ADVANCE JAVA

COURSE MATERIAL

BY

MR.NAGOOR BABU

DURGA SOFT

Importance of ADVANCED JAVA: DURGASOFT

- Every Software Company will do projects on J2EE Technologies
- If you attend any JAVA based Interview, 40% of the Questions are coming from ADVANCED JAVA only.
- If you are Strong in ADVANCED JAVA then it is very easy to learn all the futer technologies and frameworks like STRUTS, JSF, SPRING,.....
- If you are Strong in ADVANCED JAVA surveying in software companies is very easy.
- ADVANCED JAVA is basis for all the enterprise applications.

Why NAGOORBABU sir ADVANCE JAVA:

- Covered Advance java topics in more depth.
- Provide live execution for each and every application in the class itself.
- Very good notes dictation for each and every topic which contains 500+ pages.
- Certification Oriented training on each and every topic (OCWCD).
- Conducting classes Even in Sundays for the benefit of Students i.e Covering each and every topic from scratch and also to complete course with in the time.
- Reviewing previous day topic before starting the class.
- Conducting interview questions sessions at the end of each and every topic.
- Covering interview questions and answers in detail in the notes dictation.
- Providing 2 mini projects on Advance JAVA.

Java Database Connectivity:

1.Storage Areas

1)Temporary Storage Areas

2)Permanent Storage Areas

2.QueryProccessing System

1)Query Tokenization

2)Query Processing

3)Query Optimization

4)Query Execution

3.Driver and Driver Types

1)Type 1 Driver

2)Type 2 Driver

3)Type 3 Driver

4)Type 4 Driver

4.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.

6.ResultSet and ResultSet Types

1)Read only ResultSet

2)UpdatableResultSet

3)Forward only ResultSet

4)ScrollableResultSets

1)Scroll Sensitive ResultSet

2)Scroll Insensitive ResultSet

7.Prepared Statement

1)PreparedStatement with insert sql query

2)PreparedStatement with update sql query

3)PreparedStatement with select sql query

8.Callable Statement

1)CallableStatement with procedure

2)CallableStatement with function

3)CallableStatement with CURSOR Type Procedure

4)CallableStatement with CURSOR type function

9.Transaction Management

1. Atomicity
2. Consistency
3. Isolation
4. Durability

10.Savepoint

11.BatchUpdations

12.Connection Pooling

13.BLOB and CLOB

Servlets:

1.Introduction

1) Standalone Applications

2) Enterprise Applications

2.Client-Server Arch

1)Client

2)Server

3)Protocol

3.Servlets Design

4.Servlets Lifecycle

5.User Interface

1.Static Form Generation

2.Dynamic Form Generation

6.ServletConfig

7.Servlet Context

8.Servlet Communication

1. Browser-servlet
2. Web-component
3. Applet-Servlet

9.Session Tracking Mechanisms

1. HttpSession Session Tracking Mechanism
2. Coockies Session Tracking Mechanism
3. URL-Rewriting Session Tracking Mechanism
4. Hidden Form Fields Session Tracking Mechanism

10.Servlets Filters

11.Servlets Wrappers

12.Servlets Listeners

Java Server Pages:

1.Introduction

2.JSP Deployment

3.JSPLifeCycle

4.Jsp Elements

1.Jsp Directives

2.Scripting Elements

3.Jsp Actions

5.JSP Directives

1. Page Directive
2. Include Directive
3. Taglib Directive

6.JSP Scripting Elements

1. Declarations
2. Scriptlets
3. Expressions

7.JSP implicit objects

1.out
2. request
3. response
4. config
5. application
6. session
7. exception
8. page
9. pageContext

8.JSP Scopes

1.Page Scope

2.request Scope

3.Application Scope

4.SessionScoper

9.JSP Standard Actions

10.JSP Custom Actions

11.JSTL

1. Core Tags
2. XML Tags
3. Internationalization or I18N Tags (Formatted tags)
4. SQL Tags
5. Functions tags

12.Expression Language

1)EL operators

2)EL implicit objects.

3)EL functions.

Storage Areas

As part of the Enterprise Application development it is essential to manage the organizations data like Employee Details, CustomerDetails, Products Details..etc

-->To manage the above specified data in enterprise applications we have to use storage areas (Memoryelements).There are two types of Storage areas.

1) Temporary Storage Areas:

These are the memory elements, which will store the data temporarily.
Eg:Buffers,Java Objects

2) Permanent Storage Objects:

These are the memory elements which will store data permanently.
Eg:FileSystems,DBMS,DataWareHouses.

File Systems:
It is a System, it will be provided by the local operating System.

--->Due to the above reason File Systems are not suitable for platform independent technologies like JAVA.
--->File Systems are able to store less volumes of the data.
--->File Systems are able to provide less security.
--->File Systems may increases data Redundancy.
--->In case of File Systems Query Language support is not available. So that all the database operations are complex.

DBMS:

-->Database Management System is very good compare to file System but still it able to store less data when compared to DataWareHouses.
-->DBMS is very good at the time of storing the data but which is not having Fast Retrieval Mechanisms.

DataWareHouses:

When Compared to File Systems and DBMS it is able to store large and large volumes of data.

-->Data ware houses having fast retrieval mechanisms in the form of data mining techniques.

Q)What is the difference between database and database management system?
Ans:
DataBase is a memory element to store the data.
Database Management System is a Software System,it can be used to manage the data by storing it on database and retrieving it form Database.
Database is a collection of interrelated data as a single unit.
DBMS is a collection of interrelated data and a set of programs to access the data.

There are three types of DBMS:

1)RDMS(Relational Database Management Systems)
2)OODBMS(Object Oriented DataBase Management Systems)
3)ORDBMS(Object Relational DataBase Management Systems)

1)Relational Database Management Systems:

-->It is a DBMS,it can be used to represent the data in the form of tables.
-->This DBMS will use SQL3 as a Query Language to perform DataBase Operations.

2)Object Oriented DataBase Management System:

-->It is DataBase Management System,it will represents the data in the form of Objects.
-->This database management system will require OQL(Object Query Language)as Query language to perform database operations.

3)Object Relational DataBase Management System:

-->It is a DataBaseManagement System,it will represents some part of data in the form of Tables and some other part of the data in the form of objects
-->This DataBaseManagement System will require SQl3 as Query Language to perform database operations.

Where SQL3 is the combination of SQL2 and OQL

SQL3=SQL2+OQL

Query Processing System:

When we submit an SQL Query to the Database then Database Engine will Perform the following Steps.

Step1:

Query Tokenization:

This Phase will take SQL Query as an Input,divided into no.of tokens andGenerate Stream of tokens as an output.

Step2:

Query Processing:

This phase will take Stream of tokens as an Input,constructs Query Tree with the Tokens,if Query Tree Success then no Syntax error is available in the provided SQL Query.If Query Tree is not Success then there are some syntax errors in the provided SQL Query.

Step3:

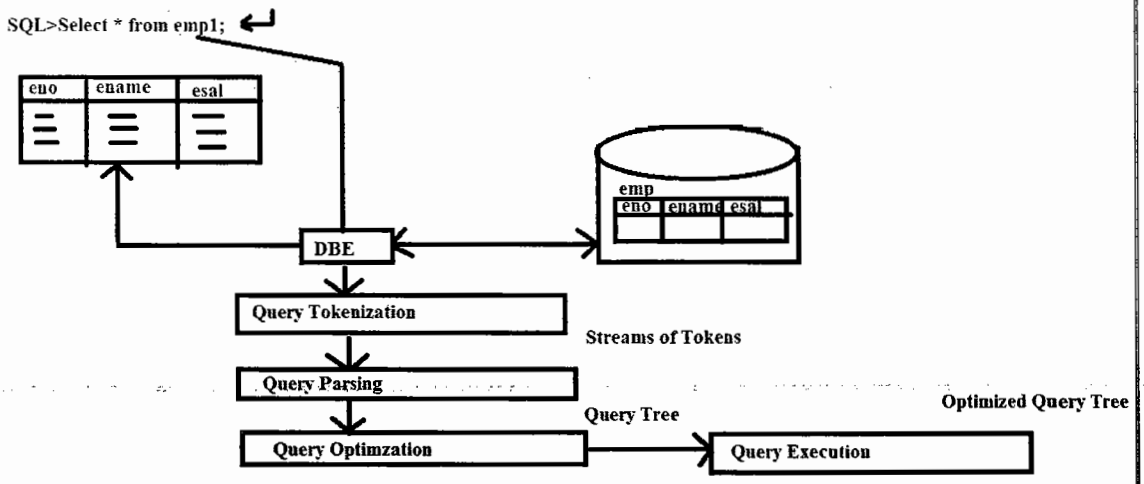
Query Optimization:

The main purpose of Query Optimization phase is to perform optimization on Query Tree in order to reduce execution time and to optimize memory utilization.

Step4:

Query Execution:

This phase will take optimized Query Tree as an input and execute the Query by using interpreters.



JDBC(Java DataBase Connectivity):

-->The process of interacting with the database from Java Applications is called as JDBC.

-->JDBC is an API,which will provide very good predefined library to connect with database from JAVAApplications in order to perform the basic database operations:

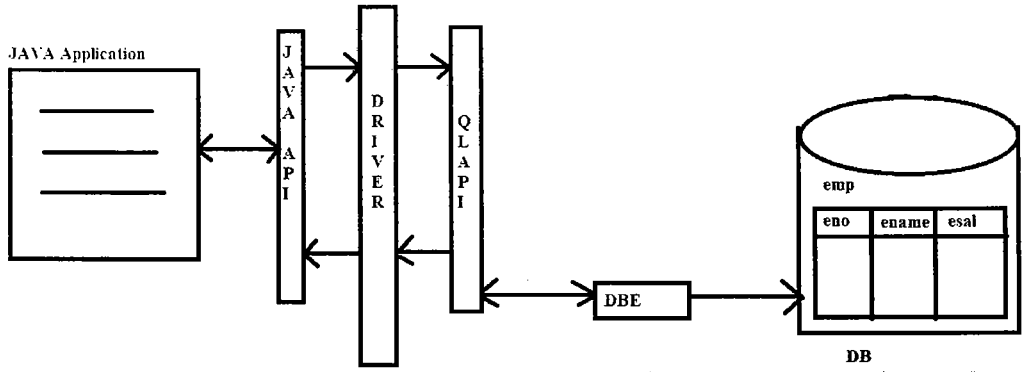
-->In case of JDBC Applications we will define the database logic and Java application and we will send a Java represented database logic to Database Engine.But database engine is unable to execute the Java represented database logic,it should required the database logic in Query Language Representations.

-->In the above context, to execute JDBC applications we should require a conversion mechanism to convert the database logic from Java representations to Query language representations and fromQuery language representations to Java representations.

-->In the above situation the required conversion mechanisms are available in the form of a software called as "Driver".

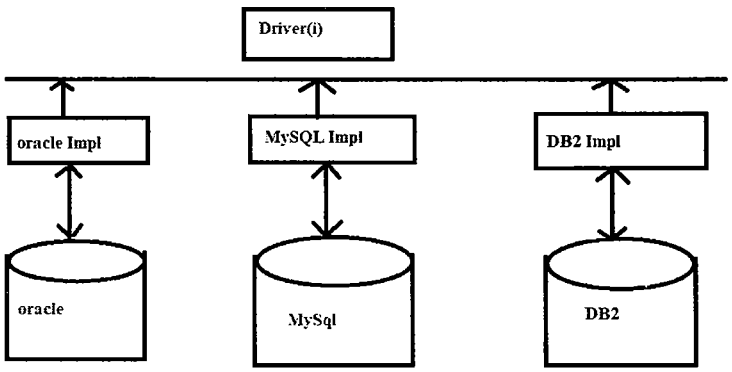
Driver:

-->Driver is an interface existed between Java application and database to map Java API calls to Query language API calls and Query language API calls to Java API calls.



-->To provide driver as a product Sun MicroSystems has provided Driver as an interface and Sun MicroSystems.lets the database vendors to provide implementation classes to the driver interface as part of their database software's.

-->If we want to use Drivers in JDBC applications then we have to get Driver implementation from the respective database software's.



-->There are 180+ number of drivers but all these drivers could be classified into the following four types

1)Type 1
2)Type 2
3)Type 3
4)Type 4

1)Type 1 Driver:

-->Type 1 Driver is also called as JDBC-ODBC Driver and Bridge Driver.

-->JDBC-ODBC Driver is a driver provided by Sun Micro Systems as an Implementation to Driver Interface.

-->Sun MicroSystems has provided JDBC-ODBC Driver with the inter dependent on the Microsoft's productODBC Driver.

-->ODBC Driver is a Open Specification,it will provide very good environment to interact with any type of database from JDBC-ODBC Driver.

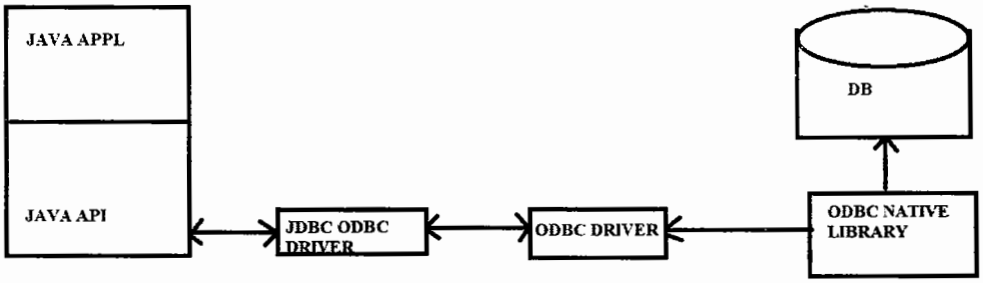
-->If we want to use JDBC-ODBC Driver in our JDBC Applications first we have to install the MicroSoft Product ODBC Driver native library.

-->To interact with the database from Java Application if we use JDBC-ODBC Driver then we shouldrequire two types conversions so that JDBC-ODBC Driver is Slower Driver.

-->JDBC-ODBC Driver is highly recommended for stand alone applications,it is not suitable for web applications,distributed applications and so on.

-->JDBC-ODBC Driver is suggestable for Simple JDBC applications,not for complex JDBC applications.

-->The portability of the JDBC-ODBC Driver is very less.



2)Type 2 Driver:

-->Type 2 Driver is also called part java,part native driver that is Type 2 Driver was implemented by using Java implementations and the database vendor provided native library.

-->When compared to Type1 Driver Type2 Driver is faster Driver because it should not require two times conversions to interact with the Database from Java Applications.

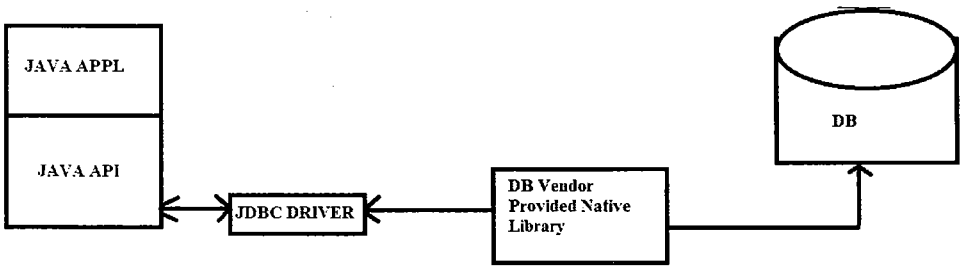
-->When compared to Type1 Driver Type2 driver portability is more.

-->Type2 Driver is still recommended for standalone application not suggestible for web applications and Enterprise applications.

-->If we want to use Type2 Driver in our Jdbc applications then we have to install the database vendor provided native library.

-->Type2 Driver is cast full Driver among all the drivers.

-->Type2 Driver's portability is not good when compared to Type3 Driver and Type4 Driver.



3)Type 3 Driver:

-->Type 3 Driver is also called as MiddleWare DataBase Server Access Driver and NetWorkDriver.

-->Type 3 Driver is purely designed for Enterprise applications it is not suggestible for stand alone applications.

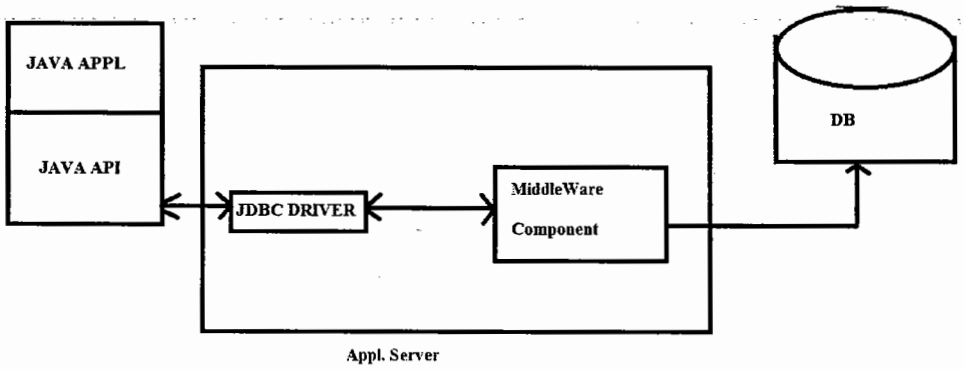
-->Type 3 Driver portability is very good when compared to Type1 and Type2 Driver's.

-->Type 3 Driver will provide very good environment to interact with multiple no.of databases.

-->Type 3 Driver will provide very good environment to switch from one database to another database without having modifications in client applications.

-->Type 3 Driver should not require any native library installations, it should require the Compatibility with the application server.

-->Type 3 Driver is fastest Driver when compared to all the Drivers.



4)Type 4 Driver:

-->Type 4 Driver is also called as pure Java Driver and Thin Driver because Type 4 Driver was implemented completely by using java implementations.

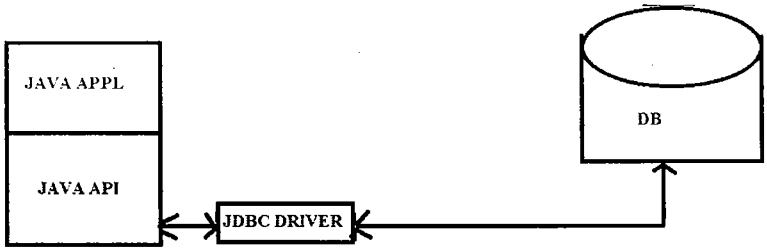
-->Type 4 Driver is the frequent used Driver when compared to all the remaining Drivers.

-->Type 4 Driver is recommended for any type application includes standalone applications, Network Applications....

-->Type 4 Driver portability is very good when compared to all the remaining Drivers.

-->Type 4 driver should not require any native library dependences and it should require one time conversion to interact with database from Java Applications.

-->Type 4 is the cheapest Driver among all.



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:-

Start
↓
Control Panel
↓
System and Security
↓
Administrative Tools
↓
Data Sources (ODBC)
↓
user DSN
↓
Click on Add button
↓
Select Microsoft ODBC for the Oracle
↓
Click on Finish button
↓
Provide DSN name (provide any name)
↓
Click on OK

-->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();

1)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 CreateTableEx{
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();

}}
```

2)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","d
        urga");
        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

3)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();
    }}
```

4)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

```
5) 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();
    }
}
```

```
6)  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();
    }
}
```


ResultSet:

In Jdbc applications if we use Selection group SQL Query as parameter to executeQuery() method then JVM will send that selection group SQL Query to the Database Engine,where Database Engine will execute that SQL Query,fetch the data from Database and send back to Java application.

--->At Java Application the fetched data will be stored 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.

ReslutSet rs=st.executeQuery("select * from emp1");

-->When ResultSet object is created automatically a cursor will be created positioned before the first record.

-->If we want to read the records data from resultset object the for each and every record we have to check whether the next record is available or not from resultset cursor position, if it is available then we have to move resultset cursor to the next record position.

-->To perform the above work we have to use the following method from resultset

public boolean next()

-->After getting ResultsSet cursor to a particular Record position we have to retrieve the data from respective columns,for this we have to use the following overloaded method

public xxx getxxx(int field_No)

public xxx getxxx(String field_Name)

where xxx may be byte,short,int.....

```
Ex: while(rs.next())
{
System.out.println(rs.getInt(1));
System.out.println(rs.getString(2));
System.out.println(rs.getFloat(3));
}
```

7)The following example demonstrate how to fetch the data from database through ResultSet object

```
import java.sql.*;
import oracle.jdbc.*;
public class JdbcApp7 {
    public static void main(String[] args)throws Exception {
            OracleDriver driver=new OracleDriver();
            Connection  con=DriverManager.getConnection
            ("jdbc:oracle:thin:@localhost:1521:xe","system","durga");
            Statement st=con.createStatement();
            ResultSet rs=st.executeQuery("select * from emp1");
            System.out.println("ENO\tENAME\tESAL\tEADDR");
            System.out.println("----------------------------");
            while(rs.next()){
            System.out.println(rs.getInt(1)+"\t"+rs.getString(2)
```

```
                +"\t"+rs.getFloat(3)+"\t"+rs.getString(4));
                }
                con.close();
        }}
```

-->In jdbc applications execute() method can be used to execute both selection group SQL Queries and updation group SQL Queries one at a time.

-->If we use execute() method to execute selection group SQL Query then JVM will send that SQL Query to database engine where Database engine will execute Selection group SQL Query, fetch data from database and return to Java application.At java application the fetched data will be stored in the form of ResultSet object but as per the internal implementation of execute() method JVM will return "true" as a boolean value.

-->In the above context to retrieve the data from resultset object we have to get ResultSetObject reference explicitly.

-->To get ResultSet object reference explicitly we have to use the following method from Statement.

Public ResultSet getResultSet() throws SQLException

```
8.  import java.sql.*;
    public class JdbcApp8 {
    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();
            boolean b=st.execute("select * from emp1");
            System.out.println(b);
            ResultSet rs=st.getResultSet();
            System.out.println("ENO\tENAME\tESAL\tEADDR");
            System.out.println("----------------------------");

            while(rs.next()){
                    System.out.println(rs.getInt(1)+"\t"+rs.getString(2)
                    +"\t"+rs.getFloat(3)+"\t"+rs.getString(4));

            }
```

```
            con.close();
        }}
```

-->If we use updation group SQL Query as parameter to execute() method then JVM will send that updation group SQL Query to Database engine,where database engine will execute it,perform updations on Database table and return the generated row Count value to Java application.

-->As per the predefined implementation of execute() method JVM will return "false" as a boolean value from execute() method

-->In the above context to retrieve the generated rowCount value we have to use the following method from statement

public int getUpdateCount() throws SQLException

```
9.  import java.sql.*;
    public class JdbcApp9
    {
        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();
                boolean b=st.execute("update emp1 set esal=esal+500 where esal<10000");
                System.out.println(b);
                int rowCount=st.getUpdateCount();
                System.out.println("Records Updated  :"+rowCount);
                con.close();
        }
    }
```

Q)If we use updation group SQL Query as parameter to executeQuery() method then what will be the response from JDBC application?
Ans:In Jdbc applications if we use updation group SQL Query as parameter to executeQuery() method then JVM will send updation group SQL Query to DatabaseEngine,where Database Engine will perform updations and Database and return rowCount Value to Java application but as per the predefined implementation of executeQuery() method JVM will expect ResultSet Object.

-->In the above context raising an exception or not to raise an exception is completely depending on the Type of Driver which we used.

-->For the above requirement if we use Type1 driver then JVM will raise an Exception like java.sql.SQLException:no ResultSet was produced.

```
10.import java.sql.*;
public class JdbcApp12 {
public static void main(String[] args) {
        Statement st=null;
        try{
                Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
                Connection con=DriverManager.getConnection
                ("jdbc:odbc:nag","system","durga");
                st=con.createStatement();
                ResultSet rs=st.executeQuery
                ("update emp1 set esal=esal+500 where esal<10000");
        }
        catch (Exception e){
                e.printStackTrace();
                try{
                        int rowCount=st.getUpdateCount();
                        System.out.println("Row Count :"+rowCount);
                }catch(Exception e1){
                        e1.printStackTrace();
                }
 }}}
```

NOTE:In the above application if we provide getUpdateCount method then we are able to generated the rowCount value in the catch block.

-->For the above requirement if we use Type4 Driver provided by oracle then JVM will not raise any exception,it will prepare a default ResultSet object implicitly.

```
import java.util.*;
class TestTwo{
public static void main(String args[]){
Statement st=null;
try{
Class.forName("oracle.jdbc.driver.oracleDriver");
Connection
con=DriverManager.getConnection(jdbc:oracle:thin:@localhost:1521:xe","system","durga");
st=con.createStatement();
ResultSet rs=st.executeQuery("update emp2 set esal=esal+500 where esal<10000");
```

```
int rowCount=st.getUpdateCount();
System.out.println("Records Updated.."+rowCount);
}
catch(Exception e){
e.printStackTrace();
}}}
```

Q)In Jdbc applications if we provide selection group SQL Query as parameter to executeUpdate() Method then what will be the response from Jdbc application?

Ans:In jdbc applications if we provide selection group SQL Query as parameter to executeUpdate() method then JVM will send that SQL Query to database Engine,where Database engine will fetch the data from database and return to Java application.

-->At java application the returned data will be stored in the form of ResultSet object.

-->As per the predefined implementation of executeUpdate() method JVM will expecting Integer value.

-->In the above context getting an exception or not is completely depends on the Driver which we used in our Jdbc application

-->For the above requirement if we use Type1 Driver then JVM will raise an exception like java.sql.SQLException:no rwoCount was produced.

NOTE:In the above situation if we getResultSet() method in catch block then we are able to get Resultset object.

```
11. import java.sql.*;
    public class JdbcApp10 {
    public static void main(String[] args) {
            Statement st=null;
            try{
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
            Connection con=DriverManager.getConnection("jdbc:odbc:nag","system","durga");
            st=con.createStatement();
            int rowCount= st.executeUpdate("select * from emp1");
            }catch(Exception e){
                    e.printStackTrace();
                    try{
                            ResultSet rs=st.getResultSet();
                            System.out.println("ENO\tENAME\tESAL\tEADDR");
                            System.out.println("-----------------------");
```

```
                        while(rs.next())
                        {
                                System.out.println(rs.getInt(1)+"\t"+rs.getString(2)
                                +"\t"+rs.getFloat(3)+"\t"+rs.getString(4));
                        }
                }catch(Exception e1){
                        e1.printStackTrace();
                }
        }
    }
}
```

-->For the above requirement if we use Type4 Driver then JVM will not raise any Exception, executeUpdate() method will return how many no.of records are retrieved from Database table.

```
12. import java.sql.*;
    public class JdbcApp11 {
    public static void main(String[] args) {
            try{
                    Class.forName("oracle.jdbc.OracleDriver");
                    Connection con=DriverManager.getConnection
                    ("jdbc:oracle:thin:@localhost:1521:xe","system","durga");
                    Statement st=con.createStatement();
                    int rowCount=st.executeUpdate("select * from emp1");
                    System.out.println("Row Count :"+rowCount);
                    ResultSet rs=st.getResultSet();
                    System.out.println("ENO\tENAME\tESAL\tEADDR");
                    System.out.println("----------------------------");


                    while(rs.next())
                    {
                            System.out.println(rs.getInt(1)+"\t"+rs.getString(2)
                            +"\t"+rs.getFloat(3)+"\t"+rs.getString(4));
                    }
            }
            catch (Exception e)
            {
                    e.printStackTrace();
            }
    }
}
```

13.The following example demonstrated how to retrieve the Data from Database and how to display that data through an HTML page

```
import java.sql.*;
import java.io.*;
public class JdbcApp14
{
    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();
            ResultSet rs=st.executeQuery("select * from emp1");
            String data="";
            data=data+"<html><body><center><table border='1' bgcolor='lightblue'>";

    data=data+"<tr><td>ENO</td><td>ENAME</td><td>ESAL</td><td>EADDR</td></tr>";
            while(rs.next())
            {
                    data=data+"<tr>";
                    data=data+"<td>"+rs.getInt(1)+"</td><td>
                    "+rs.getString(2)+"</td><td>"+rs.getFloat(3)+"</td><td>
                    "+rs.getString(4)+"</td>";
                    data=data+"</tr>";

            }
            data=data+"</table></center></body></html>";
            FileOutputStream fos=new FileOutputStream("emp.html",true);
            byte[] b=data.getBytes();
            fos.write(b);
            System.out.println("Open emp.html file to get Employees data");
            fos.close();
            con.close();
    }
}
```

Jdbc-awt app1:

```
package com.durgasoft;
import java.awt.Button;
import java.awt.Color;
import java.awt.FlowLayout;
import java.awt.Font;
import java.awt.Frame;
import java.awt.Graphics;
import java.awt.GraphicsConfiguration;
import java.awt.HeadlessException;
import java.awt.Label;
import java.awt.TextField;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;

public class LoginFrame extends Frame implements ActionListener{
    Label l1,l2;
    TextField tf1,tf2;
    Button b1;
    String status="";
    public LoginFrame() {
            this.setVisible(true);
            this.setSize(500,500);
            this.setTitle("Login Frame");
            this.setBackground(Color.green);
            this.setLayout(new FlowLayout());
            this.addWindowListener(new WindowAdapter(){
                    public void windowClosing(WindowEvent we){
                        System.exit(0);
                    }
            });

            l1=new Label("User Name");
            l2=new Label("Password");
            tf1=new TextField(20);
            tf2=new TextField(20);
            tf2.setEchoChar('*');
            b1=new Button("Login");
            b1.addActionListener(this);

            Font f=new Font("arial",Font.BOLD,20);
            l1.setFont(f);
            l2.setFont(f);
            tf1.setFont(f);
```

```
                tf2.setFont(f);
                b1.setFont(f);

                this.add(l1);
                this.add(tf1);
                this.add(l2);
                this.add(tf2);
                this.add(b1);
        }
        public void actionPerformed(ActionEvent ae) {
                String uname=tf1.getText();
                String upwd=tf2.getText();
                UserService us=new UserService();
                status=us.checkLogin(uname,upwd);
                repaint();
        }
        public void paint(Graphics g){
                Font f=new Font("arial",Font.BOLD,30);
                g.setFont(f);
                g.drawString("Status  :"+status, 50,250);
        }
}

package com.durgasoft;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;

public class UserService {
        Connection con;
        Statement st;
        ResultSet rs;
        String status="";
        public UserService() {
                try {
                        Class.forName("oracle.jdbc.OracleDriver");
                        con=DriverManager.getConnection
                        ("jdbc:oracle:thin:@localhost:1521:xe","system", "durga");
                        st=con.createStatement();
                } catch (Exception e) {
                        e.printStackTrace();
                }
        }


        public String checkLogin(String uname, String upwd){
                try {
```

```
                    rs=st.executeQuery("select * from registered_Users where
                    uname='"+uname+"' and upwd='"+upwd+"'");
                    boolean b=rs.next();
                    if(b==true){
                            status="Login Success";
                    }else{
                            status="Login Failure";
                    }
            } catch (Exception e) {
                    e.printStackTrace();
            }
            return status;
    }

}
```

```
package com.durgasoft;
public class JdbcApp15 {

    public static void main(String[] args) {
            LoginFrame lf=new LoginFrame();

    }

}
```


Jdbc-awt app2

```
package com.durgasoft;
import java.awt.Button;
import java.awt.Color;
import java.awt.Font;
import java.awt.Frame;
import java.awt.Graphics;
import java.awt.GraphicsConfiguration;
import java.awt.HeadlessException;
import java.awt.Label;
import java.awt.TextField;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
```

```
public class EmployeeAddFrame extends Frame implements ActionListener {
    Label l1,l2,l3,l4;
    TextField tf1,tf2,tf3,tf4;
    Button b1;
    String status="";
    public EmployeeAddFrame(){
        this.setVisible(true);
        this.setSize(500, 500);
        this.setTitle("Employee Registration Frame");
        this.setBackground(Color.cyan);
        this.setLayout(null);
        this.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent we){
                System.exit(0);
            }
        });

        l1=new Label("Employee Number");
        l2=new Label("Employee Name");
        l3=new Label("Employee Salary");
        l4=new Label("Employee Address");

        tf1=new TextField(20);
        tf2=new TextField(20);
        tf3=new TextField(20);
        tf4=new TextField(20);

        b1=new Button("ADD");
        b1.addActionListener(this);

        Font f=new Font("arial",Font.BOLD,20);
        l1.setFont(f);
        l2.setFont(f);
        l3.setFont(f);
        l4.setFont(f);
        tf1.setFont(f);
        tf2.setFont(f);
        tf3.setFont(f);
        tf4.setFont(f);
        b1.setFont(f);

        l1.setBounds(50, 100, 200, 25);
        tf1.setBounds(250, 100, 200, 30);
        l2.setBounds(50, 150, 200, 25);
        tf2.setBounds(250, 150, 200, 30);
        l3.setBounds(50, 200, 200, 25);
        tf3.setBounds(250, 200, 200, 30);
        l4.setBounds(50, 250, 200, 25);
```

```
                tf4.setBounds(250, 250, 200, 30);
                b1.setBounds(50, 300, 100, 30);
                this.add(l1);
                this.add(tf1);
                this.add(l2);
                this.add(tf2);
                this.add(l3);
                this.add(tf3);
                this.add(l4);
                this.add(tf4);
                this.add(b1);
        }
        public void actionPerformed(ActionEvent ae){
                try {
                        int eno=Integer.parseInt(tf1.getText());
                        String ename=tf2.getText();
                        float esal=Float.parseFloat(tf3.getText());
                        String eaddr=tf4.getText();

                        EmployeeService es=new EmployeeService();
                        status=es.add(eno,ename,esal,eaddr);
                        repaint();

                } catch (Exception e) {
                        e.printStackTrace();
                }
        }
        public void paint(Graphics g){
                Font f=new Font("arial",Font.BOLD,30);
                g.setFont(f);
                g.drawString("Status:"+status, 50, 400);
        }

}

package com.durgasoft;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;

public class EmployeeService {
    Connection con;
    Statement st;
    ResultSet rs;
    String status="";
```

```
	public EmployeeService() {
		try {
			Class.forName("oracle.jdbc.OracleDriver");
			con=DriverManager.getConnection
			("jdbc:oracle:thin:@localhost:1521:xe", "system", "durga");
			st=con.createStatement();
		} catch (Exception e) {
			e.printStackTrace();
		}
	}
	public String add(int eno,String ename, float esal, String eaddr){
		try {
			rs=st.executeQuery("select * from emp1 where enno="+eno);
			boolean b=rs.next();
			if(b==true){
				status="Employee Existed Already";
			}else{
				st.executeUpdate("insert into emp1
				values("+eno+",'"+ename+"',"+esal+",'"+eaddr+"')");
				status="Employee Registration Success";
			}
		} catch (Exception e) {
			status="Employee Registration Failure";
			e.printStackTrace();
		}
		return status;
	}

}


package com.durgasoft;

public class JdbcApp16 {

	public static void main(String[] args) {
		EmployeeAddFrame f=new EmployeeAddFrame();

	}

}
```

Jdbc-Awt App3

```
package com.durgasoft;
import java.awt.Button;
import java.awt.Color;
import java.awt.FlowLayout;
import java.awt.Font;
import java.awt.Frame;
import java.awt.Graphics;
import java.awt.GraphicsConfiguration;
import java.awt.HeadlessException;
import java.awt.Label;
import java.awt.TextField;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;

public class StudentSearchFrame extends Frame implements ActionListener {
    Label l1;
    TextField tf1;
    Button b1;
    StudentTo sto;
    public StudentSearchFrame()  {
            this.setVisible(true);
            this.setSize(500, 500);
            this.setTitle("Student Search Frame");
            this.setLayout(new FlowLayout());
            this.setBackground(Color.green);
            this.addWindowListener(new WindowAdapter() {
                    public void windowClosing(WindowEvent we){
                            System.exit(0);
                    }
            });

            l1=new Label("Student Id");
            tf1=new TextField(20);
            b1=new Button("Search");
            b1.addActionListener(this);

            Font f=new Font("arial",Font.BOLD,20);
            l1.setFont(f);
            tf1.setFont(f);
            b1.setFont(f);

            this.add(l1);
```

```
                this.add(tf1);
                this.add(b1);
        }
        public void actionPerformed(ActionEvent arg0) {
                String sid=tf1.getText();
                StudentService ss=new StudentService();
                sto=ss.search(sid);
                repaint();
        }
        public void paint(Graphics g){
                Font f=new Font("arial",Font.BOLD,30);
                g.setFont(f);
                if(sto==null){
                        g.drawString("Student Not Existed", 50, 300);
                }else{
                        g.drawString("Student Id :"+sto.getSid(), 50, 250);
                        g.drawString("Student Name :"+sto.getSname(), 50, 300);
                        g.drawString("Student Address :"+sto.getSaddr(), 50, 350);
                }
        }
}


package com.durgasoft;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;

public class StudentService {
        Connection con;
        Statement st;
        ResultSet rs;
        StudentTo sto;
        public StudentService() {
                try {
                        Class.forName("oracle.jdbc.OracleDriver");
                        con=DriverManager.getConnection("jdbc:oracle:oci8:@xe", "system",
"durga");
                        st=con.createStatement();
                } catch (Exception e) {

                }
        }
```

```
	public StudentTo search(String sid){
		try {
			rs=st.executeQuery("select * from student where sid='"+sid+"'");
			boolean b=rs.next();
			if(b==true){
				sto=new StudentTo();
				sto.setSid(rs.getString(1));
				sto.setSname(rs.getString(2));
				sto.setSaddr(rs.getString(3));
			}else{
				sto=null;
			}
		} catch (Exception e) {
			e.printStackTrace();
		}
		return sto;
	}

}
```

```
package com.durgasoft;
public class StudentTo {
	private String sid;
	private String sname;
	private String saddr;
	public String getSid() {
		return sid;
	}
	public void setSid(String sid) {
		this.sid = sid;
	}
	public String getSname() {
		return sname;
	}
	public void setSname(String sname) {
		this.sname = sname;
	}
	public String getSaddr() {
		return saddr;
	}
	public void setSaddr(String saddr) {
		this.saddr = saddr;
```

```
        }

    }
    package com.durgasoft;

    public class JdbcApp17 {

        public static void main(String[] args) {
                StudentSearchFrame sf=new StudentSearchFrame();

        }

    }
```

Jdbc-awt app4

```
package com.durgasoft;
import java.awt.Button;
import java.awt.Color;
import java.awt.FlowLayout;
import java.awt.Font;
import java.awt.Frame;
import java.awt.Graphics;
import java.awt.Label;
import java.awt.TextArea;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.util.ArrayList;

public class EditorFrame extends Frame implements ActionListener {
    Label l;
    TextArea ta;
    Button b;
    EditorService es;
    boolean bol;
    public EditorFrame(){
            this.setVisible(true);
            this.setSize(700, 800);
            this.setTitle("SQL Editor Frame");
            this.setBackground(Color.green);
            this.setLayout(new FlowLayout());
            this.addWindowListener(new WindowAdapter() {
                    public void windowClosing(WindowEvent e){
                            System.exit(0);
```

```
                }
        });

        l=new Label("Provide SQLQuery");
        ta=new TextArea(3,30);
        b=new Button("Execute");
        b.addActionListener(this);

        Font f=new Font("arial",Font.BOLD,20);
        l.setFont(f);
        ta.setFont(f);
        b.setFont(f);

        this.add(l);
        this.add(ta);
        this.add(b);


    }
    public void actionPerformed(ActionEvent ae){
        String query=ta.getText();
         es=new EditorService();
        bol=es.execute(query);
        repaint();
    }
    public void paint(Graphics g){
        Font f=new Font("arial",Font.BOLD,30);
        g.setFont(f);
        if(bol==true){
            ArrayList al=es.getEmps();
            g.drawString("ENO  ENAME  ESAL  EADDR",50,200);
            g.drawString("-----------------------------------", 50, 240);
            int y=300;
            for(int i=0;i<al.size();i++){
                EmployeeTo eto=(EmployeeTo)al.get(i);
            g.drawString(eto.getEno()+"  "+eto.getEname()+"  "+eto.getEsal()+"
            "+eto.getEaddr(), 50, y);
                y=y+50;
            }

        }else{
            int rowCount=es.getRowCount();
            g.drawString("Row Count :"+rowCount, 50, 300);
        }
    }
}
```

```
package com.durgasoft;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
import java.util.ArrayList;

public class EditorService {
    Connection con;
    Statement st;
    ResultSet rs;
    ArrayList al;
    boolean bol;
    int rowCount;
    public EditorService() {
        try {
            Class.forName("oracle.jdbc.OracleDriver");
            con=DriverManager.getConnection("jdbc:oracle:oci8:@xe", "system",
"durga");
            st=con.createStatement();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    public boolean execute(String sql_Query){
        try {

            bol=st.execute(sql_Query);
        } catch (Exception e) {
            e.printStackTrace();
        }
        return bol;
    }
    public ArrayList getEmps(){
        al=new ArrayList();
        try {
            rs=st.getResultSet();
            while(rs.next()){
                EmployeeTo eto=new EmployeeTo();
                eto.setEno(rs.getInt(1));
                eto.setEname(rs.getString(2));
                eto.setEsal(rs.getFloat(3));
                eto.setEaddr(rs.getString(4));
                al.add(eto);
```

```
                        }
                }


        catch (Exception e) {
                        e.printStackTrace();
                }
                return al;
        }
        public int getRowCount(){
                try {
                        rowCount=st.getUpdateCount();
                } catch (Exception e) {
                        e.printStackTrace();
                }
                return rowCount;
        }
}



package com.durgasoft;

public class EmployeeTo {
        private int eno;
        private String ename;
        private float esal;
        private String eaddr;
        public int getEno() {
                return eno;
        }
        public void setEno(int eno) {
                this.eno = eno;
        }
        public String getEname() {
                return ename;
        }
        public void setEname(String ename) {
                this.ename = ename;
        }
        public float getEsal() {
                return esal;
        }
        public void setEsal(float esal) {
                this.esal = esal;
        }
        public String getEaddr() {
                return eaddr;
```

```
        }
        public void setEaddr(String eaddr) {
                this.eaddr = eaddr;
        }

    }

    package com.durgasoft;

    public class JdbcApp18 {

        public static void main(String[] args) {
                EditorFrame f=new EditorFrame();

        }

    }
```


ResultSet Types:

In jdbc applications ResultSets can be divided into two types:

-->As per the ResultSet concurrency there are two types of ResultSets.

1)Read only ResultSet

It is a ResultSet object,it will allow the users to read the data only.

To represent this ResultSet object ResultSet interface has provided the following constant

public static final int CONCUR_READ_ONLY

2)Updatable ResultSet:

It is a ResultSet object,it will allow the users to perform updations on its content.

To represent this resultset object ResultSet interface has provided the following constant.

public static final int CONCUR_UPDATABLE

-->As per the ResultSet cursor movement there are two types of ResultSets

1)Forward only ResultSet:

It is ResultSet object,it will allow the users to iterate the data in forward directiononly.

-->To represent this ResultSet,ResultSet interface has provided the following constant

public static final int TYEP_FORWARD_ONLY

2)Scrollable ResultSet:

These are the Resultset objects,which will allow the users to iterate the data in both

forward and backward directions.

There are two types of Scrollable Resultsets:
1)Scroll sensitive ResultSet
2)Scroll Insensitive ResultSet

Q)What are the differences betwwn Scroll Sensitive ResultSet and Scroll Insensitive ResultSet?

Scroll sensitive ResultSet is a Scrollable resultset object,which will allow the later Database updations.

-->To refer this ResultSet,ResultSet interface has provided the following constant

public static final int TYPE_SCROLL_SENSITIVE

-->Scroll Insensitive ResultSet is scrollable Resultset object,which will not allow the later database updations after creation.

-->To represent this ResultSet,ResultInterface has provided the following constant

public static final int TYPE_SCROLL_INSENDITIVE

-->The default ResultSet type in Jdbc applications is Forward only and Read only.

-->In Jdbc applications if we want to specify a particular type to the ResultSet object then we have to provide the above specified ResultSet constants at the time of creating Statement object,for this we have to use the following method

public Statement createStatement(int forwardonly

/scrollsensitive/scrollinsensitive,int Readonly/updatable)

Ex: Statement
st=con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_UPDATABLE);

-->In jdbc applications by using scrollable ResultSet we are able to retrieve the data in both forward direction and backward direction.

-->To retrieve the data in forward direction for each and every record we have to check whether the next record is available or not,if it is available we have to move resultset cursor to the next record position.when we refer a particular record then we have to retrieve the data from the respective columns.To achieve this we have to use the following piece of code:

```
while(rs.next())
{
System.out.println(rs.getInt(1));
System.out.println(rs.getString(2));

}
```

-->If we want to retrieve the data in Backward direction then for each and every record we have to check whether the previous record is available or not from the resultset cursor position,if is available then we have to move resultset cursor to the previous record.To achieve this we have to use the following methods:

public boolean previous()

-->After moving the resultset cursor to particular record then we have to retrieve the data from the corresponding columns for this we have to use the following methods

public xxx getxxx(int field_NUM)
public xxx getxxx(String field_Name)

where xxx may be byte,short,int.....

```
Ex: while(rs.previous())
{
System.out.println(rs.getInt(1));
System.out.println(rs.getString(2));
System.out.println(rs.getFloat(3));

}
```

```
14. package com.durgasoft;
    import java.sql.Connection;
    import java.sql.DriverManager;
    import java.sql.ResultSet;
    import java.sql.Statement;

public class JdbcApp19 {

     public static void main(String[] args)throws Exception {
              Class.forName("oracle.jdbc.OracleDriver");
              Connection con=DriverManager.getConnection
              ("jdbc:oracle:oci8:@xe", "system","durga");
              Statement st=con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
              ResultSet.CONCUR_UPDATABLE );
              ResultSet rs=st.executeQuery("select * from emp1");
              System.out.println("Data In Forward Direction");
              System.out.println("ENO\tENAME\tESAL\tEADDR\t");
              System.out.println("--------------------------------");
              while(rs.next()){
              System.out.println(rs.getInt(1)+"\t"+rs.getString(2)+"\t"
              +rs.getFloat(3)+"\t"+rs.getString(4));
              }
              System.out.println("Data In Backward Direction");
              System.out.println("ENO\tENAME\tESAL\tEADDR");
              System.out.println("--------------------------------");
              while(rs.previous()){

                        System.out.println(rs.getInt(1)+"\t"+rs.getString(2)+"\t"
                        +rs.getFloat(3)+"\t"+rs.getString(4));
              }
              con.close();
    }
}
```

15.

```
package com.durgasoft;
 import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;

public class JdbcApp20 {

    public static void main(String[] args)throws Exception {
            Class.forName("oracle.jdbc.OracleDriver");
            Connection con=DriverManager.getConnection
            ("jdbc:oracle:oci8:@xe", "system", "durga");
            Statement st=con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
            ResultSet.CONCUR_UPDATABLE );
            ResultSet rs=st.executeQuery("select * from emp1");
            rs.afterLast();
            rs.previous();
            System.out.println(rs.getInt(1));
            rs.beforeFirst();
            rs.next();
            System.out.println(rs.getInt(1));
            rs.last();
            System.out.println(rs.getInt(1));
            rs.first();
            System.out.println(rs.getInt(1));
            rs.absolute(3);
            System.out.println(rs.getInt(1));
            rs.absolute(-3);
            System.out.println(rs.getInt(1));
            rs.first();
            rs.relative(2);
            System.out.println(rs.getInt(1));
            rs.last();
            rs.relative(-2);
            System.out.println(rs.getInt(1));
    }

}
```

Jdbc-Awt app5

```
package com.durgasoft;

import java.awt.Button;
import java.awt.Color;
import java.awt.Font;
import java.awt.Frame;
import java.awt.Graphics;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;

public class PlayerFrame extends Frame implements ActionListener {
    Button b1,b2,b3,b4;
    String label;
    EmployeeTo eto;
    EmployeeService es;
    public PlayerFrame() {
            this.setVisible(true);
            this.setSize(500, 500);
            this.setTitle("Player Frame");
            this.setBackground(Color.green);
            this.setLayout(null);
            this.addWindowListener(new WindowAdapter() {
                    public void windowClosing(WindowEvent we){
                            System.exit(0);
                    }
            });

            b1=new Button("First");
            b2=new Button("Next");
            b3=new Button("Previous");
            b4=new Button("Last");

            b1.addActionListener(this);
            b2.addActionListener(this);
            b3.addActionListener(this);
            b4.addActionListener(this);

            Font f=new Font("arial",Font.BOLD,20);
            b1.setFont(f);
            b2.setFont(f);
            b3.setFont(f);
            b4.setFont(f);
```

```
                b1.setBounds(50, 400, 100, 30);
                b2.setBounds(160, 400, 100, 30);
                b3.setBounds(270, 400, 100, 30);
                b4.setBounds(380, 400, 100, 30);
                this.add(b1);
                this.add(b2);
                this.add(b3);
                this.add(b4);
                es=new EmployeeService();
        }
        public void actionPerformed(ActionEvent ae){
                label=ae.getActionCommand();
                eto=es.getEmployee(label);
                repaint();
        }
        public void paint(Graphics g){
                Font f=new Font("arial",Font.BOLD,30);
                g.setFont(f);
                String msg=es.getMsg();
                if(msg.equals("")){
                        g.drawString("Employee Number :"+eto.getEno(), 50,100);
                        g.drawString("Employee Name   :"+eto.getEname(), 50,150);
                        g.drawString("Employee Salary  :"+eto.getEsal(), 50,200);
                        g.drawString("Employee Address :"+eto.getEaddr(), 50,250);
                }else{
                        g.drawString(msg, 50,300);
                }
        }
}


package com.durgasoft;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;

public class EmployeeService {
        Connection con;
        Statement st;
        ResultSet rs;
        EmployeeTo eto;
        String msg="";
        boolean b=false;
```

```
public EmployeeService() {
        try {
                Class.forName("oracle.jdbc.OracleDriver");
                con=DriverManager.getConnection
                ("jdbc:oracle:thin:@localhost:1521:xe", "system","durga");
                st=con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
                ResultSet.CONCUR_UPDATABLE);
                rs=st.executeQuery("select * from emp1");
        } catch (Exception e) {
                e.printStackTrace();
        }
}
public EmployeeTo getEmployee(String label){
        try {
                if(label.equals("First")){
                        rs.first();
                        msg="";
                }
                if(label.equals("Next")){
                        b=rs.next();
                        if(b==false){
                                msg="No More Records In Forward Direction";
                        }else{
                                msg="";
                        }
                }
                if(label.equals("Previous")){
                        b=rs.previous();
                        if(b==false){
                                msg="No More Records In Backward Direction";
                        }else{
                                msg="";
                        }
                }
                if(label.equals("Last")){
                        rs.last();
                        msg="";
                }
                eto=new EmployeeTo();
                eto.setEno(rs.getInt(1));
                eto.setEname(rs.getString(2));
                eto.setEsal(rs.getFloat(3));
                eto.setEaddr(rs.getString(4));
        } catch (Exception e) {
                e.printStackTrace();
        }
        return eto;
}
```

```
	public String getMsg(){
		return msg;
	}
}



package com.durgasoft;

public class EmployeeTo {
	private int eno;
	private String ename;
	private float esal;
	private String eaddr;
	public int getEno() {
		return eno;
	}
	public void setEno(int eno) {
		this.eno = eno;
	}
	public String getEname() {
		return ename;
	}
	public void setEname(String ename) {
		this.ename = ename;
	}
	public float getEsal() {
		return esal;
	}
	public void setEsal(float esal) {
		this.esal = esal;
	}
	public String getEaddr() {
		return eaddr;
	}
	public void setEaddr(String eaddr) {
		this.eaddr = eaddr;
	}

}
```

```
package com.durgasoft;


public class JdbcApp24 {

    public static void main(String[] args) {
            PlayerFrame pf=new PlayerFrame();
    }

}
```

Scroll Sensitive ResultSet:

```
16.
package com.durgasoft;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;

public class JdbcApp25 {
  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(ResultSet.TYPE_SCROLL_INSENSITIVE,
    ResultSet.CONCUR_UPDATABLE);
    ResultSet rs=st.executeQuery("select * from emp1");
    System.out.println("Data Before Updations");
    System.out.println("ENO    ENAME    ESAL    EADDR");
    System.out.println("----------------------------------------");
    while(rs.next()){
          System.out.println(rs.getInt(1)+"    "+rs.getString(2)+"    "+rs.getFloat(3)+"
    "+rs.getString(4));
    }
    System.out.println("Application is in Pausing state, please update database");
    System.in.read();
    rs.beforeFirst();
    System.out.println("Data After Updations");
    System.out.println("ENO    ENAME    ESAL    EADDR");
```

```
        System.out.println("-------------------------------------");
        while(rs.next()){
          rs.refreshRow();
              System.out.println(rs.getInt(1)+"    "+rs.getString(2)+"    "+rs.getFloat(3)+"
       "+rs.getString(4));
        }
        con.close();
    }

}
```

-->To move ResultSet cursor to before first record we have to use the following method from ResultSet

public void beforeFirst()

-->To move ResultSet cursor to After the last record we have to use the following method from ResultSet

public void afterLast()

-->To move ResultSet cursor to a particular record we have to use the following method from ResultSet

public void absolute(int rec_position)

-->In case of scroll sensitive ResultSet objects to reflect later database updations into the ResultSet object we have to refresh each and every record for this we have to use the following method from ResultSet

public void refreshRow()

-->where refreshRow() method can be used to refresh only one record.

-->If we use Type4 Driver provided by oracle in the above application then JVM will raise an exception like java.sql.SQLException:unsupportedfeature:refreshrow

-->In jdbc applications scroll sensitive ResultSet object should be supported by Type1 Driver provided by Sun MicroSystems,which could not be supported by Type4 Driver provided by oracle.

-->In Jdbc applications scroll Insensitive ResultSet object could not be supported by both Type1 Driver provided by Sun MicroSystems and Type4 Driver provided by oracle.

-->In jdbc applications the main purpose of UpdatableResultSet object is to perform updations on its content in order to perform the manipulations with the data available at Database

-->In jdbc applications Updatable ResultSet objects can be used to insert records on database table to achieve the above requirement we have to use the following steps:

Step1:get Updatable ResultSet object

```
Statement
st=con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_UPDATABLE);

ResultSet rs=st.executeQuery("select * from emp1");
```

Step2: After creating ResultSet Object we have to move ResultSet cursor to end of the ResultSet object
where we have to take a buffer to insert new Record data temporarily to achieve this we have to use the following method from ResultSet

public void moveToInsertRow()

Ex:rs.moveToInsertRow();

Step3:Insert record data on resultset object temporarily to do this we have to use the following method

public void updatexxx(int field_Num,xxx value)

```
Ex:rs.updateInt(1,555);
  rs.updateString(2,'xys');
  rs.updateFloat(3,9000);
```

Step4:Make the temporarily insertion as permanent insertion in resultset object as well as on database table to achieve this we have to use the following method

public void insertRow()

ex:rs.insertRow();

NOTE:The main advantage of this updatable ResultSet object is to perform updations on database table without using SQL Queries.

17.
The following example demonstrates how to insert no.of records into database table through a Jdbc application

```
package com.durgasoft;
    import java.io.BufferedReader;
    import java.io.InputStreamReader;
    import java.sql.Connection;
    import java.sql.DriverManager;
    import java.sql.ResultSet;
    import java.sql.Statement;

    public class Jdbcapp26 {

      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(ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_UPDATABLE
       );

    ResultSet rs=st.executeQuery("select * from emp1");
        rs.moveToInsertRow();
        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();
          rs.updateInt(1, eno);
          rs.updateString(2, ename);
          rs.updateFloat(3, esal);
          rs.updateString(4, eaddr);
          rs.insertRow();

    System.out.println("Employee inserted Successfully");
          System.out.print("Onemore Employee[Yes/no]  :");
          String option=br.readLine();
          if(option.equals("no")){
            break;
          }
        }
```

```
            con.close();
        }

    }
```

NOTE:In jdbc applications updatable ResultSets could not be supported by Type 4 Driver provided by oracle

-->In jdbc applications by using updatable ResultSet object it is possible to update database

-->To perform this we have to use the following steps

Step1:get updatable ResultSet object

Statement
st=con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_UPDATABLE);

ResultSet rs=st.executeQuery("select * from emp1");

Step2:update Resultset Object Temporarily

To achieve this we have to use the following method

public void updatexxx(int field_Name,xxx value)

ex: rs.updateFloat(3,7000.0f);

Step3:Make temporary updation as permanent updation on resultset obejct as well as on database to achieve this we have to use the following method

public void updateRow()

Ex:rs.updateRow();

```
18.
package com.durgasoft;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;

public class Jdbcapp27 {
```

```
    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(ResultSet.TYPE_SCROLL_SENSITIVE,
      ResultSet.CONCUR_UPDATABLE);


      ResultSet rs=st.executeQuery("select * from emp1");
      while(rs.next()){
        float esal=rs.getFloat(3);
        if(esal<10000){
          float new_Sal=esal+500;
          rs.updateFloat(3, new_Sal);
          rs.updateRow();
        }
      }
      con.close();
    }

}
```

-->In Jdbc applications by using updatable ResultSet object it is possible to delete records on database table to achieve this we have to use the following method

public void deleteRow()

Ex:rs.deleteRow();

```
19.
package com.durgasoft;
 import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;

public class Jdbcapp28 {
   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(ResultSet.TYPE_SCROLL_SENSITIVE,
      ResultSet.CONCUR_UPDATABLE);
      ResultSet rs=st.executeQuery("select.* from emp1");
      rs.last();
```

```
            rs.deleteRow();
            st.close();
            con.close();
        }

    }
```

-->To move ResultSet cursor to a particular Record position we have to use the following method from resultset

public void absolute(int position)

-->To move ResultSet cursor over some no.of records we have to use the following method from Resultset

public void relative(int no.of.records)

-->If we have bulk of records in database table,where if we are trying to retrieve all the records at a time into resultset object automatically Jdbc application performance will be reduced.

-->In the above context to improve the performance of Jdbc application we have to fetch the limited no. of records in multiple attempts.

-->To specify the no.of records which we want to fetch at an attempt then we have to use the following method

public void setFetchSize(int size)

-->To get the specified fetch size value from resultset we have to use the following method

public int getFetchSize

What is the difference between Statement and PreparedStatement?
Ans:
In Jdbc applications,when we have a requirement to execute the SQL queries independently,we have to use Statement.
In Jdbc applications,when we have a requirement to execute same SQL query in the next sequence where to improve the performance of Jdbc applications,we have to use PreparedStatement.
To achieve the above requirement,if we use Statement,then for every time of executing the same SQL query,DB engine has to perform Query Tokenization,Query Parsing,Query Optimization and Query Execution without having any validation from one time to another time.

This approach will reduce the performance of Jdbc application.

In the above context,to improve the performance of Jdbc applications,we have to use an alternative where we have to perform Query Processing one time inorder to perform or execute the same SQL Query in the next sequence.

To achieve the above alternative,we have to use PreparedStatement over Statement.

If we want to use PreparedStatement in Jdbc applications we have to use the following steps.

1.Create PreparedStatement object by providing generalised SQL Query:

To create PreparedStatement object,we have to use the following method from Connection.

```
public PreparedStatement prepareStatement(String SQL_format)
```

Eg:

```
PreparedStatement pst=con.PrepareStatement("insert into emp1 values(?,?,?)");
```

When JVM encounter the above instruction,JVM will pickup the provided SQL query and send to DB engine through Jdbc Driver and connection.

Upon receiving SQL query,DB engine will perform query processing and prepare query plan with the parameters,as a result PreparedStatement object will be created at Java application with the parameters

2.Set values to the parameters available in PreparedStatement object as per the applicationrequirement:

To set values to the PreparedStatement object,we have to the following method from PreparedStatement

```
public void setXXX(int parameter_Numb,xxx value)
```

where xxx may be byte,short,int....

Eg:

```
pst.setInt(1,111);
pst.setString(2,"Laddu");
pst.setFloat(3,30000.0f);
```

When JVM encounter the above instructions then JVM will set the specified values to the respective parameters in PreparedStatement object,where these values are reflected to Query plan parameters automatically through the Jdbc driver and connection.

3.Make Database engine to pickup the values from Query plan and perform the respective operationas per the generalised SQL query which we provided

If the generalised SQL query belongs to selection group,then we have to use the following method.

```
public ResultSet executeQuery() throws Exception
```

If the generalised SQL query belongs to updation group,then we have to use the following method

```
public int executeUpdate() throws SQLException
```

Eg: int rowCount=pst.executeUpdate();

```
20. package com.durgasoft;
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;

public class JdbcApp31 {

   public static void main(String[] args)throws Exception {
      Class.forName("com.mysql.jdbc.Driver");
      Connection
con=DriverManager.getConnection("jdbc:mysql://localhost:3306/durgadb","root","root");
      PreparedStatement pst=con.prepareStatement("insert into emp1 values(?,?,?,?)");
      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();

         pst.setInt(1, eno);
         pst.setString(2, ename);
         pst.setFloat(3, esal);
         pst.setString(4, eaddr);

         pst.executeUpdate();
         System.out.println("Employee Inserted Successfully");
         System.out.print("Onemore Employee[yes/no]  :");
```

```
            String option=br.readLine();
            if(option.equals("no")){
               break;
            }
         }
         con.close();
      }}
```

21.

```
package com.durgasoft;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;

public class JdbcApp32 {

   public static void main(String[] args)throws Exception {
      Class.forName("com.mysql.jdbc.Driver");
      Connection
con=DriverManager.getConnection("jdbc:mysql://localhost:3306/durgadb","root","root");
      PreparedStatement pst=con.prepareStatement("update emp1 set esal=esal+? where
esal<?");
      pst.setInt(1, 500);
      pst.setFloat(2, 10000.0f);
      int rowCount=pst.executeUpdate();
      System.out.println("Records Updated  :"+rowCount);
      con.close();
   }

}
```

22.

```
package com.durgasoft;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;

public class JdbcApp33 {
   public static void main(String[] args) throws Exception{
      Class.forName("com.mysql.jdbc.Driver");
      Connection
con=DriverManager.getConnection("jdbc:mysql://localhost:3306/durgadb","root","root");
      PreparedStatement pst=con.prepareStatement("select * from emp1 where esal<?");
      pst.setFloat(1, 10000.0f);
      ResultSet rs=pst.executeQuery();
```

```
        System.out.println("ENO   ENAME   ESAL   EADDR");
        System.out.println("-----------------------------------");
        while(rs.next()){
          System.out.println(rs.getInt(1)+"    "+rs.getString(2)+"    "+rs.getFloat(3)+"
"+rs.getString(4));
        }
        con.close();
    }


}
```

23.

```
package com.durgasoft;
 import java.sql.Connection;
import java.sql.Date;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.util.Properties;

public class JdbcApp35 {
   public static void main(String[] args) throws Exception{
      Class.forName("com.mysql.jdbc.Driver");
      Properties p=new Properties();
      p.setProperty("user", "root");
      p.setProperty("password", "root");
      Connection con=DriverManager.getConnection("jdbc:mysql://localhost:3306/durgadb",p);
      PreparedStatement pst=con.prepareStatement("insert into student values(?,?,?,?)");
      pst.setString(1, "S-111");
      pst.setString(2, "Durga");
      pst.setString(3, "Java");
      Date  d=Date.valueOf("2015-01-25");
      pst.setDate(4, d);
      pst.executeUpdate();

      pst.setString(1, "S-222");
      pst.setString(2, "Anil");
      pst.setString(3, "Oracle");
      java.util.Date date=new java.util.Date();
      int val=date.getDate();
      java.sql.Date dt=new java.sql.Date(val);
      pst.setDate(4, dt);
      pst.executeUpdate();
      con.close();
```

```
    }
}
```

Batch Updations:

In general in Jdbc applications, it is required to provide number of SQL queries according to application requirement.

With the above, if we execute the Jdbc application then JVM will send all the SQL queries to the database in a sequential manner.

If we use the above convention to execute SQL queries in Jdbc application we have to spend a lot of time only to carry or transfer SQL queries from Java application to database this approach will reduce the performance of Jdbc application.
In the above context, to improve the performance of the Jdbc applications we have to use Batch updations.

In batch updations, we will gather or collect all the updation group SQL queries as a single unit called as Batch and we will send batch of updation group SQL queries at a time from Java application to database.

At database, Database Engine may execute all the SQL queries and generate respective row count values in the form of an array to Java application.

To add an SQL query to batch we have to use the following method from Statement.

public void addBatch(String query);

To send batch of updation group SQL queries at a time from Java application to database and to make the Database Engine to execute all the batch of updation group SQL queries we have to use the following method from Statement.

public int[] executeBatch()

Where int[] will represent all the row count values generated from the updation group SQL queries.

Batch updations with Statement:

```
24.package com.durgasoft;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Statement;
```

```
public class JdbcApp30 {
   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();
      st.addBatch("insert into emp1 values(666,'FFF',9000,'Hyd')");
      st.addBatch("update emp1 set esal=esal-500 where esal<10000");
      st.addBatch("delete from emp1 where eno=555");
     // st.addBatch("select * from emp1");--> java.sql.BatchUpdateException
      int[] rowCounts=st.executeBatch();
      for(int i=0;i<rowCounts.length;i++){
         System.out.println("Records Manipulated    :"+rowCounts[i]);
      }
      st.close();
      con.close();
   }

}
```

Batch Updations with PreparedStatement:

```
25.
 package com.durgasoft;
 import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;

public class JdbcApp34 {
   public static void main(String[] args)throws Exception {
      Class.forName("com.mysql.jdbc.Driver");
      Connection con=DriverManager.getConnection("jdbc:mysql://localhost:3306/durgadb",
"root","root");
      PreparedStatement pst=con.prepareStatement("insert into emp1 values(?,?,?,?)");
      pst.setInt(1,666);
      pst.setString(2, "FFF");
      pst.setFloat(3, 6000);
      pst.setString(4, "Hyd");
      pst.addBatch();

      pst.setInt(1,777);
      pst.setString(2, "GGG");
      pst.setFloat(3, 7000);
      pst.setString(4, "Hyd");
      pst.addBatch();
```

```
        pst.setInt(1,888);
        pst.setString(2, "HHH");
        pst.setFloat(3, 8000);
        pst.setString(4, "Hyd");
        pst.addBatch();
        int[] rowCounts=pst.executeBatch();
        for(int i=0;i<rowCounts.length;i++){
          System.out.println("Records Manipulated    :"+rowCounts[i]);
        }
        con.close();

    }

}
```

Note: If we include selection group SQL query in a batch then JVM will raise an Exception like ava.sql.BatchUpdateException: invalid batch command: invalid SELECT batch command.

Transaction Management:-

Transaction:- Transaction is an unit of work performed by the front end application on back end system.

1. Deposit some amount in an account.
2. Withdraw some amount from an account.
3. Transfer some amount from one account to another account.

In database applications, every transaction must satisfy the following 4 properties.

1. Atomicity
2. Consistency
3. Isolation
4. Durability

1. Atomicity:-

In general we are able to perform multiple number of operations in a particular transaction, where performing all the operations or performing none of the operations is called as Atomicity property. If we perform all the operations successfully in a transaction then the state of the transaction should be success.

If we perform none of the operations in a transaction then state of the transaction should be failure.

Note: While performing operations in a transaction, if we encounter a problem with any operation then we should perform rollback operation over all the operations which are available in the transactions.

2. Consistency:
In database applications, before the transaction and after the transaction the state of the database must be stable is called as Consistency property.

To achieve consistency we will perform some checkings called as Consistency Checkings.
Consistency checkings will check correctness of the results after the transactions.

3. Isolation:
In database applications, if we perform more than one transaction on a single data item then that transactions are called as Concurrent Transactions. In concurrent transactions, one transaction execution should not be effect to the another transaction. This nature of the transaction is called as Isolation.
While performing concurrent transactions, there may be a chance to get concurrency problems, to resolve these problems we will use Isolation Levels in database application.

4. Durability:
In database applications, after committing the transaction if we have any catastrophic failures like power failure, operating system crashing and so on then we have to preserve the modifications which we performed on the database during respective transaction. This nature of the transaction is called as Durability.

In Jdbc applications, when we establish connection then automatically that connection will have a default mode i.e. auto commit mode. In case of auto commit mode, when we submit SQL query to the connection, where connection will carry that SQL query to Database Engine and make the Database Engine to execute that SQL query and store the results on to the database table permanently.

The above auto commit nature of connection may not satisfy the transaction's atomicity property.
To preserve transaction atomicity property in Jdbc applications we have to change connection's auto commit mode.

To change connection's auto commit mode we have to use the following method from Connection.
public void setAutoCommit(boolean b)throws SQLException
If b==true then the connection will be in auto commit mode else the connection will be in non-auto commit mode.

Ex: con.setAutoCommit(false);

If we change connection's auto commit mode then we have to perform either commit or rollback operations to complete the transactions.
To perform commit and roll back operations we have to use the following methods from Connection.

public void commit()throws SQLException
public void rollback()throws SQLException

Note: In case of connection's non-auto commit mode, when we submit SQL query to the connection then connection will send that SQL query to Database Engine and make the Database Engine to execute that SQL query and store the results on to the database table temporarily. In this case, Database Engine may wait for commit or rollback signal from client application to complete the transactions.

26.

```
package com.durgasoft;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Statement;

public class JdbcApp40 {

    public static void main(String[] args) {
        Connection con=null;
        try {
            Class.forName("oracle.jdbc.OracleDriver");
            con=DriverManager.getConnection("jdbc:oracle:thin:
            @localhost:1521:xe","system", "durga");
            con.setAutoCommit(false);
            Statement st=con.createStatement();
            st.executeUpdate("insert into student values(111,'AAA',78)");
            st.executeUpdate("insert into student values(222,'BBB',87)");
            st.executeUpdate("insert into student values(333,'CCC',96)");
            con.commit();
            System.out.println("Transaction Success");
        } catch (Exception e) {
            try{
                con.rollback();
                System.out.println("Transaction Failure");
            }catch(Exception e1){
                e1.printStackTrace();
            }
            //e.printStackTrace();
        }
    }

}
```

27.

```
package com.durgasoft;
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Statement;

public class JdbcApp41 {

    public static void main(String[] args) {
        Connection oracle_conn=null;
        Connection mysql_conn=null;
        try {
            Class.forName("oracle.jdbc.OracleDriver");
            Class.forName("com.mysql.jdbc.Driver");
            oracle_conn=DriverManager.getConnection
            ("jdbc:oracle:thin:@localhost:1521:xe","system","durga");
                mysql_conn=DriverManager.getConnection
            ("jdbc:mysql://localhost:3306/durgadb","root","root");

            oracle_conn.setAutoCommit(false);
            mysql_conn.setAutoCommit(false);

            Statement oracle_st=oracle_conn.createStatement();
            Statement mysql_st=mysql_conn.createStatement();

            BufferedReader br=new BufferedReader
            (new InputStreamReader(System.in));
            System.out.print("Source Account     :");
            String source_Account=br.readLine();
            System.out.print("Target Account     :");
            String target_Account=br.readLine();
            System.out.print("Amount To Transfer  :");
            int trans_Amt=Integer.parseInt(br.readLine());

            int oracle_rowCount=oracle_st.executeUpdate
            ("update account set balance=balance-"+trans_Amt+" where
            accNo='"+source_Account+"'");
            int mysql_rowCount=mysql_st.executeUpdate("update account set
            balance=balance+"+trans_Amt+" where accNo='"+target_Account+"'");
            if((oracle_rowCount==1) && (mysql_rowCount==1)){
                oracle_conn.commit();
                mysql_conn.commit();
                System.out.println(trans_Amt+" Transfered Successfully from
                "+source_Account+" to "+target_Account);
                System.out.println("Transaction Success");
```

```
                    }else{
                            oracle_conn.rollback();
                            mysql_conn.rollback();
                            System.out.println("Transaction Failure");
                    }
                    } catch (Exception e) {
                    try {
                            oracle_conn.rollback();
                            mysql_conn.rollback();
                            System.out.println("Transaction Failure");
                    } catch (Exception e2) {
                            e2.printStackTrace();
                    }
                    }

        }

    }
```

Java.sql.SavePoint:-

One of the features introduced in jdbc3.0 version. And it is a interface Savepoint is a intermediate point and makes it possible to rollback the transaction upto the save point instead of roll backing the entire transaction.

To represent Savepoint Jdbc has provided a predefined interface java.sql.Savepoint.
To set a Savepoint we have to use the following method form Connection.

public Savepoint setSavepoint()

To perform rollback operation on set of instructions executive w.r.t a particular Savepoint we have to use the following method from Connection.

public void rollback(Savepoint sp)

To release Savepoint we will use the following method form Connection.

public void releaseSavepoint(Savepoint sp)

Note: In Jdbc applications, Savepoint concept could be supported by Type-4 Driver provided by Oracle, which could not supported by Type-1 Driver provided by Sun Microsystems.

Note: Type-4 Driver is able to support Savepoint up to setSavepoint() method and rollback(_) method, not releaseSavepoint(_) method.

28.

```
package com.durgasoft;
 import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Savepoint;
import java.sql.Statement;

public class JdbcApp42 {

    public static void main(String[] args) {
    Connection con=null;
    try {
            Class.forName("oracle.jdbc.OracleDriver");
            con=DriverManager.getConnection
            ("jdbc:oracle:thin:@localhost:1521:xe","system", "durga");
            con.setAutoCommit(false);
            Statement st=con.createStatement();
            st.executeUpdate("insert into student values(111,'AAA',76)");
            Savepoint sp=con.setSavepoint();
            st.executeUpdate("insert into student values(222,'BBB',88)");
            con.rollback(sp);
            st.executeUpdate("insert into student values(333,'CCC',99)");
            con.commit();
            System.out.println("Transaction Success");
            } catch (Exception e) {
            try {
                    con.rollback();
                    System.out.println("Transaction Failure");
            } catch (Exception e2) {
                    e2.printStackTrace();
            }
    }

    }
}
```


Stored Procedures And Functions:

What is the difference between Stored procedures and functions?
Ans:Stored procedure is a block of instructions defined at database to represent a particular action.
Stored procedures will not use return statement to return a value.

Syntax: create or replace procedure procedure_name([param-list])
as

```
------------
------------ Global declarations
------------
BEGIN
------------
------------ Database logic
------------
END procedure_name;
/ ( press enter to save and compile the procedure)
```

Stored function is a block of instructions defined at database to represent a particular action.Stored functions will use return statement to return a value.

Syntax: create or replace function function_name([param-list]) return data_type
as

```
------------
------------ Global declarations
------------
BEGIN
------------
------------ Database logic
------------
return value;
END function_name;
/ (press enter to save and compile the function)
```

In Jdbc applications, to access stored procedures and functions defined at database from Java application then we have to use CallableStatement object.

To represent CallableStatement object Jdbc API has provided the interface in the form of java.sql. CallableStatement.

If we want to use CallableStatement object in Jdbc applications then we have to use the following steps.

Step 1: Get CallableStatement object.

To create CallableStatement object in Jdbc applications we have to use the following method from Connection.

public CallableStatement prepareCall(String pro_cal)throws SQLException

Ex: CallableStatement cst=con.prepareCall("{call getSal(?,?)}");

When JVM encounters the above instruction JVM will pick up procedure call and send to Database Engine, where Database Engine will parse the procedure call and prepare a query plan with the positional parameters, as a result CallableStatement object will be created at Java application.

Note: In case of stored procedures and functions we are able to pass the parameters in the following 3 ways.

1. IN Type Parameter:

This parameter will get the value from procedure call or function call and make available to the procedure body or function body.

Syntax: var_name IN data_type

2. OUT Type Parameter:

This parameter will get the value from procedure body or function body and send that value to the respective procedure call or function call.

Syntax: var_name OUT data_type

3. INOUT Type Parameter:

This parameter is acting as both IN type and OUT type parameters.

Syntax: var_name INOUT data_type

Step 2: If we have IN type parameters in CallableStatement object then set values to IN type parameters.

To set values to IN type parameters we have to use the following method.

public void setXxx(int param_position, xxx value)

Where xxx may be byte, short, int and so on.

Ex: cst.setInt (1, 111);

Step 3: If we have OUT type parameter in CallableStatement object then we have to register OUT type parameter with a particular datatype.

To register OUT type parameter we will use the following method.

public void registerOutParameter(int param_position, int data_type)

Where data_type may be the constants from Types class like BYTE, SHORT, INTEGER, FLOAT and so on.

Ex: cst.registerOutParameter(2, Types.FLOAT);

Step 4: Make Database Engine to pick up the values from Query plan and to execute the respective procedure or function.

To achieve this we have to use The following method.

public void execute()throws SQLException

Ex: cst.execute();

Step 5: Get the values from OUT type parameters available in CallableStatement object.

After executing the respective procedure or function the respective values will be stored in OUT type parameters in CallableStatement object from stored procedure or functions. Toaccess the OUT type parameter values we have to use the following method.

public xxx getXxx(int param_position)

Where xxx may be byte, short, int and so on.

EX: float sal=cst.getFloat(2);

Ex:- execution of procedures

```
create or replace procedure getSal(id IN number, sal OUT number)
as
BEGIN
select esal into sal from emp where eno=id;
END getSal;
/
```

```
29.
 import java.sql.*;
public class JdbcApp34
{
public static void main(String[] args) throws Exception
{
Class.forName("oracle.jdbc.driver.OracleDriver");
Connection
con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","manager");
CallableStatement cst=con.prepareCall("{call getSal(?,?)}");
```

```
    cst.setInt(1,101);
    cst.registerOutParameter(2, Types.FLOAT);
    cst.execute();
    System.out.println("Salary....."+cst.getFloat(2));
    con.close();
    }
    }
```

Ex:- execution of functions

```
create or replace function getAvg(id1 IN number, id2 IN number) return number
as
sal1 number;
sal2 number;
BEGIN
select esal into sal1 from emp where eno=id1;
select esal into sal2 from emp where eno=id2;
return (sal1+sal2)/2;
END getAvg;
/
```

Ex:-

```
import java.sql.*;
public class Test
{
public static void main(String[] args) throws Exception
{
Class.forName("oracle.jdbc.driver.OracleDriver");
Connection
con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","manager");
CallableStatement cst=con.prepareCall("{?=call getAvg(?,?)}");
cst.setInt(2,888);
cst.setInt(3,6666);
cst.registerOutParameter(1, Types.FLOAT);
cst.execute();
System.out.println("Average Salary....."+cst.getFloat(1));
con.close();
}
}
```

```
/*
 create or replace procedure getEmps(sal IN number, emps OUT SYS_REFCURSOR)
 AS
 BEGIN
     open emps for
             select * from emp1 where esal<sal;
 END getEmps;
```

```
/
*/
package com.durgasoft;

import java.io.FileInputStream;
import java.sql.CallableStatement;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.util.Properties;

import oracle.jdbc.internal.OracleTypes;

    public class JdbcApp38 {

    public static void main(String[] args)throws Exception {
            FileInputStream fis=new FileInputStream("db.properties");
            Properties p=new Properties();
            p.load(fis);
            String driver_class=p.getProperty("driver_class");
            String driver_url=p.getProperty("driver_url");
            Class.forName(driver_class);
            Connection con=DriverManager.getConnection(driver_url,p);
            CallableStatement cst=con.prepareCall("{call getEmps(?,?)}");
            cst.setFloat(1, 10000);
            cst.registerOutParameter(2, OracleTypes.CURSOR);
            cst.execute();
            Object obj=cst.getObject(2);
            ResultSet rs=(ResultSet)obj;
            System.out.println("ENO\tENAME\tESAL\tEADDR");
            System.out.println("----------------------------");
            while(rs.next()){

    System.out.println(rs.getInt(1)+"\t"+rs.getString(2)+"\t"+rs.getFloat(3)+"\t"+rs.getString(4));
            }
            con.close();
    }

}
```

db.properties

```
driver_class=oracle.jdbc.OracleDriver
driver_url=jdbc:oracle:thin:@localhost:1521:xe
user=system
password=durga
```

```
/*
create or replace function getEmployees(no1 IN number,no2 IN number) return SYS_REFCURSOR
AS
employees SYS_REFCURSOR;
BEGIN
open employees for
    select * from emp1 where eno>=no1 and eno<=no2;
return employees;
END getEmployees;
/
*/
package com.durgasoft;

import java.io.FileInputStream;
import java.sql.CallableStatement;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.util.Properties;

import oracle.jdbc.internal.OracleTypes;

public class JdbcApp39 {

    public static void main(String[] args)throws Exception {
            FileInputStream fis=new FileInputStream("db.properties");
            Properties p=new Properties();
            p.load(fis);
            String driver_class=p.getProperty("driver_class");
            String driver_url=p.getProperty("driver_url");
            Class.forName(driver_class);
            Connection con=DriverManager.getConnection(driver_url, p);
            CallableStatement cst=con.prepareCall("{?=call getEmployees(?,?)}");
            cst.setInt(2, 111);
            cst.setInt(3, 555);
            cst.registerOutParameter(1, OracleTypes.CURSOR);
            cst.execute();
            ResultSet rs=(ResultSet)cst.getObject(1);
            System.out.println("ENO\tENAME\tESAL\tEADDR");
            System.out.println("------------------------");
            while(rs.next()){

    System.out.println(rs.getInt(1)+"\t"+rs.getString(2)+"\t"+rs.getFloat(3)+"\t"+rs.getString(4));
            }
            con.close();
    }
}
```

db.properties

```
driver_class=oracle.jdbc.OracleDriver
driver_url=jdbc:oracle:thin:@localhost:1521:xe
user=system
password=durga
```

Connection Pooling:

In general in Jdbc applications, when we have a requirement to perform database operations we will establish the connection with the database from a Java application, at the end of the application we will close the connection i.e. destroying Connection object.

In Jdbc applications, every time establishing the connection and closing the connection may increase burden to the Jdbc application, it will reduce the performance of the jdbc application.
In the above context, to improve the performance of Jdbc applications we will use an alternative called as Connection Pooling.

In Connection pooling at the time of application startup we will prepare a fixed number of Connection objects and we will keep them in a separate base object called Pool object.
In Jdbc applications, when we have a requirement to interact with the database then we will get the Connection object from Pool object and we will assign it to the respective client application.

At the end of the Jdbc application we will keep the same Connection object in the respective Pool object without destroying.
The above mechanism will improve the performance of the application is called as Connection Pooling.

If we want to implement Connection pooling in Jdbc application we have to use the following steps.

Step 1: Prepare DataSource object.

DataSource is an object, it is able to manage all the Jdbc parameter which are required to establish the connections.

To represent DataSource object Java API has provided a predefined interface i.e. javax.sql.DataSource.

DataSource is an interface provided by Jdbc API, but whose implementation classes are provided by all the database vendors.

With the above convention Oracle has provided an implementation class to DataSource interface in ojdbc6.jar file i.e. oracle.jdbc.pool.OracleConnectionPoolDataSource.

Ex: OracleConnectionPoolDataSource ds=new OracleConnectionPoolDataSource();

Step 2: Set the required Jdbc parameters to DataSource object.
To set the Jdbc parameters like Driver url, database username and password to the DataSource object we have to use the following methods.

public void setURL(String driver_url)
public void setUser(String user_name)
public void setPassword(String password)

Ex: ds.setURL("jdbc:oracle:thin:@localhost:1521:xe");
ds.setUser("system");
ds.setPassword("venkat");

Step 3: Get the PooledConnection object.

PooledConnection is an object provided by DataSource, it can be used to manage number of Connection objects.

To represent PooledConnection object Jdbc API has provided a predefined interface i.e. javax.sql.PooledConnection.

To get PooledConnection object we have to use the following method from DataSource.
public PooledConnection getPooledConnection()

Ex: PooledConnection pc=ds.getPooledConnection();

Step 4: Get Connection object from PooledConnection.

To get Connection object from PooledConnection we have to use the following method.
public Connection getConnection()

Ex: Connection con=pc.getConnection();

Step 5: After getting Connection prepare Statement or preparedStatement or CallableStatement and perform the respective database operations.

Ex: Statement st=con.createStatement();

```
import java.sql.*;
import javax.sql.*;
import oracle.jdbc.pool.*;
public class ConnectionPoolDemo
{
public static void main(String[] args) throws Exception
{
OracleConnectionPoolDataSource ds=new OracleConnectionPoolDataSource();
```

```
ds.setURL("jdbc:oracle:thin:@localhost:1521:xe");
ds.setUser("system");
ds.setPassword("durga");
PooledConnection pc=ds.getPooledConnection();
Connection con=pc.getConnection();
Statement st=con.createStatement();
ResultSet rs=st.executeQuery("select * from emp");
System.out.println("EID ENAME ESAL");
System.out.println("-----------------------------");
while (rs.next())
{
System.out.println(rs.getInt(1)+" "+rs.getString(2)+"
"+rs.getFloat(3));
}
}
}
```

Note: The above approach of implementing Connection pooling is suggestible up to standalone applications, it is not suggestible in enterprise applications. If we want to implement Connection pooling in enterprise applications we have to use the underlying application server provided Jdbc middleware server.

BLOB and CLOB:

BLOB (Binary Large Object):

Up to now in Jdbc applications, we are able to interact with the database in order to insert a record, retrieve a record and so on with the varchar data or number data and so on.

As per the application requirement if we want to insert an image or a document in the database table then Oracle provided datatypes numbers, varchar are not sufficient, we have to use BLOB and CLOB datatypes provided by Oracle.

The main purpose of the BLOB and CLOB datatypes is to represent large volumes of binary data and large volumes of character data in a database table.

To insert large volumes of binary data (an image) on to the database table we have to use the following steps.

Step 1: Prepare a table at database with blob data type.
Ex: create table emp_details(eno number, image blob);

Step 2: Represent an image file in the form of File class object.
Ex: File f=File("Desert.jpg");

Step 3: Get File class object content in the form of FileInputStream.
Ex: FileInputStream fis=new FileInputStream(f);

Step 4: Create preparedStatement object with insert SQL query format.

Ex: PreparedStatement pst=con.prepareStatement("insert into emp_details values(?,?)");

Step 5: Set BinaryStream to the blob type positional parameter in PreparedStatement.

To set a BinaryStream with the blob type positional parameter we have to use thefollowing method from PreparedStatement.

public void setBinaryStream(int param_index, InputStream is, int length);

Ex: pst.setBinaryStream(2, fis, (int)f.length());

Step 6: Execute PreparedStatement.

Ex: pst.executeUpdate();

```
import java.sql.*;
import java.io.*;
public class BLOBDemo
{
public static void main(String[] args) throws Exception
{
Class.forName("oracle.jdbc.driver.OracleDriver");
Connection
con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","manager");
File f=new File("Desert.jpeg");
FileInputStream fis=new FileInputStream(f);
PreparedStatement pst=con.prepareStatement("insert into employee values(?,?)");
pst.setInt(1,105);
pst.setBinaryStream(2,fis,(int)f.length());
pst.executeUpdate();
System.out.println("Employee Image inserted Successfully");
con.close();
}
}
```

Steps to retrieve blob data from database table to Jdbc application:

Step 1: Prepare ResultSet object with blob data.

Ex: ResultSet rs=st.executeQuery("select * from emp_details");

Step 2: Read normal data from ResultSet object.

Ex: rs.next();

int eno=rs.getInt(1);

Step 3: Get BinaryStream from blob datatype available at ResultSet object

To get a BinaryStream from blob type parameter available at ResultSet object we have to use the following method.

public InputSteram getBinaryStream(int param_index)

Ex: InputSteram is=rs.getBinaryStream(2);

Step 4: Prepare the target resource to hold up the retrieved blob data by using FileOutputStream.

Ex: FileOutputStream fos=new FileOutputStream("myimage.jpeg");

Step 5: Read bit by bit from InputStream and write the same bit by bit on FileOutputStream to store the retrieved data on target file.

Ex: int i=read();

```
while(i!=-1) {
fos.write(i);
i=is.read();
}
```

```
import java.sql.*;
import java.io.*;
public class BLOBDemo1
{
public static void main(String[] args) throws Exception
{
Class.forName("oracle.jdbc.driver.OracleDriver");
Connection
con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","durga"
);
Statement st=con.createStatement();
ResultSet rs=st.executeQuery("select * from employee");
rs.next();
System.out.println("Employee Id....."+rs.getInt(1));
InputStream is=rs.getBinaryStream(2);
FileOutputStream fos=new FileOutputStream("myimage.jpg");
int i=is.read();
```

```
while (i != -1)
{
fos.write(i);
i=is.read();
}
System.out.println("Image created Successfully with the name
myimage.jpeg");
fos.close();
con.close();
}
}
```

CLOB (Character Large Object):

clob is a datatype provided by Oracle, it can be used to represent large values of character data like documents, pdf files and so on.

If we want to perform operations with clob datatype then we have to use the same steps what we have used with blob datatype, but we need to provide the following replacements.

```
import java.sql.*;
import java.io.*;
public class CLOBDemo
{
public static void main(String[] args) throws Exception
{
Class.forName("oracle.jdbc.driver.OracleDriver");
Connection
con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","durga"
);
PreparedStatement pst=con.prepareStatement("insert into webinfo
values(?,?)");
pst.setString(1, "app");
File f=new File("web.xml");
FileReader fr=new FileReader(f);
pst.setCharacterStream(2, fr, (int)f.length());
pst.executeUpdate();
System.out.println("web application stored in database successfully");
fr.close();
con.close();
}
}
```

Ex:-

```
import java.sql.*;
import java.io.*;
public class CLOBDemo1
{
public static void main(String[] args) throws Exception
{
Class.forName("oracle.jdbc.driver.OracleDriver");
Connection
con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","durga"
);
Statement st=con.createStatement();
ResultSet rs=st.executeQuery("select * from webinfo");
rs.next();
System.out.println("Application name is....."+rs.getString(1));
FileWriter fw=new FileWriter("myweb.xml");
Reader r=rs.getCharacterStream(2);
int i=r.read();
while (i != -1)
{
fw.write(i);
i=r.read();
}
System.out.println("web.xml is retrieved successfully with the name
myweb.xml");
fw.close();
con.close();
}
}
```


JDBC INTERVIEW QUESTIONS:

1. What is the difference between Database and Database management system?
2. How a query could be executed when we send a query to Database?
3. What is Driver? How many Drivers are available in JDBC? What are the types?
4. What is JDBC and What are the steps to write a JDBC application?
5. How to load a JDBC driver?
6. How to establish a Database connection between java application and Database?
7. Basically Connection is an interface, how getConnection() will create an object for Connection interface?
8. What is the requirement to use Statement object?
9. How to execute SQL Queries from a java application?
10. What are the differences between executeQuery(...), executeUpdate(...) and execute(...) methods?
11. How to create a table dynamically from a jdbc application?.
12. How to insert records into a table from a JDBC application?

13. How to update a table from a jdbc application?.
14. How to delete records from a table from jdbc application?.
15.What is ment by ResultSet object and How to Fetch the Data from Database?.
16.Ingeneral execute() method can be used to execute selection group SQl queries for getting the data from Database , but execute() return a boolean value true so here how it possible to fetch the data from database?
17.Ingeneral execute() method can be used to execute updatation group SQl queries for updating the data on Database , but execute() return a boolean value false so here how it possible to get the records updated count value(int value)?
18. If we use selection group SQL query to executeUpdate() ,what happened?
19. If we use updatation group SQL query to executeQuery() ,what happened?
20. What is ment by ResultSet and What are the types of ResultSets are available in JDBC application?
21. What is the difference between ScrollSensitive ResultSet and ScrollInsensitive ResultSets?
22. What is the default ResultSet type in JDBC application and How it is possible to create a specific type of ResultSet object?
23. How to iterate the data from Scrollable ResultSet objuect in both forward and backword direction?
24. How to generate ScrollSensitive Result Set and how to reflect the later updations from database automatically to the ResultSet object?
25. How to insert records into Database throws Updatable ResultSet?
26. How to perform updations on Database throws Updatable ResultSet?
27. What is meant by ResultSetMetaData ?How to get The ResultSet metadata of a ResultSet object?
28. How to display the data with the respective field names
29. What are the differences between Statement and PreparedStatement? (or) Tell me the situations where we should go for PreparedStatement over Statement object.
30. Hhow to insert number of records into a table through Prepared Statement object.
31. How to update the database through PreparedStatement object.
32. How to fetch the data from database through PreparedStatement object.
33. What is meant by Transaction? How it is possible to maintain Transactions in JDBC applications?
34. What is meant by SavePoint?How to use Savepoints in JDBC applications?