNotFoundException|SQLException e)
    {
      e.printStackTrace();
    }
    finally
    {
      if(pstmt!=null)
      {
        try
        {
          pstmt.close();
        }
        catch (SQLException e2)
        {
          e2.printStackTrace();
        }
      }
      if(con!=null)
```

```
      {
        try
        {
          con.close();
        }
        catch (SQLException e3)
        {
          e3.printStackTrace();
        }
      }
    }
  }
}
```

▼ Rename Table

```
import java.sql.*;
public class Test2
{
  public static void main(String[] args)
  {
    Connection con=null;
    Statement stmt=null;
    String qry="RENAME MANU TO MANUKM";
    try
    {
      Class.forName("oracle.jdbc.driver.OracleDriver");
      con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl","scott","tiger");
      stmt=con.createStatement();
      stmt.execute(qry);
      System.out.println("Table is created");
    }
    catch (ClassNotFoundException|SQLException e)
    {
      e.printStackTrace();
    }
    finally
    {
      if(stmt!=null)
      {
        try
        {
          stmt.close();
        }
        catch (SQLException e2)
        {
          e2.printStackTrace();
        }
      }
      if(con!=null)
      {
        try
        {
          con.close();
        }
        catch (SQLException e3)
        {
          e3.printStackTrace();
        }
      }
    }
  }
}
```

▼ Add New Column

```
import java.sql.*;
public class Test2
```

```
{
  public static void main(String[] args)
  {
    Connection con=null;
    Statement stmt=null;
    String qry="ALTER TABLE MANUKM ADD ADDRESS VARCHAR(40) NOT NULL";
    try
    {
      Class.forName("oracle.jdbc.driver.OracleDriver");
      con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl","scott","tiger");
      stmt=con.createStatement();
      stmt.execute(qry);
      System.out.println("Table is created");
    }
    catch (ClassNotFoundException|SQLException e)
    {
      e.printStackTrace();
    }
    finally
    {
      if(stmt!=null)
      {
        try
        {
          stmt.close();
        }
        catch (SQLException e2)
        {
          e2.printStackTrace();
        }
      }
      if(con!=null)
      {
        try
        {
          con.close();
        }
        catch (SQLException e3)
        {
          e3.printStackTrace();
        }
      }
    }
  }
}
```

▼ Remove Column

```
import java.sql.*;
public class Test2
{
  public static void main(String[] args)
  {
    Connection con=null;
    Statement stmt=null;
    String qry="ALTER TABLE MANUKM DROP COLUMN ADDRESS";
    try
    {
      Class.forName("oracle.jdbc.driver.OracleDriver");
      con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl","scott","tiger");
      stmt=con.createStatement();
      stmt.execute(qry);
      System.out.println("Table is created");
    }
    catch (ClassNotFoundException|SQLException e)
    {
      e.printStackTrace();
    }
    finally
    {
      if(stmt!=null)
```

```
      {
        try
        {
          stmt.close();
        }
        catch (SQLException e2)
        {
          e2.printStackTrace();
        }
      }
      if(con!=null)
      {
        try
        {
          con.close();
        }
        catch (SQLException e3)
        {
          e3.printStackTrace();
        }
      }
    }
  }
}
```

▼ Change Column Name

```
import java.sql.*;
public class Test2
{
  public static void main(String[] args)
  {
    Connection con=null;
    Statement stmt=null;
    String qry="ALTER TABLE MANUKM RENAME COLUMN ADDRESS TO BIODATA";
    try
    {
      Class.forName("oracle.jdbc.driver.OracleDriver");
      con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl","scott","tiger");
      stmt=con.createStatement();
      stmt.execute(qry);
      System.out.println("Table is created");
    }
    catch (ClassNotFoundException|SQLException e)
    {
      e.printStackTrace();
    }
    finally
    {
      if(stmt!=null)
      {
        try
        {
          stmt.close();
        }
        catch (SQLException e2)
        {
          e2.printStackTrace();
        }
      }
      if(con!=null)
      {
        try
        {
          con.close();
        }
        catch (SQLException e3)
        {
          e3.printStackTrace();
        }
      }
```

```
    }
  }
}
```

▼ Change Data Type

```
import java.sql.*;
public class Test2
{
  public static void main(String[] args)
  {
    Connection con=null;
    Statement stmt=null;
    String qry="ALTER TABLE MANUKM MODIFY ADDRESS NUMBER(10)";
    try
    {
      Class.forName("oracle.jdbc.driver.OracleDriver");
      con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl","scott","tiger");
      stmt=con.createStatement();
      stmt.execute(qry);
      System.out.println("Table is created");
    }
    catch (ClassNotFoundException|SQLException e)
    {
      e.printStackTrace();
    }
    finally
    {
      if(stmt!=null)
      {
        try
        {
          stmt.close();
        }
        catch (SQLException e2)
        {
          e2.printStackTrace();
        }
      }
      if(con!=null)
      {
        try
        {
          con.close();
        }
        catch (SQLException e3)
        {
          e3.printStackTrace();
        }
      }
    }
  }
}
```

▼ Truncate

```
import java.sql.*;
public class Test2
{
  public static void main(String[] args)
  {
    Connection con=null;
    Statement stmt=null;
    String qry="TRUNCATE TABLE MANUKM";
    try
    {
      Class.forName("oracle.jdbc.driver.OracleDriver");
      con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl","scott","tiger");
```

```
      stmt=con.createStatement();
      stmt.execute(qry);
      System.out.println("Table is created");
    }
    catch (ClassNotFoundException|SQLException e)
    {
      e.printStackTrace();
    }
    finally
    {
      if(stmt!=null)
      {
        try
        {
          stmt.close();
        }
        catch (SQLException e2)
        {
          e2.printStackTrace();
        }
      }
      if(con!=null)
      {
        try
        {
          con.close();
        }
        catch (SQLException e3)
        {
          e3.printStackTrace();
        }
      }
    }
  }
}
```

▼ Drop

```
import java.sql.*;
public class Test2
{
  public static void main(String[] args)
  {
    Connection con=null;
    Statement stmt=null;
    String qry="DROP TABLE MANUKM";
    try
    {
      Class.forName("oracle.jdbc.driver.OracleDriver");
      con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl","scott","tiger");
      stmt=con.createStatement();
      stmt.execute(qry);
      System.out.println("Table is created");
    }
    catch (ClassNotFoundException|SQLException e)
    {
      e.printStackTrace();
    }
    finally
    {
      if(stmt!=null)
      {
        try
        {
          stmt.close();
        }
        catch (SQLException e2)
        {
          e2.printStackTrace();
        }
      }
```

```
      if(con!=null)
      {
        try
        {
          con.close();
        }
        catch (SQLException e3)
        {
          e3.printStackTrace();
        }
      }
    }
  }
}
```

▼ Purge

```
import java.sql.*;
public class Test2
{
  public static void main(String[] args)
  {
    Connection con=null;
    Statement stmt=null;
    String qry="PURGE TABLE MANUKM";
    try
    {
      Class.forName("oracle.jdbc.driver.OracleDriver");
      con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl","scott","tiger");
      stmt=con.createStatement();
      stmt.execute(qry);
      System.out.println("Table is created");
    }
    catch (ClassNotFoundException|SQLException e)
    {
      e.printStackTrace();
    }
    finally
    {
      if(stmt!=null)
      {
        try
        {
          stmt.close();
        }
        catch (SQLException e2)
        {
          e2.printStackTrace();
        }
      }
      if(con!=null)
      {
        try
        {
          con.close();
        }
        catch (SQLException e3)
        {
          e3.printStackTrace();
        }
      }
    }
  }
}
```

2. Retrieving All Records from Employee Table from Oracle ?

▼ Ans

```
import java.sql.*;
import java.util.Date;
public class Test2
{
  public static void main(String[] args)
  {
    Connection con=null;
    PreparedStatement pstmt = null;
    ResultSet rs = null;
    String qry="SELECT * FROM EMP";
    try
    {
      Class.forName("oracle.jdbc.driver.OracleDriver");
      con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl","scott","tiger");
      pstmt = con.prepareStatement(qry);
      rs = pstmt.executeQuery();
      while(rs.next())
      {

        int id = rs.getInt(1);
        String name = rs.getString("ENAME");
        String job = rs.getString("JOB");
        int mgr = rs.getInt(4);
          Date hdt=rs.getDate(5);
          int sal = rs.getInt(6);
          int com = rs.getInt(7);
          int deptno = rs.getInt(8);

        System.out.println("ID     : "+id);
        System.out.println("Name     : "+name);
        System.out.println("Job      : "+job);
        System.out.println("MGR      : "+mgr);
        System.out.println("Hiredate : "+hdt);
        System.out.println("Salary   : "+sal);
        System.out.println("Commision: "+com);
        System.out.println("Deptno   : "+deptno);
        System.out.println("-----------------------");
      }
    }
    catch (ClassNotFoundException|SQLException e)
    {
      e.printStackTrace();
    }
    finally
    {
      if(rs!=null)
      {
        try
        {
          rs.close();
        }
        catch (SQLException e3)
        {
          e3.printStackTrace();
        }
      }
      if(pstmt!=null)
      {
        try
        {
          pstmt.close();
        }
        catch (SQLException e2)
        {
          e2.printStackTrace();
        }
      }
      if(con!=null)
      {
        try
        {
          con.close();
```

```
        }
        catch (SQLException e3)
        {
          e3.printStackTrace();
        }
      }
    }
  }
}
```