UNIT-II

I.J2EE DATABASE CONCEPTS:

1)Data
2)Database.
3)Database Schema.
4)The art of Indexing.

II. JDBC OBJECTS

1)JDBC Packages.
2)Concept of JDBC.
3)JDBC Driver Types.
4)OverView of JDBC process.
5)DB Connection.
6)Statement Object.
7)ResultSet
8)MetaData
9)Transaction Processing.

III. JDBC AND EMBEDDED SQL

1) Model Programs
2) Tables.
3) Inserting Data into Tables.
4) Selecting Data from a table
5) Deleting Data from table.
6) Updating tables
7) Joining Tables
8) Calculating Data.
9) Grouping and Ordering Data.
10)Metadata
11)View.
12)Indexing.

J2EE DATABASE CONCEPTS

1. **Data:-** Data is nothing but facts and statics stored or free flowing over a network,generally it's raw and unprocessed.
   For example: when you visit any website, they might store your IP address , that is data.
   Data becomes information when it is processed,turning it into something meaning ful.

2**. DataBase:-** A Database is a collection of related data organised in a way that data can be easily accessed,managed and updated.

3. DataBase Schema:- A database schema is a document that defines all components of a database such as tables,columns, and indexes. A database schema also shows relationship between tables , which are used to join together rows of two tables. There are six steps that must be performed to create database schema. These are

a) Identifying information used in existing system.
b) Decompose this information into data.
c) Define data.
d)Normalize data into logical groups.
e)Create primary key and foreign key.
f)Group data together into logical groups.

4. Art Of the Indexing:- The DBMS always uses an index, if one exists, to locate search criteria. If an index does not match the search criteria, then DBMS either creates a temporary index as part of the search or sequentially searches the table. The method used with your DBMS depends on the how the manufacturer designed the DBMS.

AN Index is used to quickly locate information in table similar to how information is located in a book. Conceptually An index is table that has two columns. One column is key to the index and other column is number of row in corresponding table that contains the value of key. An Index is created using primary key as Index key. Indexed keys are sorted in alphabetical or numerical order depending on the value of key.

The DBMS begins at the center of index when searching for a particular key value. The search criteria is either an exact match to key value of center row of index, is greater than key value or less than key value. If there is match, then row that corresponds to key is retrieved, If value is greater than key , then DBMS begins the next search at the center row of the lower half of the index. Likewise, if the value is less than key, the search continues at the center row of upper half of the index. The process is repeated until either a match is found or search criteria can not be found.

Index

Customer No	Row Number
123a	1
125b	3
342c	2

Customer Table

RowNo	Customer No	Firstname	LastName
1	123a	Bob	Smith
2	342b	Marry	Jones
3	125b	Tim	Jones

4.1)DrawBacks Using an Index:- There is a drawback when too many indexes are used with one table. That is, an unacceptable delay can occur whenever a row is inserted into or deleted from the table.

Once an index is built , the DBMS is responsible for automatically maintain the index whenever a row is inserted or deleted from the table. This means that each index associated with table must be modified whenever a row is inserted or deleted from the table, which can cause performance degradation if a table is associated with many indexes.

4.2)Clustured keys:- A clustered key is index key whose value represents data contained in multiple columns of the corresponding table.

Ex:- Customer name is used as index key. The customer name is composed of two data elements:

Customer first name and customer last name.

4.3)Derived key:- A derived key consists of value that represents part of value of column rather than entire value of the column.

The order number is comprised of 3 components. The first component is sales region. The second component is sales representatives number. The last component is unique number that identifies the order.

The Order number appears in one column of order table. The index key is first component.

4.4)Exact Matches and partial Matches:- An exact match requires that all the characters of search criteria match index key. If single character is mismatched, then DBMS does not return data in corresponding row of table.

A partial match requires that some-not all – characters of index key match the search criteria. That is , if the first character of the search criteria and index key are the same and remaining characters are different, the DBMS stills considers it a match and returns data in corresponding row of the table.

JDBC OBJECTS

1.JDBC Packages:- JDBC API provides mainly two packages.

1.1)java.sql.
1.2)javax.sql.

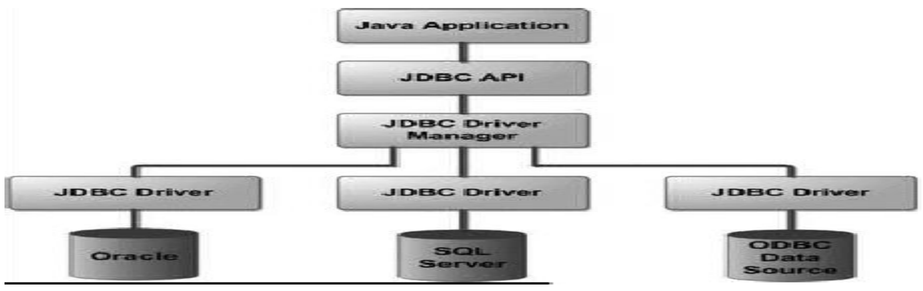
1.1)java.sql:- This package contains basic classes and interfaces.

Class Name	Interface Name
DriverManager	Driver
Time	Connection
Date	Statement
TimeStamp	PreparedStatement
Types	CallableStatement
	ResultSet
	DatabaseMetadata
	ResultSetMetadata
	ParameterMetadata
	Savepoint
	ROWID

1.2)javax.sql:- This package contains advanced classes and interfaces.

Class Name	Interface Name
ConnectionEvent	DataSource
StatementEvent	StatementEventListener
RowsetEvent	ConnectionEventListener
	RowsetListener
	Rowset

2. Concept of JDBC:- JDBC is an JAVA API(application programming Interface) . It is a collection of classes and interfaces. It is used to connect java application with database. Using JDBC, the application program can interact with various type of database such asOracle, MySql, SQL Server …etc.



2.1) JDBC API:- see JDBC Packages for JDBC API.

2.2) DriverManager:- The DriverManager is class. It is available in java.sql package. It manages all registered driver software list.

It has methods to register and unregister the driver software .

1. Public static void registerDriver(Driver drivername)
   This method register driver software at drivermanager.
2. Public static void unRegisterDriver(Driver drivername)
   This method deregister driver software from driver manager.(i.e) Driver software is removed from list.

Driver Manager establish connection between java application and data base with help of driver software.

1. Public static Connection getConnection(String database-url) throws SQLException
2. Public static Connection getConnection(String database-url,String uname,String pwd)throws SQLException.

Where URL represents URL of Database. URL is changed from driver to driver.

Driver Type	URL for Oracle DB
Type-1	Jdbc:odbc:demondsn
Type-2	1.jdbc:oracle:oci8:@DSID. 2.jdbc:oracke:oci:@DSID.
Type-3	
Type-4	Jdbc:oracle:thin:@localhost:1521:DSID

Where DSID is Database Unique System ID. We can find the DSID of our database in following way:

- Select * from global_name;

2.3) Driver Software:-

->Driver:- It is interface. It is available in java.sql package. It act as requirement specification to implement Driver class.

->Driver class:- It is implementation of Driver Interface.

EX:-

Driver Type	Class Name
Type-1	Sun.jdbc.odbc.jdbcOdbcDriver
Type-2	1.Oracle.jdbc.driver.OracleDriver 2.oracle.jdbc.OracleDriver
Type-3	Id.sql.IDSDriver
Type-4	1.Oracle.jdbc.driver.OracleDriver 2.oracle.jdbc.OracleDriver

Every Driver software is identified with driver class. This class name is vary from driver to driver.

The collection of implementation classes of various interfaces present in JDBC API is called Driver Software. It is responsible to convert java calls to database specific calls and database specific calls into java calls. Usually ,Driver software is available in form of jar files.

Example :- **ojdbc14.jar,odbc6.jar, ojdbc7.jar,mysql-connector.jar**.

2.3.1) Driver software providers:- The Driver softwares can be provided by following vendors.

- java vendor(upto java 1.7)
- Database Vendor(Oracle,Microsoft,..etc)
- Third party vendors.

EX:- Inet is third party vendor and provided several software for different databases.

- Inet ORacxo for oracleDatabase.
- Inet Merlia for Microsoft SQL server.
- Inet sybelux for Sybase Database.

Note:- Everybody recommends to use Database vendor provided Driver Software.

While developing driver software , Vendor use either java or java with other native languages like c, c++.

If driver software is developed using only java language, then such type of driver software is called “Pure java driver”.

If driver software is developed using java and native languages, then such type of driver software is called “partial java driver”.

2.3.2) Driver Types:-

Based on Functionalities and similarities , we divide the driver software into four categories.

- ➔ Type1 Driver.
- ➔ Type2 Driver.
- ➔ Type3 Driver.
- ➔ Type4 Driver.

Q) which driver should be used?

A) If one type of db is being used in our application then it is recommended to use type-4 driver.

EX:- Stand alone applications, small scale web aplications.

If multiple databases are being used in our application then Type-3 driver is recommended to use.

Ex:- Large scale web applications and enterprise applications.

If type4,type3 driver is not available then you use either type-2 driver.

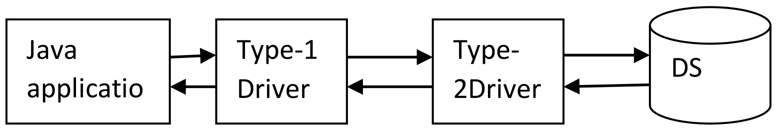
If no driver is available,then only use type-1 driver.

2.4)data sources:- The Data sources processes commands from driver and returns result.

3.Driver Types:- A driver is collection of Implemented classes of interfaces of JDBC API. Driver translates the ODBC or JDBC calls into DBMS-specific calls as well as translates DBMS specific calls into JDBC calls. The driver establishes connection with data source. After establishing connection, driver submits specific calls to data sources. Driver receives results from the source and returning them to application.

Drivers in JDBC are classified into four types depending on the architectural relationship between the application and the data source:

3.1)Type-1 Bridge:- This type of driver translates JDBC function calls into ODBC function calls that is not native to DBMS.

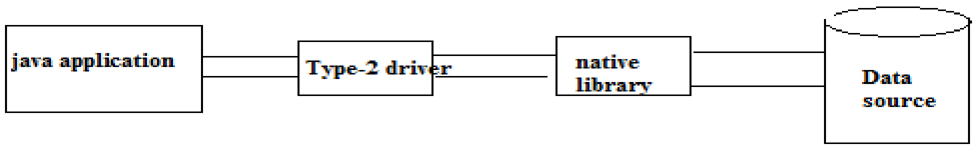


Limitations:-

----

a) It is slowest driver (snail driver).
b) It is platform dependent driver because it works only on windows platform.

3.2)Type-2 Driver:-

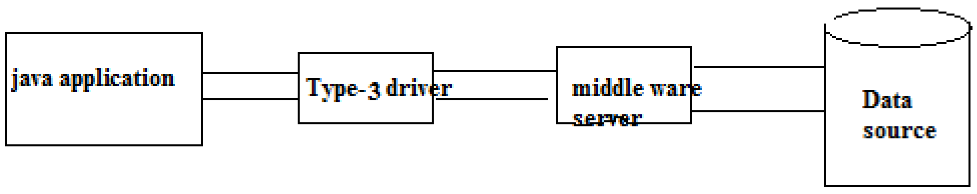


This driver translates JDBC function calls directly into native library calls. These native library calls can be under stood by data source without translation. The native library is usually written in c or c++. The native library is provided by data base vendor. Type2 driver is developed combination of c++ and java.

Drawback:-

1. The native library must be installed on each computer that runs java application.

3.3)Type-3 Driver:-



Type-3 driver converts jdbc calls into middle ware calls. The middleware server converts them into specific data source calls. Type-3 driver is completely developed in java. Type-3 driver follows 3-tier jdbc architecture. Remaining all drivers follows 2-tier jdbc architecture.

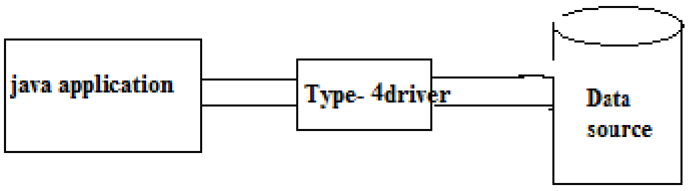
Level-1 : java application and type3-driver both are at client site.
Level-2 : Middleware server is at level2.
Level-3 : data source is level 3.

Type3 driver talks over network to middleware server.

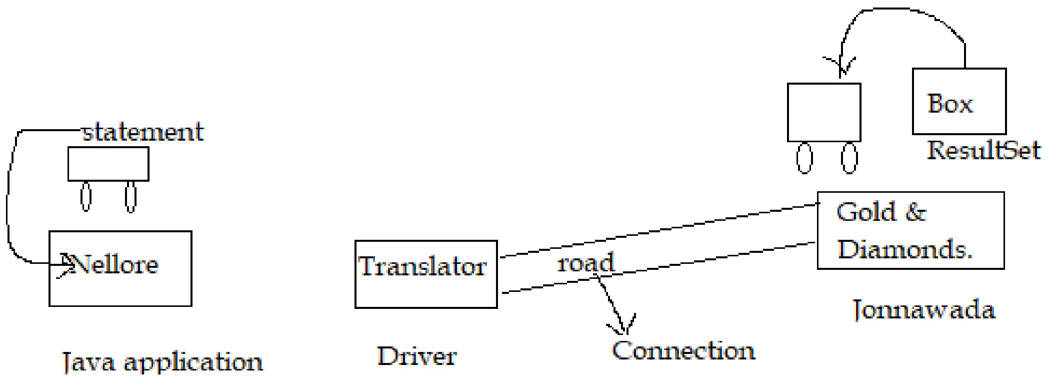
3.4) Type-4 Driver: This driver communicate directly with data source using database native protocol. This protocol is provided by DBMS vendor. Type-4 driver translates jdbc calls to dbms calls. Type-4 driver is completely developed in java. It is also known as thin driver.With out having any extra component , this driver communicate with data source,so it is called thin driver.

Advantages:-

1.performance is high.



4)OverView Of JDBC Process:-



→Translator is driver.
→The road is Connection object.
→Vehicle is statement Object.
→Box is ResultSet object.

Step:1) Load & Register Driver Class:- Usually we use Class.forName("classname") to load any java class. Hence by using the same method we can load Driver class.

Syntax:- System.forName(Driver className)

Whenever we are loading Driver class automatically static block present in that driver class will be executed.

```
 Class  driverclassName
       {
      {      driverclassName ab=new driverclassName();
              DriverManager.registerDriver(driverclassName);
       }
 }
```

Note:- 1.DriverClassName is vary from driver to driver.
2.From JDBC 4.0 onwards,Programmer does ‘t need to write a statement to load and register driver. Because JDBC 4.0 onwards, JVM automatically load the driver from class path.

Step2. Establish connection between java application and database.

Connection con=DriverManager.getConnection(“url of DB”,”username”,”password”);

Step3. Create Statement Object.

```
Statement s1=con.createStatement();
```

Step4. Send and execute sql query.

```
ResultSet r1=s1.executeQuery(“sql query”);
```

Step5. Process Result from ResultSet.

Step6. Close the connection:- After completing Database operations, we should close the opended connection.

Syntax:- ConnectionObject.close();

5)DatabaseConnection:- The connection is Interface. It is available in java.sql package. It establish session between java application and database.

The implementation of Connection Interface is responsibility of DB vendor. The DB vendor selects name for implementation class.
This interface has several abstract methods for creating statements , for getting information about DB and for transaction management.

5.1) Methods:-

Methods for creating statements:-

a) createStatement:-

syntax:-1

public Statement creteStatement()

This method create statement object. It returns that object reference.

Syntax:2

Public Statement **createStatement(int resultSetType,int resultSetConcurrency)**

Creates a Statement object that will generate ResultSet objects with the given type and concurrency.

b) prepareStatement:-

syntax:-

public PreparedStatement prepareStatement("sql query")

This method create prepared statement object. It returns that object reference.

c) callableStatement:-

syntax:-

public callableStatement prepareCall(String invoking statement)

This method creates callableStatement object. This object will be useful to invoke the stored procedure and functions.

Methods for getting info about DB.

a) getMetadata:-

syntax:-
public DatabaseMetaData getMetadata()
This method returns DatabaseMetaData object which contains extra information about connected database.

Methods for Transaction Management.

a)**setAutoCommit(boolean autoCommit)-** Sets this connection's commit mode to true or false.

c)rollback()- Undoes all changes made in the current transaction and releases any database locks currently held by this Connection object.

d)commit()- Makes all changes made since the previous commit/rollback permanent and releases any database locks currently held by this Connection object.

e)rollback(savepoint):- It perform rollback operation for a particular group of operations WRT to savepoint.

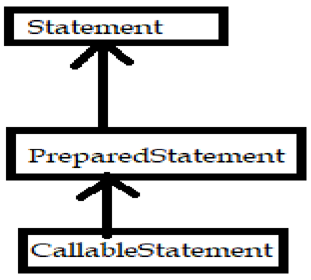
f) public Savepoint setSavePoint():- This method sets the savepoint in transaction.

g) public void releaseSavepoint(Savepoint var):-This method release or delete savepoint .

Note :- Type4 driver does not support releaseSavePoint method.

6) Statement:- There are 3 types of statements.

6.1) Statement/simple Statement.
6.2) Prepared Statement.
6.3)Callable Statement.



Statement is interface.
PreparedStatement is child interface of Statement.
CallableStatement is child interface of PreparedStatement.

6.1) Statement:- It is interface. Its implementation is responsibility of DB vendor. It provides methods to execute queries with database.

Methods:-

a) executeQuery:-
public ResultSet executeQuery("select Query") throws SQLException.
This method is used for executing the select operation. It returns the ResultSet.

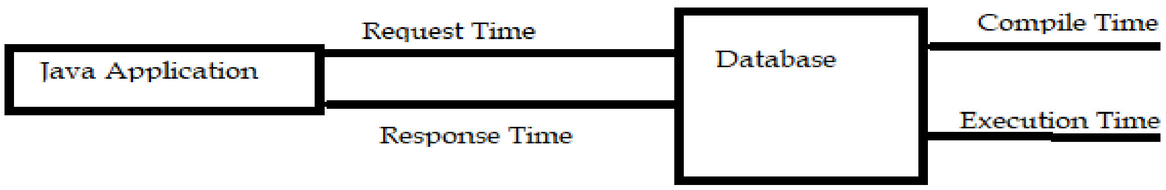
b) executeUpdate:-
Syntax: public int executeUpdate("non-select query") throws SQLException.
This method is used for executing the non-select operations(insertion,deletion,updataion) .It returns integer value. This number represents no.of rows affected in table.

c) Execute:-
Public Boolean execute("SQLquery")throws SQLException.
If we don't know type of query in advance and it is available dynamically then we should use execute() method .
It returns either true or false. True represents executed query is select query. False means executed query is non-select query.
EX:- Boolean b=statementobject.execute("query");

```
If (b==true)
{
   Resulstset r=statementobject.getResultset();
}
Else
 {
   Int a=statementobject.getUpdateCount();
 }
```

Drawback of Statement:- in the case of normal statement,whenever we are executing SQL query , every time compilation and execution will be happened at database side.

EX:- Statement s=con.createStatement();
ResultSet r=s.executeQuery("select * from sample");



Total time perquery= ReqT+CT+ET+ResT
=1ms+1ms+1ms+1ms;
=4ms.

Sometimes,our application requires multiple executions of same query with same or different input values. Every time same query is compiled which creates performance problems.

EX:1 In IRCTC application , it is common requirements to list of all possible trains between 2 places.

Select * from train source= ' ' and dest= ' ';

Query is same but values may be same or different.

Examples:-

```
1.import java.sql.DriverManager;
import java.sql.Connection;
import java.sql.Statement;
import java.sql.ResultSet;
import java.sql.SQLException;
class Type4
{
  public static void main(String arg[])
  {
     try
       {
         Connection
con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:ORCL","scott"
,"sukumar");
         Statement s=con.createStatement();
         ResultSet r1=s.executeQuery("select * from sukumar");
         while(r1.next())
         {
           System.out.println(r1.getInt(1)+" "+r1.getString(2));
         }
       }
     catch(SQLException e){System.out.println(e.getMessage());}
   }
}
```

Output:

```
D:\> java Type4
1 suku
2 veena
3 sulakshmi.
```

EX:2

```
import java.sql.DriverManager;
import java.sql.Connection;
import java.sql.Statement;
import java.sql.ResultSet;
import java.sql.SQLException;
class Type4
{
  public static void main(String arg[])
  {
     try
       {
         Connection
con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:ORCL","scott"
,"sukumar");
         Statement s=con.createStatement();
         Boolean a=s.execute("select * from sukumar");
         if(a==true)
         {
            ResultSet r=s.getResultSet();
            while(r.next())
            System.out.println(r.getInt(1)+" "+r.getString(2));
         }
         else
```

```
          {
            System.out.println(s.getUpdateCount()+" "+ "records updated By sql query");
          }

        }
     catch(SQLException e){System.out.println(e.getMessage());}
   }
}
```

Output:-

```
D:\>java Type4
1 suku
2 veena
3 sulakshmi
```

Ex:3

```
import java.sql.DriverManager;
import java.sql.Connection;
import java.sql.Statement;
import java.sql.ResultSet;
import java.sql.SQLException;
class Type4
{
  public static void main(String arg[])
  {
     try
        {
          Connection
con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:ORCL","scott"
,"sukumar");
          Statement s=con.createStatement();
          int a=s.executeUpdate("delete from sukumar where sno=3");
          System.out.println(s.getUpdateCount()+" "+ "Records Deleted By sql query");


        }
     catch(SQLException e){System.out.println(e.getMessage());}
   }
}
```

Output:-

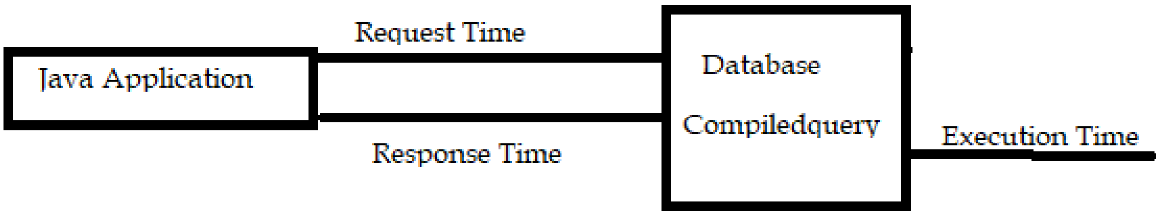
```
D:\>javac Type4.java

D:\>java Type4
```

1 Records Deleted By sql query

6.2) PreparedStatement:- It is child interface of Statement Interface. It over comes drawback of Statement Object. It is used to execute parameterized sql query.

The main advantage of preparedstatement is query will be compiled only once eventhough we are executing multiple times , so that over all application performance will be improved. DB Engine place compiled query in DB. That precompiled query will be returned to the java application in the form of prepared Statement object.

EX:- PreparedStatement p=con.prepareStatement("insert into sample values(?,?,..)");
p.executeUpdate();



Total Time per query= reqT+ResT+E.T
= 1ms+1ms+1ms=3ms

Methods:-
----

1. setInt:-
   Syntax:- public void setInt(int index,int value)
   This method set value in positional parameter using index.
2. setFloat:-
   Syntax:- public void setFloat(int index,float value)
   This method set value in positional parameter using index.
3. setString:-
   Syntax:- public void setString(int index,String value).
   This method set value in position parameter using index.

Q) when is PreparedStatement object to be used?
A) when we want to execute same query with same value or with different value, we should use PreparedStatement.
EX:1 In IRCTC application , it is common requirements to list of all possible trains between 2 places.

Select * from train source= ' ' and dest= ' ';

Query is same but values may be same or different.

→Limitation of PreparedStatement:- we can use prepared statement for only one sql query.

Examples:-

EX:1

```
class Type4
{
  public static void main(String arg[])
  {
     try
        {
          Connection
con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:ORCL","scott"
,"sukumar");
          PreparedStatement s=con.prepareStatement("select * from sukumar");
          ResultSet r=s.executeQuery();
          while(r.next())
          System.out.println(r.getInt(1)+" "+r.getString(2));

        }
     catch(SQLException e){System.out.println(e.getMessage());}
   }
}
```

Output:-

```
D:\>java Type4
1 suku
2 veena
```

Ex:2

```
class Type4
{
  public static void main(String arg[])
  {
     try
        {
          Connection
con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:ORCL","scott"
,"sukumar");
          PreparedStatement s=con.prepareStatement("select * from sukumar where
sno=?");
```

```
        s.setInt(1,1);
        ResultSet r=s.executeQuery();
        while(r.next())
        System.out.println(r.getInt(1)+" "+r.getString(2));
       }
    catch(SQLException e){System.out.println(e.getMessage());}
  }
}
```

Output:-

```
----------
1 suku
```

6.3) CallableStatement:-

6.3.1) Stored Procedure:- In database programming, if any group of sql statements execution is required repeatedly then we define those sql statements as group. We call that group repeatedly based on our requirement.

Hence procedure is group of statements that performs a business logic. This procedure is always stored in database permanently for future purpose and hence name stored procedure. Usually stored procedure is created by DBA not by Java developer.

Syntax:- create or replace procedure namexxx(parameter def1,parameter def2,….)

```
             As
            Begin
                Group of sql statements.
            End;
```

Where procedure can have 3 types of parameters.

1.IN:- It provides value to procedure from outside the procedure.

2.OUT:- Procedure supplies values to outside environment by out type parameter.

3.INOUT:- It used to provide input and to collect output.

The parameter definition syntax:

```
Parametername [IN/Out/INOUT] datatype;
```

Ex:-

```
SQL> create or replace procedure abc(a IN number,b IN number,c OUT number) as
  2  begin
  3  c:=a+b;
  4  end;
  5  /
Procedure created.
```

Note :- If procedure is created with compile time errors, then to know compilation errors,use following command.
SQL>show errors.
It will displays list of errors.

6.3.4) Steps to developjava application using Callable statement.

To call stored procedure and function from java application, then we should use callable statement. It is interface. Driver software vendor is responsible to provide implementation for callable statement.

Step 1: create storedprocedure or function.

Step 2: create Callable statement object. While creating object,we write stored procedure/function invoking statement.

EX:-
CallableStatement st=con.prepareCall("{call storeprocedurename(?,?..)}");--Stored procedure.
CallableStatement st=con.prepareCall("{?=call functionname(?,?,..)}");--For function.

Step3:- set a values only in IN type positional parameter using setter methods of prepared statement Interface.

Step4:- Register Out type positional parameter using following method.

Syntax:-
public void registerOutParameter(int index,int JDBCtype);
First parameter is index of out type parameter.
Second parameter is JDBC data type.

Step5:- execute the stored procedure or Function using only execute() method.

Step6:- Get the values from only OUT type parameters using getter methods of Prepared Statement interface.
EX:- int a= Callablestatementobject.getInt(int index);

Example:

EX1: Invoking Stored procedure from java application.

```
class Type4
{
  public static void main(String arg[])
  {
     try
       {
         Connection
con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:ORCL","scott"
,"sukumar");
```

```
        CallableStatement s=con.prepareCall("{call abc(?,?,?)}");
        s.setInt(1,10);
        s.setInt(2,60);
        s.registerOutParameter(3,Types.INTEGER);
        s.execute();
        int a=s.getInt(3);
        System.out.println("Sum:"+a);

      }
    catch(SQLException e){System.out.println(e.getMessage());}
  }
}
```

Output:

```
Sum:70
```

7)ResultSet:-JDBC ResultSet is an interface of java.sql package. It is a table of data representing a database query result. The ResultSet Object maintains a cursor that pointing to rows of query result. Initially cursor points to before the first row(BFR).

Constancts in ResultSet Interface:-

1. TYPE_FORWARDONLY-→1003
2. TYPE_SCROLL_INSENSITIVE→1004
3. TYPE_SCROLL_SENSITIVE→1005.
   The above 3 constants represents the Resultset type. Default result set type is TYPE_FORWARDONLY.
4. CONCUR_READ_ONLY→1007.
5. CONCUR_UPDATABLE→1008.
   The above 2 constants represents mode/Concurrency of ResultSet. Default mode of ResultSet mode is CONCUR_READ_ONLY.

Methods:-

The methods of the ResultSet interface can be broken down into three categories

1.Navigational methods − Used to move the cursor around.

2.Get methods − Used to view the data in the columns of the current row being pointed by the cursor.

3.Update methods − Used to update the data in the columns of the current row. The updates can then be updated in the underlying database as well.

1.Navigational Methods:-

There are several methods in the ResultSet interface that involve moving the cursor, including −

S.N.	Methods & Description
1	**public boolean first() throws SQLException** Moves the cursor to the first row.
2	**public void last() throws SQLException** Moves the cursor to the last row.
3	**public boolean absolute(int row) throws SQLException** Moves the cursor to the specified row.
4	**public boolean previous() throws SQLException** Moves the cursor to the previous row. This method returns false if the previous row is off the result set.
5	**public boolean next() throws SQLException** Moves the cursor to the next row. This method returns false if there are no more rows in the result set.

2.Get Methods:- This interface has following methods to get data of column of current row.

1. Public int getInt(index/columnname)
2. Public byte getByte(index/columnname)
3. Public short getByte(index/columnname)
4. Public float getFloat(index/columnname)
5. Public double getDouble(index/columnname)
6. Public long getLong(index/columnname)
7. Public String getString(index/columnname)
8. Public Object getObject(index/columnname)

3.Updating Methods:- This interface contains several methods to update a value in column of current row.

1. Public int updateInt(index/columnname,int value)
2. Public byte updateByte(index/columnname,byte value)
3. Public short updateShort(index/columnname,short value)
4. Public float updateFloat(index/columnname,float value)
5. Public double updateDouble(index/columnname,double value)
6. Public long updateLong(index/columnname,long value)
7. Public String updateString(index/columnname,string value)

8. Public Object updateObject(index/columnname, Object value)

Updating a row in the result set changes the columns of the current row in the ResultSet object, but not in the underlying database. To update your changes to the row in the database, you need to invoke one of the following methods.

1. **public void updateRow()**

   Updates the current row by updating the corresponding row in the database.

2.public void deleteRow()

Deletes the current row from the database.

3.**public void refreshRow()**

Refreshes the data in the result set to reflect any recent changes in the database

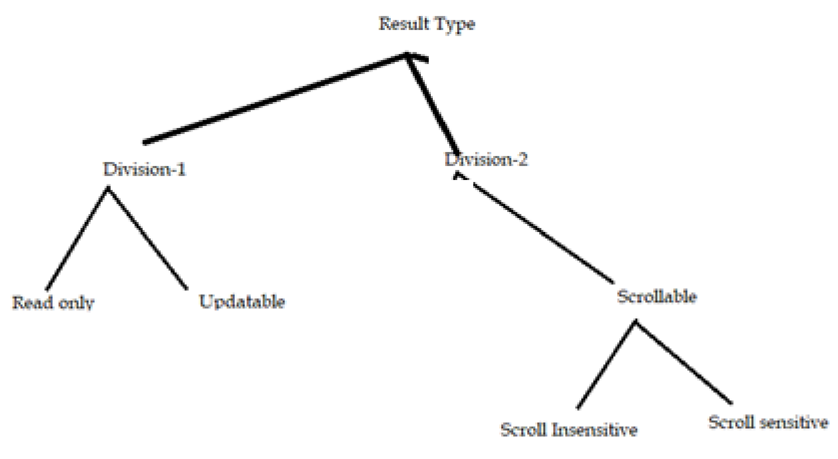
4.**public void insertRow()**

Inserts a row into the database. This method can only be invoked when the cursor is pointing to the insert row.

5.ResultSet MetaData method:- The following method return ResultSet meta Data object.

Public ResultSetMetaData getMetadata();

7.1) ResultSet Types:-



Division1:- Based on operations performed on ResultSet, we can divide ResultSet into 2 types.

1. Read Only ResultSets(Static ResultSets) :- we can perform only read operations on the ResultSet by using corresponding getter methods and we can not perform any updations.
   By default ResultSet is Read Only.
   We can specify explicitly ResultSet as Read only by using the following constant of ResultSet.
   Public static final int CONCUR_READ_ONLY →1007.
2. Updatable ResultSets(Dynamic ResultSets):- The ResultSet which allows programmer to perform updations, such type of ResultSets are called updatable ResultSets. These updations can be made permanent to DB.

   In this case we can perform select,insert,delete and update operations.
   We can specify ResultSet explicityly as updatable by using the following constant of ResultSet.

   Public static final int CONCUR_UPDATABLE→1008.

➔ Updatable resultset allows the programmer to perform insert,update and delete database operations with out using SQL queries.

➔ Very few drivers supports for updatable resultsets.
Ex:- Type1 driver supports .
Type2 supports but we should not use * in SQL query and we should use column names.

Division2:-

1.Scrollable ResultSets:- It allows the programmers to iterate in both forward and backward directions.

We can also jump to particular position randomly or relative to current position. Here wec can move to anywhere. There are two types of Scrollable ResultSets.

1.1) Scroll Insensitive ResultSet

1.2)Scroll Sensitive ResultSet.

1.1) Scroll Insensitive ResultSet:- After getting ResultSet if we are performing any change in DB and if those changes are not reflecting to the ResltSet,such type of ResultSets are called Scroll insensitive ResultSets.

We can specify explicitly ResultSet as Scroll insensitive by using the following constant

Public static final int TYPE_SCROLL_INSENSITIVE→1004

1.2) Scroll sensitive ResultSets:- After getting the ResultSet if we perform any change in the data base and if those changes are visible to ResltSEt ,such type of ResultSt is called Scroll Sensitive ResultSet.

We can specify explicitly ResultSet as scroll sensitive by using the following constant.

Public static final int TYPE_SCROLL_SENSITIVE→10005.0

9.2) How To get Required ResultSet:- we should create statement objects as follows to get required ResultSet.

Statement s=con.createStatement(int type,int mode)

PreparedStatement p=con.prepareStatement(query,int type,int mode).

Example:-

----

1. Create Readonly and Insensitive scrollable ResultSet.

```
import java.sql.DriverManager;
import java.sql.Connection;
import java.sql.Statement;
import java.sql.ResultSet;
import java.sql.SQLException;
class Type4
{
  public static void main(String arg[])
  {
     try
       {
         Connection
con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:ORCL"
,"scott","sukumar");
         Statement
s=con.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,ResultSet.
CONCUR_READ_ONLY);
         ResultSet r=s.executeQuery("select * from sukumar");
         r.absolute(3);
         System.out.println("Third Record From Begining:"+r.getInt(1)+"
"+r.getString(2));
         r.previous();
         System.out.println("Befoe Third Record from begining is
:"+r.getInt(1)+" "+r.getString(2));
         r.relative(2);
          System.out.println("Second Record from last:"+r.getInt(1)+"
"+r.getString(2));

       }
     catch(SQLException e){System.out.println(e.getMessage());}
   }
}
```

Output:

```
D:\>java Type4
Third Record From Begining:3 sulakshmi
Before Third Record from begining is :2 veena
Second Record from last:1 suku
```

8) MetaData:- MetaData means data about data or extra information of original data.

JDBC provided following types of MetaData.
8.1)DatabaseMetaData.
8.2)ResultSetMetaData.

8.1) DatabaseMetaData:- It is interface present in java.sql package. Driver software vendor is responsible to provide implementation.
We use Database metadata to get information about connected Database. The information is product name,product version, max no.of tables the database supports, max no.of columns each table can allows …etc.
We use Database metadata to check whether a particular feature is supported by DB or not like stored procedures,full joins etc.
We get Database metadata object using getMetaData() method of Connection Interface.

```
DatabaseMetaData d= con.getMetaData();
```

Example:-

```
import java.sql.Connection;
import java.sql.Statement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.DatabaseMetaData;
class Type4
{
  public static void main(String arg[])
  {
     try
       {
         Connection
con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:ORCL","scott"
,"sukumar");
          DatabaseMetaData d=con.getMetaData();
          System.out.println(" DB product Name:"+ d.getDatabaseProductName());
        System.out.println(" Driver Name:"+d.getDriverName());
        System.out.println("DB URL:"+ d.getURL());
        System.out.println("DB User:"+ d.getUserName());
        System.out.println("Maximum columns Table:"+d.getMaxColumnsInTable());
```

```
          System.out.println("Key words:"+d.getSQLKeywords());
         System.out.println("Numeric Functions:"+d.getNumericFunctions());
         System.out.println("String Functions:"+d.getStringFunctions());

       }
    catch(Exception e){System.out.println(e.getMessage());}
   }
}
```

Output:

```
D:\>java Type4
 DB product Name:Oracle
Driver Name:Oracle JDBC driver
DB URL:jdbc:oracle:thin:@localhost:1521:ORCL
DB User:SCOTT
Maximum columns Table:1000
Max Tables in select statement:0
Key words:ACCESS, ADD, ALTER, AUDIT, CLUSTER, COLUMN, COMMENT,
COMPRESS, CONNECT, DATE, DROP, EXCLUSIVE, FILE, IDENTIFIED,
IMMEDIATE, INCREMENT, INDEX, INITIAL, INTERSECT, LEVEL, LOCK,
LONG, MAXEXTENTS, MINUS, MODE, NOAUDIT, NOCOMPRESS, NOWAIT,
NUMBER, OFFLINE, ONLINE, PCTFREE, PRIOR, all_PL_SQL_reserved_ words
Numeric
Functions:ABS,ACOS,ASIN,ATAN,ATAN2,CEILING,COS,EXP,FLOOR,LOG,LO
G10,MOD,PI,POWER,ROUND,SIGN,SIN,SQRT,TAN,TRUNCATE
String
Functions:ASCII,CHAR,CHAR_LENGTH,CHARACTER_LENGTH,CONCAT,LC
ASE,LENGTH,LTRIM,OCTET_LENGTH,REPLACE,RTRIM,SOUNDEX,SUBSTR
ING,UCASE
```

8.2) ResultSetMetaData:- It is an interface present in java.sql package.Driver software vendor is responsible to provide implementation. It provides information about data base table represented by ResultSet object.The information is no.of column,column name,column type …etc.

We can get ResultSetMetaData object by using getMetaData() method of ResultSet interface.

Ex:-

```
import java.sql.DriverManager;
import java.sql.Connection;
import java.sql.Statement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.ResultSetMetaData;
class Type4
```

```
{
  public static void main(String arg[])
  {
     try
       {
         Connection
con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:ORCL","scott"
,"sukumar");
         Statement s=con.createStatement();
         ResultSet r1=s.executeQuery("select * from sukumar");
         ResultSetMetaData r=r1.getMetaData();
         int a=r.getColumnCount();
         System.out.println("Total no.ofcolumns in ResultSet:"+a);
         for(int i=1;i<=a;i++)
         {
         System.out.println("Column number:"+ i);
         System.out.println("Column Name:"+r.getColumnName(i));
         System.out.println("Column type:"+ r.getColumnType(i));
         System.out.println("Column Size:"+r.getColumnDisplaySize(i));
           System.out.println(" ----------------------------------------");
         }

       }
     catch(Exception e){System.out.println(e.getMessage());}
   }
}
```

Output:

```
D:\>java Type4
Total no.ofcolumns in ResultSet:2
Column number:1
Column Name:SNO
Column type:2
Column Size:22
 ----------------------------------------
Column number:2
Column Name:SNAME
Column type:12
Column Size:10

 ----------------------------------------
```

9.Transaction Processing:-

9.1)Definition:- Programmer combines related operations as single unit of work. This single unit of work is called transaction. Every transaction must work on rule “either all or none”.

Ex1:- Transaction Name: Fund Transfer.
Operation1: Debit fund from sender account.
Operation2:credit fund into receiver account.

Ex2:- Transactiion Name: Movie Ticket Reservation.
Operation1:Verify the Status.
Operation2:Reserve the tickets.
Operation3:Payment.
Operation4:issue Ticket.

All operations in Transaction should be performed as single unit only. If some operations success and some operations fails then there may be data inconsistency problem.Therefore every transaction must work on rule “either all or none” to avoid data inconsistency problem.

9.2) Transaction Types:- There are two types of Transactions.

9.2.1) Local Transaction
9.2.2)Global Transaction.

9.2.1) Local Transaction:- if all operations in transaction are executed over same database, then that transaction is called Local Transaction.
Ex:- Fund transfer from one account to another account where both accounts in same bank.
Note:- JDBC supports Local Transaction.

9.2.2) Global Transaction:- If transaction is executed over different databases , then
That transaction is called Global Transaction.
Ex:-Fund transfer from one account to another account where both accounts in different banks.
Note:- JDBC does not supports global Transaction.
EJB,spring Frame work supports global Transaction.

9.3)Transaction Properties:- Every transaction should follow the following four ACID properties.
1)Atomicity 2)consistency 3)Isolation 4)Durability.
See the DBMS material.

9.4)Process of Transaction management in jdbc:-

JDBC manages transaction with following four methods.
1.setAutoCommit(false/true).
2.commit();
3.rollback();

All above methods belongs to Connection Interface.

Example:

```
import java.sql.ResultSet;
import oracle.jdbc.pool.OracleConnectionPoolDataSource;
import java.util.Scanner;
class Type4
{
  public static void main(String arg[])
  {
    try
      {
        OracleConnectionPoolDataSource                    ds=new
OracleConnectionPoolDataSource();
        ds.setURL("jdbc:oracle:thin:@localhost:1521:ORCL");
        ds.setUser("scott");
        ds.setPassword("sukumar");
        Connection con=ds.getConnection();
        Statement p=con. createStatement();
        ResultSet r=p.executeQuery("select * from sukumar");
        System.out.println("------Data Before Transaction-----");
        while(r.next())
        System.out.println(r.getInt(1)+" "+r.getInt(2));
        System.out.println("---------------Transaction Begins--------");
        con.setAutoCommit(false);
        p.executeUpdate("update  sukumar  set  balance=balance-1000  where
ano=1");
        p.executeUpdate("update  sukumar  set  balance=balance+1000  where
ano=2");
        System.out.println("Do you commit Transaction:");
        Scanner sv=new Scanner(System.in);
        String ch=sv.next();
        if(ch.equalsIgnoreCase("yes"))
        con.commit();
        else
        con.rollback();
        r=p.executeQuery("select * from sukumar");
        System.out.println("------Data After Transaction-----");
        while(r.next())
        System.out.println(r.getInt(1)+" "+r.getInt(2));
      }
    catch(Exception e){System.out.println(e.getMessage());}
   }
}
Output:
--------
D:\>java  Type4
------Data Before Transaction-----
1 7000
2 5000
---------------Transaction Begins--------
Do you commit Transaction:
yes
------Data After Transaction-----
1 6000
2 6000
```

9.5)SavePoint:- It is an interface in java.sql package. Driver software vendor is responsible to provide implementation.Savepoint concept is applicable only in transactions.

The following methods are used for managing savepoint.

1.setSavepoint()
2.releaseSavepoint()
3.rollback(Savepoint   var);

All above methods belongs to Connection Interface.

JDBC and EMBEDDED SQL

1.Tables:- You create table by formatting SQL query. The query contains the CREATE TABLE SQL statement that contains name of the table, which is called customer address because table will contain address information on customers.

```
  Try
{
  String Query=”CREATE TABLE CustomerAddress(“+”
“CustomerNumber Char(30),”+
“CustomerStreet char(30)”,+
 “CustomerCity Char(30))”;
  DataRequest=Database.createStatement();
 DataRequest=execute(query);
 DataRequest.close();
}
```

1.1)Setting Default Value for Column:- The DBMS can enter default value into column automatically if column is left empty whenever a new row is inserted into table. You determine the value entered by the DBMS by creating a default value when you create table. Any value can be used as default value as long as the value conforms to the data type and size of column.

```
  Try
{
        String Query=”CREATE TABLE CustomerAddress(“+”
        CustomerNumber Char(30)NOT NULL,”+
          “CustomerStreet char(30)NOT NULL”,+
          “CustomerCity Char(30)DEFAULT ‘NLR’)”;
         DataRequest=Database.createStatement();
        DataRequest.execute(query);
        DataRequest.close();
}
```

1.2)Dropping a Table:- A developer may have the right to remove table, but this is usually reserved for the development environment only.

Using the Drop table statement in the query Drops the Table.

```
   Try
{
   String Query=new String("DROP TABLE CustomerAddress");
        DataRequest=Database.createStatement();
       DataRequest.execute(query);
       DataRequest.close();
}
```

2.Inserting The Data Into Tables:- Once Database,tables, and indexes are create, a J2EE components can insert a new row into table.

2.1)Inserting a Row:- The insert into statement is used to insert a new row into a table. The INSERT INTO statement contains the name of the table into which row is to be inserted and the name of the columns in which the values are inserted. The VALUES clause is used to define the values that are to be placed into the row.

Each value is passed in the VALUES clause in the same order as the corresponding column's name. The first value in the VALUES clause is customer number and is followed by the customer's firstname,lastname, and date of the first order placed by the customer.

```
  Try{
 String query="INSERT INTO Customers"+"(customernumber,firstname,lastname)"+
       "VALUES(1,'sv','smith')";
      DataRequest=Database.createStatement();
      DataRequest.executeUpdate(query);
      DataRequest.close();
      }
```

3.Selecting Data From table:- Retrieving information from database is most frequently used routine of J2EE components that interact with a database.

3.1)Selecting All Data From table:- The Select statement is used to retrieve data from table. In this example, all the columns of all rows of the customers table are returned in the resultset by DBMS after query is processed.

```
    Try
{
String query=new String("SELECT"+"firstname,LastName,street"+"FROM Customers");
DataRequest=Database.createStatement();
ResultSet result=DataRequest.executeQuery(query);
displayResult(result);
 DataRequest.close();
}
```

3.2)Requesting One Column:- You can specify a column that you want returned from the table by using column name in the SELECT statement.

```
   Try{

         String query=new String("select lastname FROM customers");
          DataRequest=Database.createStatement();
ResultSet result=DataRequest.executeQuery(query);
displayResult(result);
 DataRequest.close();

}
```

3.3)AND ,OR and NOT Clauses:- The WHERE clause in SELECT statement can evaluate values in more than one column of a row by using the AND,OR and NOT clauses to combine expressions in the WHERE clause.

3.3.1) AND clause:- the purpose of the AND clause is to join two subexpressions together to form one compound expression. The AND clause tells the DBMS that the Boolean value of both subexpressions must be true for the compound expression to be true.

```
Try{
String query=new String("SELECT firstname,lastname"+"FROM customers"+
                                        "WHERE lastname='jones'+ "AND firstname='bob'");
 DataRequest=Database.createStatement();
ResultSet result=DataRequest.executeQuery(query);
displayResult(result);
 DataRequest.close();
}
```

3.3.2) OR clause:- The purpose of the OR clause is to join two subexpressions together to form one compound expression. The OR clause tells the DBMS that the Boolean value of either subexpressions must be true for the compound expression to be true.

```
Try{
String query=new String("SELECT firstname,lastname"+"FROM customers"+
                                        "WHERE lastname='jones'+ "OR firstname='bob'");
 DataRequest=Database.createStatement();
ResultSet result=DataRequest.executeQuery(query);
displayResult(result);
 DataRequest.close();
}
```

3.4)LESS Than and GreaterThan Operators:- The less than and greater than operators direct the DBMS to assess whether or not the value in the specified column of the current row is less than or greater than the value in the WHERE clause expression.

3.4.1) LESS THAN OPERATOR:- in the following example, all the columns of rows from customers table are returned as long as the value of the sales column is less than 50000.

```
    Try{
        String query=new String("SELECT"+ "firstname,lastname,sales"+"FROM Customers"+"WHERE sales<5000");
        DataRequest=Database.createStatement();
        ResultSet result=DataRequest.executeQuery(query);
        displayResult(result);
        DataRequest.close();
      }
```

3.4.2)Greather THAN operator:- :- in the following example, all the columns of rows from customers table are returned as long as the value of the sales column is greater than 50000.

```
          Try{
           String query=new String("SELECT"+ "firstname,lastname,sales"+"FROM
Customers"+"WHERE
sales>5000");
         DataRequest=Database.createStatement();
        ResultSet result=DataRequest.executeQuery(query);
        displayResult(result);
        DataRequest.close();
       }
```

4.DELETING DATA FROM TABLE:- A row is deleted from a table by using DELETE FROM statement. The DELETE FROM statement includes the name of the table and WHERE clause that contains an expression that identifies the row or rows to remove from table.

```
      Try{
          String query=new String("DELETE FROM customers"+ "Where
lastname='jones'");
          DataRequest=Database.createStatement();
         Int result=DataRequest.executeUpdate(query);
        displayResult(result);
        DataRequest.close();
       }
```

5.Updating Tables:-Modifying data in database is one of the most common functionalities includes in every J2EE component that provides database interations.

5.1)Updating row and column:- The UPDATE statement is used to change value of one or more columns in one or multiple rows of table.

```
  Try{
  String query=new String(“UPDATE customers”+”SET street=’5 Main Street’ “+
                    “WHERE firstname=’blob’”);
            DataRequest=Database.createStatement();
       Int result=DataRequest.executeUpdate(query);
      displayResult(result);
      DataRequest.close();
     }
```

5.2)Updating Every Row:- All rows in a table can be updated by excluding the WHERE clause in UPDATE statement.

```
    Try{
           String query=new String(“UPDATE Customers”+ “SET discount=0”);
                       DataRequest=Database.createStatement();
       Int result=DataRequest.executeUpdate(query);
      displayResult(result);
      DataRequest.close();
     }
```

5.3)Updating Multiple Columns:-Multiple columns of rows can be updated simultaneously by specifying the column names and appropriate values in the SET clause of the query.

```
 Try{
     String Query=new String(“UPDATE customers”+ “SET discount=12 ,
street=’jonesstreet’”+
                          “WHERE lastname=’jones’”);
                 DataRequest=Database.createStatement();
       Int result=DataRequest.executeUpdate(query);
      displayResult(result);
      DataRequest.close();
     }
```

6)JOINING TABLE:- Tables are joined in query using 2 step process.

First, both tables that are being joined must be identified in the FROM clause where tables are listed one after the other and are separated by a comma.

Next, an expression is created in WHERE clause that identifies the columns that are used to create join. Let's say that an orders table is joined to customers table using the customer number. The following line contains a WHERE clause expression that joins these tables:

```
WHERE customernumber=custNum;
```

The joined tables create logical table that has all the columns of both tables. All the tasks performed on single table previously in this chapter can also be applied to join tables.

6.1) Joining Two Tables:- we join the customers table and zipcode table. These tables are identified in FROM clause.The WHERE clause expression identifies the zip column from customers table and ZIP code column from ZIPCode table.

```
 Try{
String query=new String("SELECT firstname,lastname,city,state,zipcode"+"FROM
Customers,zipcode"+"WHERE zip=zipcode");
DataRequest=Database.createStatement();
Results=DataRequest.executeQuery(query);
displayResults(Results);
        DataRequest.close();
    }
```

6.2)Multitable Join:- More than two tables can be joined together by using the name of each table in the join in the FROM clause and by defining the join with appropriate column names in the WHERE clause expression. There are 3 tables joined in this example. These are the customers table, and products table and products table.

```
  Try
    {
 String query=new String(
"SELECT firstname,lastname,ordernumber,productname,quantity"+
 "FROM customers,orders,Products"+
 "WHERE prodnumber=productnumber"+"AND custnumber=customerNumber");
  DataRequest=Database.createStatement();
  Results=DataRequest.executeQuery(query);
  displayResults(Results);
   DataRequest.close();
      }
```

7) Calculating Data:- The DBMS can calculate values in a table and return result of calculation inResultSet by using one of five built-in calculation functions. There are five kinds of built-in functions.

a)sum():Tallies values in column that is passed to built-in function.
b)AVG():- Averages values in column that is passed to built-in functions.
c)MIN():- Determines the minimum value in column that is passed to built-in function.
d)MAX():- Determines the maximum value in column that is passed to built-in function.
e)COUNT():- Determines the number of rows in column that is passed to built-in function.

Example to sum():-

```
Try
{
  String query=new String("SELECT SUM("quantity")"+ "From orders");
              DataRequest=Database.createStatement();
              Results=DataRequest.executeQuery(query);
              displayResults(Results);
              DataRequest.close();
}
```

8)Grouping & Ordering Data:- Columns are retuned in resultset in the order the column names appear in the SELECT statement of the query. The order in which rows appear in the resultSet can be grouped into similar values or sorted in ascending or descending order by using GROUP BY clause or the ORDER BY clause.

8.1)Group By:- The Group By clause specifies the name of column whose values are used to group rows in the ResultSet.

```
   Try
{
  String query=new String("SELECT storenumber,SUM(sales)"+"FROM sales"+
                        "Group By storenumber");
              DataRequest=Database.createStatement();
             Results=DataRequest.executeQuery(query);
             displayResults(Results);
            DataRequest.close();
}
```

8.2)Conditional Grouping:- The number of rows that are included in a group can be limited by including a conditional expression in query. A conditional expression is similar to WHERE clause expression.

Here are the requirements for using conditional expression in the HAVING clause:

The expression must result in single value.

The result must appear in every column named in the expression.

The expression can include a built-in calculation function.

```
  Try
{
   String Query=new String(" SELECT storenumber,sum(sales)"+"From sales"+"Group By
                Storenumber"+ "Having sum(sales)>400");
                 DataRequest=Database.createStatement();
                 Results=DataRequest.executeQuery(query);
                displayResults(Results);
```

```
            DataRequest.close();
          }
```

9. MetaData:- Go to MetaData concept in second chapter of this unit.

10.View:- The view is virtual table. Each view is uniquely identified with a name. Once view is created, the J2EE component references a VIEW the same way that a table is referenced in a query.

Rules for using Views.

- More than one table can be used in a VIEW.
- More than one column can be used in a VIEW.
- Create view for each classification of user,rather than for each user.
- Restrict access to a table on a need to know basis.
- Create many views as necessary to simplify access to database.

10.1)Creating View:- View is created by using CREATE VIEW statement.

```
Try
   {
  String query=new String(
   "create view store AS"+"SELECT  orderno,productno "+
   "FROM orders "+
   " WHERE storenum =17");
   DataRequest=Database.createStatement();
   DataRequest.execute(query);
   DataRequest.close();
   }
```

10.2)Dropping view:- A view can be removed by using the DROP VIEW statement. The drop view statement requires the name of the view that is to be dropped. There are two modifiers that are used with DROP VIEW statement.

1. CASCADE: Removes all views that depends on view specified in the DROP VIEW statement and removes specified VIEW.

2. RESTRICT: Removes only view specified in the Drop VIEW statement.

```
  Try
{
         String query=new String(" DROP view store CASCADE");
          DataRequest=Database.createStatement();
          DataRequest.execute(query);
          DataRequest.close();
           }
```

10.3)Creating Horizantal VIEW:- There are two kinds of Views. These are a vertical VIEW and horizontal VIEW.

A vertical view includes all rows of the underlying table and includes some , but not all, columns of the table.

A horizontal view contains all column in the underlying table, but only some rows or all rows of the table.

```
        Try
           {
         String query=new String("create view store AS"+"SELECT  * "+
        "FROM orders "+
          " WHERE storenum =17");
          DataRequest=Database.createStatement();
          DataRequest.execute(query);
          DataRequest.close();
           }
```

10.4) Inserting Row into view:- A new row can be inserted into the underlying tables tht comprise a VIEW by using INSERT INTO statement and referencing the name of the VIEW.

```
  Try
   {
     String    query=new    String("Insert    into
store"+"(productname,orderno)"+"values('x',7)"));
          DataRequest=Database.createStatement();
```

```
                Result=DataRequest.executeUpdate(query);
               S.o.p(result);
                DataRequest.close();
                 }
```

11.Indexing:-

11.1) Creating Indexing:- An index is creaed by using CREATE INDEX statement in a query. The create index statement contains name of the index and any modifier that describes to DBMS the type of index that is to be creted.

```
        Try{
             String query=”CREATE UNIQUE Index custname”+
                                   “ON customeraddress(customernumber)”;


     DataRequest=Database.createStatement();
      DataRequest.execute(query);
       DataRequest.close();
    }
```

11.2)Creating the Cluster Index:- cluster index is an index whose key is ceaed from two or more columns of table.

```
       Try{
             String query=”CREATE UNIQUE Index custname”+
             “ON customeraddress(firstname,lastname)”;
     DataRequest=Database.createStatement();
     DataRequest.execute(query);
     DataRequest.close();
    }
```

11.3) Drop the Index:- An existing index can be dropped from the database by using DROP INDEX statetment.

```
     Try{
      String query=new String(“DROP INDEX custname on customeraddress”);
     DataRequest=Database.createStatement();
```

```
  DataRequest.execute(query);
  DataRequest.close();
}
```