# ICON OF JAVA

# ADV. JAVA

# JDBC

## 5. Steps To design Jdbc Applications

**Mr. Nagoor Babu** M.Tech

**[ICON of JAVA]**
**(Sun certified & Realtime Expert)**
**Ex. HCL Employee**
**Trained Lakhs of Students for last 14 years across INDIA**

**India's No.1 Software Training Institute**

**DURGASOFT**

**www.durgasoft.com Ph: 9246212143 ,8096969696**

## Steps To design Jdbc Applications

1)Load and register the Driver.

2)Establish the connection between Java Application.

3)Prepare either Statement or preparedStatement or CallableStatement Objects.

4)Write and execute SQL Queries.

5)close the connection.

## Steps to design JDBC Application:

1)Load and register the Driver.

2)Establish the connection between Java Application.

3)Prepare either Statement or preparedStatement or CallableStatement Objects.

4)Write and execute SQL Queries.

5)close the connection.

## 1)Load and Register the Driver:

In general Driver is an interface provided by Sun Microsystems and whose implementation classes are provided by the Database Vendors as part of their Database Softwares.

-->To load and Register the Driver first we have to make available Driver implementation to JDBC application.For this we have to set classpath environment variable to the location Where we have Driver implementation class.

-->If we want to use Type1 Driver provided by Sun MicroSystems in our JDBC applications then it is not required to set classpath environment variable because Type1 Driver was provided by Sun MicroSystems as part of Java Software in the form of**sun.jdbc.odbc.JdbcOdbcdriver**

-->If we want to use Type1 Driver in our JDBC applications then before loading we have to Configure the **Microsoft product odbc Driver.**

-->To configure Microsoft product odbc Driver we have to use the following path.

**To setting DSN:-**

$$\begin{tikzcd}
\text{Start} \arrow[d] \\
\text{Control Panel} \arrow[d] \\
\text{System and Security} \arrow[d] \\
\text{Administrative Tools} \arrow[d] \\
\text{Data Sources (ODBC)} \arrow[d] \\
\text{user DSN} \arrow[d] \\
\text{Click on Add button} \arrow[d] \\
\text{Select Microsoft ODBC for the Oracle} \arrow[d] \\
\text{Click on Finish button} \arrow[d] \\
\text{Provide DSN name (provide any name)} \arrow[d] \\
\text{Click on OK}
\end{tikzcd}$$

-->To load and register Driver in our Jdbc applications we have to use the following method from class 'Class'

Public static Class forName(String class_Name)

Eg:Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

-->When JVM encounter the above instruction JVM will pickup the parameter that is **JDBCOdbcDriver**

Class name and JVM will search for its .class file in the current location,if it is not available then JVM will

search for it in Java predefined library.

-->If JVM identify JDBCODBCDriver.class file in Java pre-defined library(rt.jar) then JVM will load

**JdbcOdbcDriver** class byte code to the memory.

-->At the time of loading JdbcOdbcDriver class byte code to the memory JVM will execute a static block,

As part of this JVM wiil execute a method call like**DriverManager.registerDriver(--);** by the execution of registerDriver() method only JDBCOdbcDrvier will be available to our Jdbc applications.

-->In case of Type1 Driver if we use either Jdbc 4.0 version or Jdk 6.0 version then it is optional

To perform loading and register the driver step because JVM will perform Drvier registration

automatically at the time of establishing the connection between Java application and Database.

NOTE:To prepare Jdbc applications Java API has provided the required pre-defined library in the form

of java.sql package so that we have to import this package in our Java file.

Import  java.sql.*;

-->java.sql package includes the following pre-defined library to design Jdbc applications.

**java.sql package includes the following predefined library:-**
**I-----→interface            C-------→class**

1. Driver (I)
2. DriverManager (C)
3. Connection (I)
4. Statement (I)
5. PreparedStatement (I)
6. ResultSet (I)
7. ResultSetMetaData (I)
8. DatabaseMetaData (I)
9. Savepoint(i)

## 2)Establish the Connection between Java application and Database:

-->To establish the connection between Java application and Database we have to use the following Method from **DriverManager class.**

Public static Connection getConnection(String url,String db_user_name,String db_password)

Ex:Connection con=DriverManager.getConnection("jdbc:odbc:dsnName","system","durga");

-->When JVM encounter the above instruction JVM will access getConnection method,as part

of the getConnection method JVM will access connect() method to establish virtual socket connection

Between Java application and database as per the url which we provided.

-->wher getConnection() method will take three parameters

1.Driver URL
2.Database username
3.Database password

-->In general from Driver to Driver Driver class name and Driver url will varied.

-->If we use Type1 Driver then we have to use the following Driver class name and URL

d-class : sun.jdbc.odbc.JdbcOdbcDriver
url :jdbc:odbc:dsnName

-->In general all the Jdbc Drivers should have an url with the following format.

**main-protocol: sub-protocol**

-->where main-protocol name should be Jdbc for each and every Driver but the sub protocol name should be varied from Driver to Driver.

Q) In Jdbc applications getConnecction() method will establish the connection between Java application and Database and return connection object but connection is an interface how it is possible to Create connection object?

Ans:In general in Java technology we are unable to create objects for the interfaces directly,if we want to accommodate interface data in the form of objects then we have to take either an implementation class or Anonymous Inner class.

-->If we take implementation class as an alternative then it may allow its own data a part from the data Declared in interface and implementation class object should have its own identity instead of interface identity.

-->If we want to create an object with only interface identity and to allow only interface data we have to use Anonymous inner class as an alternative.

-->In jdbc applications getConnection() method will return connection object by returning anonymous Inner class object of connection interface.

NOTE: To create connection object taking an implementation class or anonymous inner class is Completely depending on the Driver Implementation.

## 3)Create either Statement or PreparedStatment or CallableStatement objects as per the requirement:

As part of the Jdbc applications after establish the connection between Java application and Database

We have to prepare SQL Queries,we have to transfer SQL Queries to the databseEngine and we have to

make Database Engine to execute SQL Queries.

-->To write and execute SQL Queries we have to use same predefined library from Statement prepared

Statement and callableStatement.

-->To use the above required predefined library we have to prepare either Statement or

preparedStatement or CallableStatement objects.

Q)What is the difference between Statement,PreparedStatement and Callable Statement Objects.
Ans:
-->In Jdbc applications when we have a requirement to execute all the SQL Queries independently we have to use Statement.

-->In jdbc applications when we have a requirement to execute the same SQL Query in the next

Sequence where to improve the performance of JDBC application we will use prepared Statement.

-->In jdbc applications when we have a requirement to access stored procedures and functions available

At Database from Java application we will use Callable Statement object.

-->To prepare Statement object we have to use the following method from Connection.

    Public Statement createStatement()

    Ex: Statement st=con.createStatement();

Where createStatement() method will return Statement object by creating Statement interfaces Anonymous inner class object.

## 4)Write and execute SQL Queries:

1.executeQuery()
2.executeUpdate()
3.execute()

Q)What are the differences between executeQuery(),executeUpdate() and execute() method?

Ans:Where executeQuery() method can be used to execute “selection group SQL Queries” in order to fetch(retrieve) data from Database.

-->when JVM encounter executeQuery() method with selection group SQL query then JVM will pickup

Selection group SQL Query,send to JdbcOdbcDriver,it will send to connection.Now connection will carrythat SQL Query to the database engine through Odbc Driver.

-->At database database engine will execute the selection group SQL Query by performing Query

Tokenization,Query parsing,Query optimization and Query Execution.

-->By the execution of selection group SQL Query database engine will fetch the data from database and

return to Java Application.

-->As Java technology is pure object oriented,Java application will store the fetched data in the form of

an object at heap memory called as “ResultSet”.

-->As per the predefined implementation of executeQuery method JVM will return the generated

ResultSet object reference as return value from executeQuery() method.

Public ResultSet executeQuery(String sql_Query) throws SQLException

Ex: ResultSet rs=st.executeQuery(“select * from emp1”);

-->where executeUpdate() method can be used to execute updation group SQLQueries in order to

perform the database operations like create,insert,update,delete,Drop….
-->when JVM encounter updation group SQL Query with execteUpdate() method the JVM will pickup

That Sql Query and send to Database through Database connection.At Database side Database engine

Will execute it,perform updation from Database,identify rowCount value (number of records got updated) and return to Java application.

-->As per the predefined implementation of executeUpdate() method JVM will return row count value

From executeUpdate() method.

Public int executeUpdate(String sql_Query) throws Exception

Ex: int rowCount=st.executeUpdate("update emp1 set esal=esal+500 where esal<1000");

-->In Jdbc applications execute() method can be used to execute both selection group and updation

Group SQL Queries.

-->when JVM encounter selection group SQL Query with execute() method then JVM will send selection

Group SQL Query to database engine,where database engine will execute it and send back the fetched

Data to Java Application.

-->In Java application ResultSet object will be created with the fetched data but as per the predefined

implementation of execute() method JVM will return "true" as a Boolean value.

-->when JVM encounter updation group SQL Query as parameter to execute() method then JVM

Will send it to the database engine,where database engine will perform updations on database and

return row Count value to Java application.But as per the predefined implementation of execute()

method JVM will return "false" as Boolean value from execute() method

public Boolean execute(String sql_Query) throws SQLException.

Ex:
boolean b1=st.execute ("select * from emp1");

boolean b2=st.execcute("update emp1 set esal=esal+500 where esal<10000");

## 5)Close the connection:

In Jdbc applications after the database logic it is convention to close Database connection for this we have to used the following method.

Public void close() throws SQLException

Ex:con.close();

**JdbcApp1: The following example demonstrate how to create a table on Database through a JDBC applicationby taking table name as Dynamic input.**

```
//import section
import java.sql.*;
import java.io.*;
class JdbcApp1{
public static void main(String args[]) throws Exception{

//load a register driver

Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

//establish connection between Java application and database
Connection con=DriverManager.getConnection("jdbc:odbc:nag","system","durga");

//prepare Statement
Statement st=con.createStatement();

//create BufferedReader
BufferedReader br=new BufferedReader(new InputStreamReader(System.in));

//take table name as dynamic input

System.out.println("Enter table name");

String tname=br.readLine();

//prepare SQLQuery

String sql="create table " + tname + "(eno number,ename varchar2(10),esal number)";

//execute SQL Query

st.executeUpdate(sql);
System.out.println("table created successfully");

//close the connection

con.close();
}}
```

**JdbcApp2: The following example demonstrates how to insert no.of records on database table by taking records data as dynamic input.**

```
import java.io.*;
import java.sql.*;
public class JdbcApp2 {
    public static void main(String[] args)throws Exception {
            Class.forName("oracle.jdbc.OracleDriver");
            Connection
            con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","du
            rga");
            Statement st=con.createStatement();
            BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
            while(true)
            {
                    System.out.print("Employee Number   :");
                    int eno=Integer.parseInt(br.readLine());
                    System.out.print("Employee Name     :");
                    String ename=br.readLine();
                    System.out.print("Employee Salary   :");
                    float esal=Float.parseFloat(br.readLine());
                    System.out.print("Employee Address  :");
                    String eaddr=br.readLine();
                    st.executeUpdate("insert into emp1
                    values("+eno+",'"+ename+"',"+esal+",'"+eaddr+"')");
                    System.out.println("Employee Inserted Successfully");
                    System.out.print("Onemore Employee[Yes/No]?  :");
                    String option=br.readLine();
                    if(option.equals("No"))
                    {
                            break;
                    }
            }
            con.close();
    }}
```

In Jdbc applications if we want to used Type1 driver provided by sun micro systems then we have to usethe following Driver class and URL

**driver.class:sun.jdbc.odbc.JdbcOdbcDriver**
**url:jdbc:odbc:dsnName**

-->Similarly if we want to use Type4 Driver provided by oracle we have to use the following Driver class and URL

**driver_class:oracle.jdbc.driver.OracleDriver**
**url:jdbc:oracle:thin:@localhost:1521:xe**

-->Oracle Driver is a class provided by oracle software in the form of Ojdbc14.jar file

-->Oracle Software has provided ojdbc14.jar file at the following location

**C:\oracleexe\app\oracle\product\10.2.0\server\jdbc\lib\ojdbc.jar**

-->If we want to use Type4 Driver provided by oracle in our Jdbc applications we have to set classpath environment variable to the location where we have ojdbc14.jar

```
D:\jdbc4>set
classpath=%classpath%;C:\oracleexe\app\oracle\product\10.2.0\server\jdbc\lib\ojdbc14.jar
```

**JdbcApp3: The following example Demonstrate how to perform updations on Database table through Jdbc Application**

```
import java.io.*;
import java.sql.*;
public class JdbcApp3 {
    public static void main(String[] args)throws Exception {
            //Class.forName("oracle.jdbc.OracleDriver");
            Connection   con=DriverManager.getConnection
            ("jdbc:oracle:thin:@localhost:1521:xe","system","durga");
            Statement st=con.createStatement();
            BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
            System.out.print("Bonus Amount  :");
            int bonus_Amt=Integer.parseInt(br.readLine());
            System.out.print("Salary Range  :");
            float sal_Range=Float.parseFloat(br.readLine());
            int rowCount=st.executeUpdate
            ("update emp1 set esal=esal+"+bonus_Amt+" where    esal<"+sal_Range);
            System.out.println("Employees Updated  :"+rowCount);
            con.close();
    }}
```

JdbcApp4: **The following example demonstrates how to delete no.of records from database table through a Jdbc application**

```
import java.io.*;
import java.sql.*;
public class JdbcApp4 {
    public static void main(String[] args)throws Exception {
            DriverManager.registerDriver(new oracle.jdbc.OracleDriver());
            Connection   con=DriverManager.getConnection
            ("jdbc:oracle:thin:@localhost:1521:xe","system","durga");
            Statement st=con.createStatement();
            BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
            System.out.print("Salary Range   :");
            float sal_Range=Float.parseFloat(br.readLine());
            int rowCount=st.executeUpdate("delete from emp1 where esal<"+sal_Range);
            System.out.println("Records Deleted   :"+rowCount);
            con.close();
    }
}
```

-->In jdbc application we will use executeUpdate() method to execute the Updation group SQL queries like create,insert,update,delete,drop,alter and so on.

-->If we execute the SQL Queries like insert,update and delete then really some no.of record will be updated on database table then that number will be return as rowCount value from executeUpdate().

-->If we execute the SQL Queries like create,alter,drop with executeUpdate() method then records manipulation is not available on database,in this context the return value from executeUpdate() method is completely depending on the type of Driver which we used in JDBC application.

-->In the above context if we use type1 Driver provided by SunMicroSystems the executeUpdate() method will return "-1" as rowCount value.

-->For the above requirement if we use Type4 Driver provided by oracle then executeUpdate() method will return "0" as rowCount value

```
JdbcApp5:
import java.sql.*;
    public class JdbcApp5 {
    public static void main(String[] args)throws Exception {
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
            Connection con=DriverManager.getConnection("jdbc:odbc:nag","system","durga");
            Statement st=con.createStatement();
            int rowCount1=st.executeUpdate("create table emp1(eno number)");
            System.out.println(rowCount1);
            int rowCount2=st.executeUpdate("drop table emp1");
            System.out.println(rowCount2);
            con.close();
    }
}
```

```
JdbcApp6:
import java.sql.*;
    public class JdbcApp6 {
    public static void main(String[] args)throws Exception {
            Class.forName("oracle.jdbc.OracleDriver");
            Connection con=DriverManager.getConnection
            ("jdbc:oracle:thin:@localhost:1521:xe","system","durga");
            Statement st=con.createStatement();
            int rowCount1=st.executeUpdate("create table emp1(eno number)");
            System.out.println(rowCount1);
            int rowCount2=st.executeUpdate("drop table emp1");
```

```
                System.out.println(rowCount2);
                con.close();
        }
    }
```