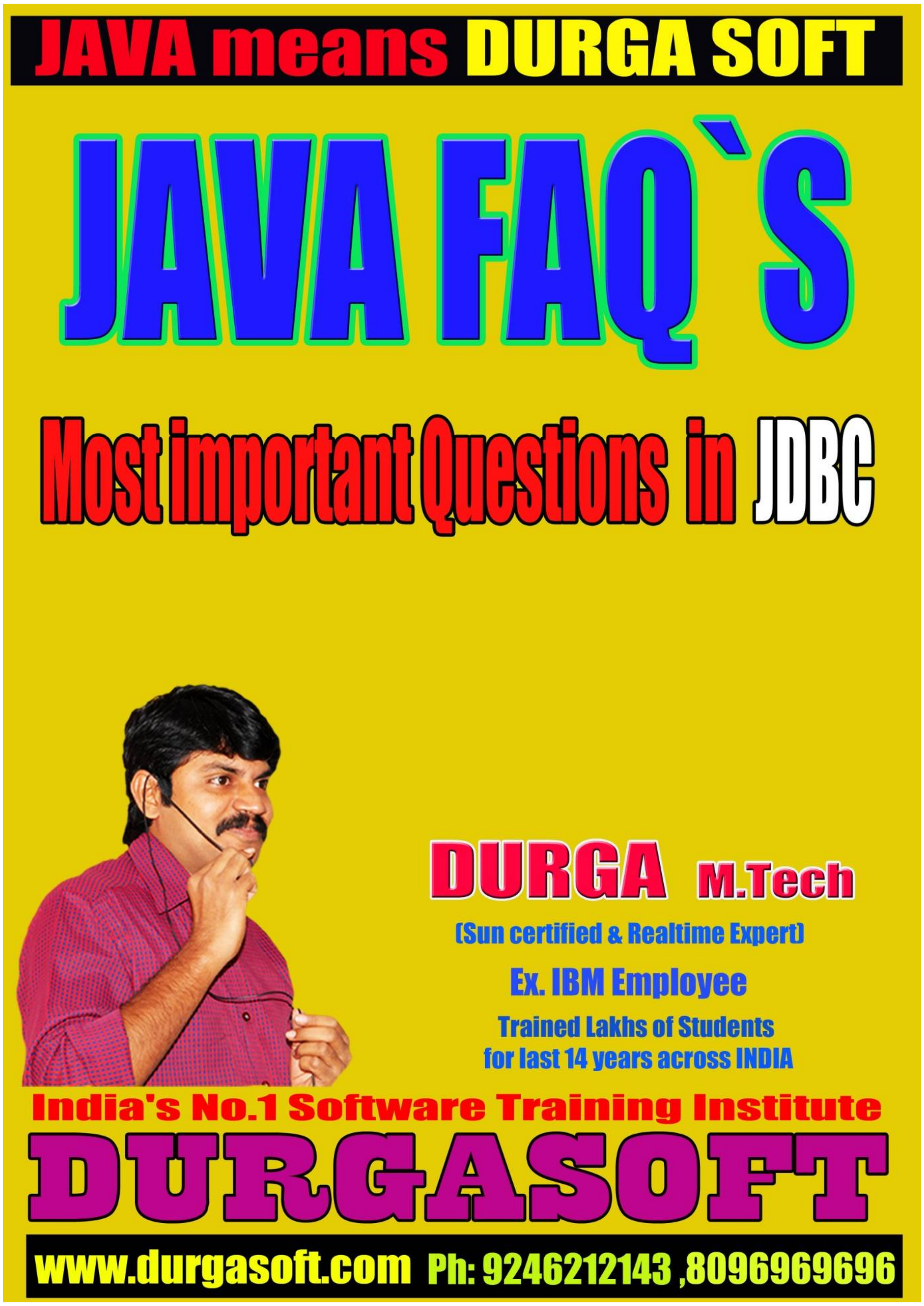
JAVA means DURGA SOFT
JAVA FAQ`S
Most important Questions in JDBC
DURGA M.Tech
(Sun certified & Realtime Expert)
Ex. IBM Employee
Trained Lakhs of Students
for last 14 years across INDIA
India's No.1 Software Training Institute
DURGASOFT
www.durgasoft.com Ph: 9246212143 ,8096969696

JDBC

1: **What is the difference between Database and Database management system?**
Ans: Database is a collection of interrelated data. Database management system is a software which can be used to manage the data by storing it on to the data base and by retrieving it from the data base. And DBMS is a collection of interrelated data and some set of programs to access the data.

There are 3 types of Database Management Systems.

- Relational DataBase Management Systems(RDBMS): It is a software system, which can be used to represents data in the form of tables. RDBMS will use SQL2 as a Queries language.
- Object Oriented DataBase Management Systems(OODBMS): It is a software system, which can be used to represent the data in the form of objects. This DBMS will use OQL as a Query language.
- Object Relational DataBase Management Systems(ORDBMS): It is a DBMS which will represents some part of the data in the form of tables and some other part of the data in the form of objects. This management system will use SQL3 as a Query Language, it is a combination of SQL2 and OQL.

2: How a query could be executed when we send a query to Database?
When we send an SQL Query from SQL prompt to the DataBaseEngine, then Database Engine will take the following steps.

- Query Tokenization: This phase will take SQL query as an input and devide into stream of tokens.
- Query Parsing: This phase will take stream of tokens as an input, with them it tried to construct a query tree. If query parser constructs query tree successfully then it was an indication that no grammatical mistakes in the taken SQL query. Otherwise there are some syntactical errors in the taken SQL query.
- Query Optimization: This phase will take query tree as an input and performs number of query optimization mechanisms to reduce execution time and memory utilization.
- Query Execution: This phase will take optimized query as an input and executes that SQL query by using interpreters internally as a result we will get some output on the SQL prompt.

3: What is Driver? How many Drivers are available in JDBC? What are the types?

- It is a process of interacting with the database from a java application.
- In JDBC applications we must specify the complete database logic in java application as for the java API representations, later on we need to send java represented database logic to the database engine(DBE).
- DBE must execute the database logic but it was configured as per the java representations but DBE able to understand only Query Language representations.
- At the above situation if we want to execute our database logic, we need to use one interface in between java application and the database, that interface must convert

java representations to query language representations and query language representations to java representations. Now this interface is called as a “Driver”.

Driver:

- It is a software or an interface existed in between a java application and database, which will map java api calls with query language api calls and vice versa.
- Initially sun Microsystems has provided “driver interface” to the market with this sun Microsystems has given an intimation to all the database vendors to have their own implementation as per their requirements for the Driver interface.
- As a response all the database vendors are providing their own implementation for the Driver interface inorder to interact with the respective databases from a java application.
- The users of the respective databases they must get the respective database provided Driver implementation from the database software and make use as part of the JDBC applications to interact with the respective databases form a java application.

Types of Drivers:

There are 180+ number of Drivers in the market. But all these Drivers could be classified into the following 4 types.

- Type 1 Driver
- Type 2 Driver
- Type 3 Driver
- Type 4 Driver

Type 1 Driver:

- o Type 1 Driver is also called as Jdbc-Odbc Driver or Bridge Driver.
- o Jdbc-Odbc Driver is an implementation to Driver interface provided by the sun Microsystems along with the java software.
- o Jdbc-Odbc Driver internally depends on the Microsoft product Odbc Driver.
- o Odbc is nothing but open database connectivity. It is a open specification which can be used to interact with any type of databases.

Advantages:

- This Driver is already available with java software that's why no need to bother about how to get the Driver implementation explicitily.
- Allmost all the databases could support this Driver.

Dis advantages:

- This Driver internally depends on Odbc Driver that's why it is not suitable for internet or web applications or network applications.
- This Driver is a slower Driver, why because Jdbc-Odbc Driver will convert java calls to Odbc calls. Then Odbc Driver has to convert Odbc calls to query language calls.
- This driver is not portable Driver why because it was not complete the java implementations in Jdbc-Odbc Driver.
- It we want to use Jdbc-Odbc Driver in our jdbc applications then we must require to install Odbc-Native Library.

Type 2 Driver:

Type 2 Driver is also called as "part java part native Driver". i.e., this Driver was designed by using some part of the java implementations and some other part of the database vendor provided native implementations. This Driver is also called as "native driver".

Advantages:
When compared to Type 1 driver it is efficient driver why because Type 2 driver directly will convert java api calls to database vendor api calls.

Dis advantages:

- If we want to use Type 2 Driver in our Jdbc applications then we must require to install database vendor native api.
- It is a costful Driver.
- It is not suitable for web applicadtions, distributed applications and web applications.
- Type 2 Driver performance is low when compared to Type 3 and Type 4 drivers.
- This driver is not portable driver. Why because this driver was not designed completely in java technology.

Type 3 Driver:

  - It is also called as middleware database access server driver.
  - This driver could be used to interact with multiple databases from the multiple clients.
  - This driver could be used in collaboration with application server.
  - This driver is suggestable for network applications.

Advantages:

- It is a fastest driver among all the drivers available in the market.
- To use Type 3 driver in our jdbc applications it is not required to install odbc native library and database native library.

- It is very much suitable for network applications.

Dis advantages:

- This driver is not suitable for simple jdbc applications.
- This driver requires minimum 3-Tier Architecture.
- When compared to Type1 and Type2 drivers.. Type3 driver is efficient and portable. But when compared to Type4 driver, Type3 driver is not portable.

Type 4 Driver:

- This driver is also called as pure java driver i.e, this driver was completely implemented by using java technology.
- When compared to Type1, Type2 and Type3 drivers.. Type4 driver is portable driver.
- Type4 driver can be used for any kind of applications.
- Type4 driver is a cheapest driver when compared to all the drivers that's why it is frequently used driver.

4: What is JDBC and What are the steps to write a JDBC application?

The process of interacting with the database from a java application is called as JDBC(Java Database Connectivity)
To interact with the database from a java application we will use the following five steps.

1. load and register the driver.
2. Establish a connection between java application and the database.
3. prepare either statement object or PreparedStatement object or CallebleStatement object as per the application requirements.
4. write and executer the sql queries.
5. terminate the connection which we have established.

5: How to load a JDBC driver?

- In general sun Microsystems has provided Driver interface for this all the database vendors has provided their own implementation.
- If we want to use the database vendor provided Driver implementation to our jdbc application, first we need to make the availability of the respective Driver's .class file to JVM, for this we need to set class path environment variable to the location where we have the driver implementation.
- Sun Microsystems is also provided an implementation to the Driver interface in the form of JdbcOdbcDriver class as part of the java software.
- If we want to use JdbcOdbcDriver in our jdbc applications no need to set class path environment variable. Why because it was already available in the java software's pre-defined library.
- JdbcOdbcDriver internally depends on the mocrosoft product Odbc driver. If we want to use the JdbcOdbcDriver in our jdbc applications first we must configure Odbc driver, for this we will use the following path.

Start/ conrlol panel / performance and maintenance / administrative tools / data source(Odbc)/ user dsn / click on Add / select microsofr Odbc for oracle / finish / provide data source name only / click on ok / ok.

- To load the driver's class byte code to the memory we will use the following method.

Public void forName(String class name)

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

Where forName() is a static method, which can be used to load the respective driver class byte code to the memory.

- Each and every driver class has already a static block at the time of loading the respective driver class byte code to the memory automatically the available static block could be executed, by this DriverManager.registerDriver(….) method will be executed as part of the static block.
- By the execution of the registerDriver(….) method automatically the specified driver will be register to the jdbc application.
- If you want to design any jdbc application, we need to use some pre-defined library, which was provided by the Jdbc API in the form of java.sql package, that's why we need to import java.sql package in our jdbc application.

Note:- The best alternative for Class.forName(..) is
DriverManagre.registerDriver(new sun.jdbc.odbc.JdbcOdbcDriver());
To register the driver.

6: How to establish a Database connection between java application and Database?

If we want to establish a connection between java application and the database we will the following piece of code.

Connection con=
DriverManager.getConnection("jdbc:odbc:nag","nag","system","manager");

Where getConnectin() is a static method from DriverManager class, which can be used to return connection object.

7: **Basically Connection is an interface, how getConnection() will create an object for Connection interface?**

Ans: Connection is an interface from java.sql package, for which getConnection(_) was return an anonymous inner class object of the Connection interface.

Note:- Anonymous inner class is a nameless inner class, which can be sued to provide an implementation either for the interfaces or for abstract classes.

Eg:
```
interface I
{
      void m1();
 }
Class Outer
{
            I  i = new I(){
                          public void m1()
                          {

                          }
                          public void m2()
                          {

}
                         }
   }
   Outer o = new Outer();
   o.i.m1();   à  correct
   o.i.m2();   à  wrong
```

getConnection(_) is a static method from DriverManager class, which will call internally connect() method, this connect() will establish a virtual socket connection in between the java application and the database.

8: What is the requirement to use Statement object?

- After establishing a connection between java application and the database we need to write the sql queries and we need to execute them.
- To execute the sql queries we will use some pre-defined library, which was defined in the form of Statement object, PreparedStattement object and CallableStatement object.
- As per the application requirements we need to create either Statement object or CallableStatement object and PreparedStatement object.
- To create Statement object dwe will use the following method from connection object.

public Statement createStatement()
Eg: Statement st = con.createStatement();
9***: How to execute SQL Queries from a java application?***

To execute the sql queries we will use the following methods from Statement object.

- st.executeQuery(…)
- st.executeUpdate(…)
- st.execute(…)

***10: What are the differences between executeQuery(…), executeUpdate(…) and execute(…)
methods?***
Ans: where executeQuery() can be used to execute selection group sql queries to fetch the data from database.
When we use selection group sql query with executeQuery() then JVM will send that sql query to the database engine, database engine will execute it, by this database engine(DBE) will fetch the data from database and send back to the java application.
Java is a purely object oriented technology. That’s why the jdbc application will maintain the fetched data from database, in the form of an object at heap memory, called as ResultSet object.

public ResultSet executeQuery(String sqlquery)

where executeUpdate() can be used to execute updation group sql query to update the database. When we provide updation group sql query as a parameter to executeUpdate(), then JVM will send that sql query to DBE, here DBE will execute it and perform updations on the database, by this DBE will identify the number of records got updated value called as “records updated count” and return back to the java application.

public int executeUpdate(String sqlquery)

where execute() can be used to execute either selection group sql queries or updation group queries.
When we use selection group sql query with the execute() then we will get ResultSet object at heap memory with the fetched data. But execute() will return “true” as a Boolean value.

When we use updation group sql query with execute() then we will get " records updated count value" at jdbc application. But execute() will return "false" as a Boolean value.

public boolean execute(String sqlquery)

11: How to create a table dynamically from a jdbc application?.

```
//import section
 import java.sql.*;
import java.io.*;

public class CreateTableEx
{
        public static void main(String[] args)throws Exception
        {
                    //create buffered reader object
           BufferedReader br = new
BufferedReader(new      InputStreamReader(System.in));

           //load and register the driver
           Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

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

           //create statement object
           Statement st = con.createStatement();

           //take table name as dynamic input
           System.out.println("Enter table name");
           String tname = br.readLine();

           //execute sql query
           St.executeUpdate("create table"+tname+"(eno number,ename varchar2(10),esal
number,eaddr varchar2(10))");
```

```
                System.out.println("table created successfully");

                //closing the connection
                con.close();
        }
}
```

12: How to insert records into a table from a JDBC application?

```
import java.sql.*;
import java.io.*;
public class InsertTableEx
{
   public static void main(String[] args) throws Exception
   {
            BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
            Class.forName("oracle.jdbc.driver.OracleDriver");
Connection
con  =              DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","syste
m","durga");
Statement st = con.createStatement();
while(true)
{
            System.out.println("Enter emp number");
            Int eno = Integer.parseInt(br.readLine());
            System.out.println("Enter emp name");
            String ename = br.readLine();
            System.out.println("Enter emp sal");
            Float esal = Float.parseFloat(br.readLine());
            System.out.println("Enter emp address");
            String eaddr = br.readLine();
st.executeUpdate("insert into emp1 values("+eno+",'"+ename+"',"+esal+",'"+eaddr+"')");
            System.out.println("read successfully inserted");
            System.out.println("one more record[y/n]);
            String option = br.readLine();
            If(option.equals("n"))
                        break;
}
      }
}
```

13: How to update a table from a jdbc application?.

```
import java.sql.*;
public class UpdateTableEx
{
   public static void main(String[] args)throws Exception
   {
            //load n register the driver in alternative way to Class.forName
            DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
Connection con =
DriverManager.getConnection(“jdbc:oracle:thin:@localhost:1521:xee”,”system”,”durga”);

Statement st = con.createStatement();
int updateCount = st.executeUpdate(“update emp1 set esal = esal+500 where esal<9000”);
System.out.println(“records updated……..”+updateCount);

con.close();
   }
}
```

14: How to delete records from a table from jdbc application?.

```
import java.sql.*;
public class DeleteTableEx
{
   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();
         int updateCount = sst.executeUpdate(“delete from emp3 where esal>=7000”);
         System.out.println(“records deleted………”+updateCount);

         con.close();
   }
}
```

15:What is ment by ResultSet object and How to Fetch the Data from Database?.

<u>ResultSet:-</u>

ResultSet is an Object which can be used to maintain the fetched data from database in the JDBC applications

When we execute a selection group sql query, either with executeQuety() or with execute() automatically a ResultSet object will be created at heap memory with the fetched data from database.

- To get the ResultSet object reference directly we will use executeQuery(..).
- When we create a ResultSet object automatically a cursor will be create called as “ResultSet cursor” to read the data from ResultSet object.
- When we create the ResultSet object by default ResultSet cursor will be created before the first record.
- If we want to read the data from ResultSet object every time we need to check whether the next record is available or not. If the next record is available automatically we need to move that ResultSet cursor to next record position.
- To perform this work we will use the following method from ResultSet interface.

```
public boolean next()
```

- After getting ResultSet cursor to a record position then we need to get the data from respective fields of the particular record, for this we will use following method.

```
public xxx getXxx(int fno)
        (or)
public xxx getXxx(String fname)
```

where xxx is byte, shor, int, long, float, double, char.

Eg:
```
while(rs.next())
        {
System.out.println(rs.getInt(1)+”   ”+rs.getString(2)+”   ”+rs.getFloat(3)+”   ”+rs.getString(4)
);
                }
```

The following example demonstrates how to fetch the data from database through ResultSet object.

```
import java.sql.*;
public class FetchEx
{
  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 rs = st.executeQuery("select * from emp1");
            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));
            }
    }
}
```

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?

- Execute() can be used to execute both selection group sql query and updation group sql query.
- If we use execute() to execute a selection group sql query then DBE(Database engine) will execute that sql query and send back the fetched data from database to java application. Now java application will prepare a ResultSet object with the fetched data but execute() will return "true" as a Boolean value.
- At this situation to get the reference of the ResultSet object explicitily, we will use the following method from Statement object.

```
public ResultSet getResultSet()
```

Eg:
```
boolean b = st.execute("select * from emp1");
System.out.println(b);
ResultSet rs = st.getResultSet();
```

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)?

- Execute() can be used to execute both selection group sql queries and updation group sql queries.
- If we use execute() to execute an updation group sql query then DBE will execute it and send back the records updated count value to the java application. But execute() will return "false" as a Boolean value. At this instance, to get the records updated count value explicitly we will use the following method from Statement object.

```
public int getUpdateCount()
```

Eg:
```
import java.sql.*;
public class FetchEx
{
```

```
                              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();
boolean b = st.execute("update emp1 set esal=esal+500 where esal<9000");
System.out.println(b);
int updateCount = st.getUpdateCount();j
System.out.println(updateCount);
                              }
                    }
```

18: If we use selection group SQL query to executeUpdate() ,what happened?

- If we use selection group sql query as a parameter to executeUpdate(…) then JVM will send that sql query to the DBE, DBE will fetch the data and send back to the java application here java application will store the fetched data in the form of ResultSet object. But executeUpdate() is expecting records updated count value.

  Due to this contradiction JVM will rise an exception like java.lang.SQLException.

If we handle the above exception properly then we will get ResultSet abject and we will get the data from Database

```
import java.sql.*;
class Test
{
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();
boolean b = st.executeUpdate("select * from emp1");
        }
        catch(Exception e)
        {
                ResultSet rs=st.getResultSet();
        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));
        }

                e.printStackTrace();
        }

}
```

19: If we use updatation group SQL query to executeQuery() ,what happened?

- If we use updation group sql query as a parameter to executeQuery() then JVM will send that sql query to the DBE, DBE will perform updations on the database and send back records updated count value to the java application. But here executeQuery() is expecting ResultSet object reference.

  Due to this contradiction JVM will rise an exception like java.lang.SQLException.

```
import java.sql.*;
class Test
{
public static void main(String[] args)
{
        Statement st=null;
        try
        {
                Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

              DriverManager.getConnection("jdbc:odbc:nag","system","durga");

                st = con.createStatement();
 boolean b = st.executeQuery("update  emp1 set esal=esal+1000 where       esal <10000");
        }
        catch(Exception e)
        {
                int count=st.getUpdateCount();
                System.out.println(count);

                e.printStackTrace();
        }

}
```

20: What is ment by ResultSet and What are the types of ResultSets are available in JDBC application?

In jdbc applications ResultSets could be classified in the following two ways.

- **<u>On the basis of ResultSet privilizations (Concurancy):-</u>**

-

There are 2 types of ResultSets.

- Read only ResultSet
- Updatable ResultSet

<u>Read only ResultSet:</u>- It is a ResultSet, which will allow the users to read the data only. To refer this ResultSet, we will use the following constant from ResultSet interface.

```
public static final int CONCUR_READ_ONLY;
```

<u>Updatable ResultSet:-</u> If is a ResultSet object, which will allow users to perform some updations on its content. To refer this ResultSet we will use the following constant from ResultSet interface.

```
public static final int CONCUR_UPDATABLE;
```

<u>2)On the basis of the ResultSet cursor movement:-</u>

There are 2 types of ResultSets.

- Forward only ResultSet
- Scrollable ResultSet

<u>Forward only ResultSet</u>:- It is a ResultSet object, which will allow the users to iterate the data in any forward direction. To refer this ResultSet object we will use the following constant from ResultSet interface.

```
public static final int TYPE_FORWARD_ONLY;
```

Scrollable ResultSet:- These are the ResultSet objects, which will allow the users to iterate the data in both forward and backward directions.

There are 2 types of Scrollable ResultSets.

- Scroll sensitive ResultSets
- Scroll in sensitive ResultSets.

21: What is the difference between ScrollSensitive ResultSet and ScrollInsensitive ResultSets?

Ans: Scroll sensitive ResultSet is a ResultSet object, which will allow the later updations from database automatically after creating it. To refer this ResultSet we will use the following constant.

public static final int TYPE_SCROLL_SENSITIVE;

Scroll insensitive ResultSet is a ResultSet object, which will not allow later updations from database after creating it. To refer this ResultSet we will use the following constant from ResultSet interface.

- public static final int TYPE_SCROLL_INSENSITIVE;

22:What is the default ResultSet type in JDBC application and How it is possible to create a specific type of ResultSet object?

- The default ResultSet type in the jdbc applications is Read only and forward only.
- In jdbc applications we are able to specify the following types of the ResultSet combination to any particular ResultSet.
    - read-only, forward only
    - read-only, scroll sensitive
    - read-only, scroll insensitive
    - updatable, forward only
    - updatable, scroll sensitive
    - updatable, scroll insensitive

- if we want to specity a particular type to the ResultSet object then we should use either of the above constants combination as a parameter to createStatement() method, for this we will use the following method.

public Statement createStatement(int forward / ScrollSensitive / ScrollInsensitive, int readonly / updatable)

Eg: Statement st = con. createSensitive(ResultSet.TYPE_SCROLL_SENSITIVE, ResultSet.CONCUR_UPDATABLE);
ResultSet rs = con.executeQuery(….);

23:How to iterate the data from Scrollable ResultSet objuect in both forward and backword direction?

- to iterate the data in forward direction from a ResultSet object we will use the following 2 methods.

public Boolean next()
public xxx getXxx(int fieldno.)
Where xxx may be byte, short, char, int, long, float, double.

- To iterate the data in backward direction from Scrollable ResultSet object we will use the following 2 methods.

public Boolean previous()
public xxx getXxx(int fieldno)
Where previous() is a Boolean method, which can be used to check whether the previous record is available or not, if it is available then cursor will be moved to previous record position.

The following example demonstrates how to iterate the data in both forward and backward direction from the ResultSet object

```
import java.sql.*;
public class ScrollResEx
{
        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_SENSEITIVE,ResultSet.CONCUR_UPDA
TABLE);
ResultSet rs = st.executeQuery("select * from emp1");
System.out.println("data in forward direction");
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.in.read();
System.out.println("data in backward direction");
System.out.println("ENO      ENAME      ESAL      EADDR");
System.out.println("**********************************");
While(rs.previous())
{
System.out.println(rs.getInt(1)+"    "+rs.getString(2)+"    "+rs.getFloat(3)+"    "+rs.getString(
4));

}
}
}
```

24: **how to generate ScrollSensitive Result Set and how to reflect the later updations from database automatically to the ResultSet object?**

```
import java.sql.*;
public class Test
{
        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_SENSEITIVE,ResultSet.CONCUR_UPDA
TABLE);
ResultSet rs = st.executeQuery("select * from emp1");
rs.next();
System.out.println("old salary emp111……."+rs.getFloat(3));
System.in.read();//application is in pause, perform database updations
Rs.refreshRow();
System.out.println("new salary of emp111……..”+rs.getFloat(3));
}
}
```

Where refreshRow() is a method from Scrollable ResultSet object, which can be used to refresh the current row in the ResultSet object to allow the later updations from database. Prototype of this method is

public void refreshRow()

25: **How to insert records into Database throws Updatable ResultSet?**

If we want to insert a new record on to the database through Updatable ResultSet object, we will use the following steps.

Step1: Get the Updatable ResultSet object with fetched data.

Step2: Move ResultSet cursor to the end of the ResultSet object, where we need to take a buffer to hold new records data temporarily, for this we use the following method from updatable ResultSet object.

public void moveToInsertRow()

Step3: Insert new records data on to the buffer temporarily at Updatable ResultSet object for this we will use the following method format.

public void updateXxx(int fieldno,xxx value)

Where xxx may be byte, short, int, char, double, float, long.

Step4: Make the temporary insertion as the permanent insertion in Updatable ResultSet object as will as in database, for this we will use the following method.

public void insertRow()

The following example demonstrates how to insert no. of records onto the database through Updatable ResultSet objects.

```
import java.sql.*;
import java.io.*;
public class UpdateResEx
{
        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_SENSEITIVE,ResultSet.CONCUR_UPDA
TABLE);
ResultSet rs = st.executeQuery(“select * from emp1”);
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
rs.moveToInsertRow();
                while(true)
                {
                        System.out.println("enter employee number");
                        int eno = Integer.parseInt(br.readLine());
                        System.out.println("enter employee name");
                        String ename = br.readLine();
                        System.out.println("enter employee salary");
                        float esal = Float.parseFloat(br.readLine());
                        System.out.println("enter 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("record successfully inserted");
                        System.out.println("one more record[y/n]);
                        String option = br.readLine();
                        if(option.equals("n"))
                                break;
}
}
```

26: How to perform updations on Database throws Updatable ResultSet?

By using Updatable ResulSet object we are able to perform some updations on to the database. To perform updations on to the database through Updatable ResultSet object we will use the following steps.

Step1: Get the Updatable ResultSet objectd with the fetched data.

Step2: Move ResultSet cursor to a record where we want to perform updations, for this we will use the following method.

public void absolute(int recordno.)

Step3: Perform Temporary updations on to the particular record, for this we will use following method.

public void updateXxx(int fieldno,xxx value)

Step4: Make the temporary updation as a parmenent updation on to the Updatable ResultSet object as well as to the database. For this we will use the following method.

public void updateRow()

The following example demonstrates how to perform updations on to the database through Updatable ResultSet object.

```
import java.sql.*;
public class UpdateResEx1
{
        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_SENSEITIVE,ResultSet.CONCUR_UPDA
TABLE);
ResultSet rs = st.executeQuery("select * from emp1");
rs.absolute(3);
float newsal = rs.getFloat(3)+500;
rs.updateFloat(3,newsal);
rs.updateRow();
                    }
           }
```

27:what is meant by ResultSetMetaData ?How to get The ResultSet metadata of a ResultSet object?

Data about the data is called as Metadata. Similarily data about the data available in ResultSet object called as "ResultSet Metadata".

- ResultSet Metadata includes the number of columns of a table in ResultSet object, all the column names, column datatypes and the column display sizes.
- To get the ResultSet Metadata object we will use the following method from ResultSet object.

public ResultSetMetaData getMetaData()

- To get the number of columns available in ResultSet object we will use the following method from ResultSetMetaData object.

public int getColumnCount()

- To get the name of a particular column, we will use the following method.

public String getColumnName(int fieldno)

- To get the column datatype of a particular column, we will use the following method

public String getColumnTypeName(int fieldno)

- To get the column display size of a particular column we will use the following method.

public int getColumnDisplaySize(int fieldno)

The following example demonstrates how to get ResultSetMetaData from a ResultSet object

```
import java.sql.*;
public class ResultSetMD
{
        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 rs = st.executeQuery(“select * from emp1”);
ResultSetMetaData rsmd = rs.getMetaData();
int count = rsmd.getColumnCount();
System.out.println(“number of columns......”+count);
for(int i=1;i<=count;i++)
{
System.out.println(rsmd.getColumnName(i)+”  “+rsmd.getColumnTypeName(i)+”  “+rsmd.
getColumnDisplaySize(i));
System.out.println()
                        }
                }
        }
```

***28:* how to display the data with the respective field names**

```
import java.sql.*;
public class RSMD1
{
        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_SENSEITIVE,ResultSet.CONCUR_UPDA
TABLE);
ResultSet rs = st.executeQuery(“select * from emp1”);
                        ResultSetMetaData rsmd = rs.getMetaData();
System.out.println(rsmd.getColumnName(1)+”    “+rsmd.getColumnName(2)+”   “+rsmd.ge
tColumnName(3)+”      “+rsmd.getColumnName(4));
System.out.println(“*******************************”);
while(rs.next())
{
System.out.println(rs.getInt(1)+”   “+rs.getString(2)+”   “rs.getFloat(3)+”   “+rs.getString(4
));
                              }
                    }
          }
```

***29:* What are the differences between Statement and PreparedStatement?**
(or)
Tell me the situations where we should go for PreparedStatement over Statement object.
Ans:

- When we have a requirement to execute same kind of sql query in the next sequence then we should go for PreparedStatement over Statement object.
- For the above requirement if we use Statement object, every time execution of the same sql query DBE must perform query tokenization, query parsing, query optimization and query execution.
- This approach will increase burden to the DBE. To reduce burden to the DBE we should go for an alternative. That is PreparedStatement over Statement object.
- For the same requirement if we use PreparedStatement object then for our complete requirement DBE will go for only one time query parsing (tokenization, parsing, optimization and execution);

If we want to use PreparedStatement object for the above requirement then
we will use following steps.
Step1: Prepare PrepareStatement object by providing generalized sql query format with the required number of parameters, for this we will use the following method from Statement object.

```
public PreparedStatement prepareStatement(String  sqlqueryformat)
```

Eg: PreparedStatement pst = con.prepareStatement(“insert into emp1   values(?,?,?,?)”);

When JVM encounters above instruction jvm will pickup specified generalized sql query format and send to the DBE, here DBE will process query format only one time and prepare a Buffer with the specified parameters, called as “query plan”. As a result PreparedStatement object will be created with the parameters at java side.

<u>Step2:</u> Set the values to parameters in PreparedStatement object. After getting PreparedStatement object with parameters, we need to set some values to perform an operation, for this we will use the following method.

```
public void setXxx(int parano,xxx value)
```

where xxx may be byte, short, char, int, long, float, double.

Eg:
```
pst.setInt(1,111);
pst.setString(2,”abc”);
```

When JVM encounters the above method then jvm will set the specified values to the specified parameters at the PreparedStatement object, intern that parameter values could be reflected to query plan.

<u>Step3:</u> Given an intimation to DBE to perform the respective operation. After setting the values to the parameters we should give an intimation to the DBE explicitly pickup the values from query plan and perform the operation specified in generalized sql query format, for this we will use the following methods.

- If the generalized sql query belongs to selection group then we will use following method from PreparedStatement object

```
public ResultSet executeQuery(…)
```

- If the generalized sql query belongs to updation group then we will use the following method.

```
public int executeUpdate(…)
```

***30:* Hhow to insert number of records into a table through Prepared Statement object.**

```
import java.sql.*;
import java.io.*;
public class PreparedInsertEx
{
        public static void main(String[] args)throws Exception
        {
                Class.forName(“sun.jdbc.odbc.JdbcOdbcDriver”);
Connection con = DriverManager.getConnection(“jdbc:odbc:nag”,”system”,”durga”);
PreparedStatement pst= con.prepareStatement(“insert into emp1 values(?,?,?,?)”);
BufferedReader br= new BufferedReader(new InputStreamReader(System.in));
while(true)
{
;              }
        }
```

31: **how to update the database through PreparedStatement object.**

```
import java.sql.*;
public class PreparedUpdateEx
{
        public static void main(String[] args)throws Exception
        {
                Class.forName(“sun.jdbc.odbc.JdbcOdbcDriver”);
Connection con = DriverManager.getConnection(“jdbc:odbc:nag”,”system”,”durga”);
PreparedStatement pst = con.prepareStatement(“update emp1 set esal = esal+? Where
esal<?”);
Pst.setInt(1,500);
Pst.setFloat(2,10000.0f);
Int count = pst.executeUpdate();
System.out.println(“no. of records updated:”+count);
                }
        }
```

32:**how to fetch the data from database through PreparedStatement object.**

```
import java.sql.*;
public class UpdateResEx
{
        public static void main(String[] args)throws Exception
        {
                Class.forName(“sun.jdbc.odbc.JdbcOdbcDriver”);
Connection con = DriverManager.getConnection(“jdbc:odbc:nag”,”system”,”durga”);
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));
                        }
                }
        }
```

33:What is meant by Transaction? How it is possible to maintain Transactions in JDBC applications?

- Transaction is nothing but an unit of work performed by the applications.
- Every transaction should have the following properties.
- Atomicity
- Consistency
- Isolation
- Durability

- Where atomicity is nothing but perform all the operations or not to perform all the operations in a transaction. That is every transaction must be in either success state or failure state.
- As part of the jdbc applications when we establish a connection automatically the connection should have a default nature called as "auto commit".
- Auto commit in the sense when we send an sql query to the connection then connection will carry that to the DBE and make the DBE to execute provided sql query and store the results on the database permanently.
- The connections default auto commit nature violates the transactions atomicity property.
- To preserve transactions atomicity property we should change the connections auto commit nature to non-auto commit nature, for this we will use the following method.

```
Public void setAutoCommit(Boolean b)
Where b=true    connection is in auto commit
And b=false     connection not in auto commit.
```

- If we use connections non auto commit nature in our jdbc applications then we must use either commit or rollback operations explicitily as part of the transactions.

```
Public void commit()
Public void rollback()
```

The following example demonstrates how to maintain the transactions with atomicity property in the jdbc applications.

```
import java.sql.*;
public class TransactionEx
{
    public static void main(String[] args)throws Exception
    {
            Connection con = null;
            try
            {
                    Class.forName("sun.jdbc.odbd.JdbcOdbcDriver");
Con = DriverManager.getConnection("jdbc:odbc:nag","system","durga");
con.setAutoCommit("false");
Statement st = con.createStatement();
st.executeUpdate("insert into emp1 values(888,'fff',8000,'hhh')");
st.executeUpdate("update emp1 set esal = esal-500 where esal>= 'abc' ");
st.executeUpdate("delete emp1 where esal<7000");
con.commit();
                    }
                    catch(Exception e)
                    {
                              con.rollback();
                              System.out.println(e);
                    }
          }
     }
```

34:What is meant by SavePoint?How to use Savepoints in JDBC applications?

- Save point is a concept introduced by jdbc 3.0 which can be used to block a set of instructions execution in the transactions committing operation.
- To set a save point we will use the following method.

public SavePoint setSavePoint()

- To block a set of sql queries execution prior to the save point we will use the following method.

public void rollback(savepoint s)

- To release a savepoint we will use the following method

public void releaseSavePoint();

- SavePoint concept could not be supported be type1 driver, it could be supported by type4 driver.
- Even type 4 driver is supporting up to setSavePoint() and rollback() , not releaseSavepoint();

Eg:

```
      import java.sql.*;
public class SavePointEx
{
      public static void main(String[] args)throws Exception
      {
                  Connection con = null;
                  try
                  {
                              Class.forName("oracle.jdbc.driver.OracleDriver");
con =
DriverManager.getConnection("jdbc:oracle:thin:@locajhost:1521:xe","system","durga");
con.setAutoCommit("false");
Statement st = con.createStatement();
st.executeUpdate("insert into emp1 values(111,'fff',8000,'hhh')");
savepoint sp= con.Savepoint();
st.executeUpdate("insert into emp1 values(222,'ggg',7000,'iii') ");
con.rollback(sp);
st.executeUpdate("insert into emp1 values(333,'hhh',9000,'jjj')");
con.commit();
                        }
                        catch(Exception e)
                        {
                                    con.rollback();
                                    System.out.println(e);
                        }
            }
      }
```