UNIT - 5
Data Base Connectivity using JDBC driver interface

5.1 know about JDBC

JDBC Stands for Java Data Base Connectivity. It is Standard API for all Java programs to connect to databases. The JDBC API is available in 2 packages.

- Core API **java.sql.**
- Standard extension to JDBC API **javax.sql**(supports connection pooling, transactions, etc.)

There are few steps to connect to a database and retrieve/insert/update the database:

- Load the driver
- Establish the connection
- Create the statements
- Execute the query
- Obtain results
- Close connection

5.2 Understand JDBC Architecture:

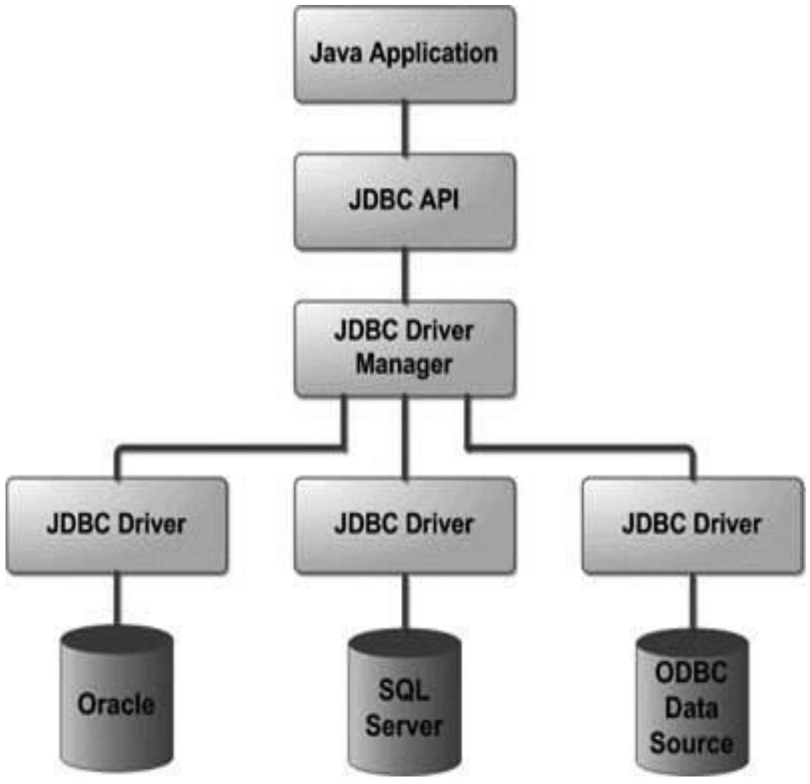
The JDBC API supports both two-tier and three-tier processing models for database access but in general, JDBC Architecture consists of two layers −

- **Application layer (JDBC API):** This provides the application-to-JDBC Manager connection.
- **Database layer (JDBC Driver API):** This supports the JDBC Manager-to-Driver Connection.

The JDBC API uses a driver manager and database-specific drivers to provide transparent connectivity to heterogeneous databases.

The JDBC driver manager ensures that the correct driver is used to access each data source. The driver manager is capable of supporting multiple concurrent drivers connected to multiple heterogeneous databases.

Following is the architectural diagram, which shows the location of the driver manager with respect to the JDBC drivers and the Java application −



Common JDBC Components

The JDBC API provides the following interfaces and classes −

- **DriverManager(class):** This class can be used to register or load drivers into your application and get the connection from data base.
- **Driver(interface):** This interface handles the communications with the database server. Every database vender must need to implement Driver interface for accessing database.
- **Connection(interface):** this interface useful to get connection from database.
- **Statement(interface):** this interface useful to pass SQL statements to database.
- **ResultSet(interface):** These objects hold data retrieved from a database after you execute an SQL query using Statement objects. It acts as an iterator to allow you to move through its data.
- **SQLException(class):** This class handles any errors that occur in a database application.

5.3 Know about Connection interface and DriverManager Class :

DriverManager Class: This class can be used to register or load drivers into your application and get the connection from data base.The DriverManager class maintains a list of Driver classes that have registered themselves by calling the method **DriverManager.registerDriver().**

Useful methods of DriverManager class

Method	Description
1) public static void registerDriver(Driver driver);	is used to register the given driver with DriverManager.
2) public static void deregisterDriver(Driver driver);	is used to deregister the given driver (drop the driver from the list) with DriverManager.
3) public static Connection getConnection(String url);	is used to establish the connection with the specified url.
4) public static Connection getConnection(String url,String userName,String password);	is used to establish the connection with the specified url, username and password.

Connection Interface:

A Connection is the session between java application and database. The Connection interface is a factory of Statement, PreparedStatement, and DatabaseMetaData i.e. object of Connection can be used to get the object of Statement and DatabaseMetaData. The Connection interface provide many methods for transaction management like commit(), rollback() etc.

Methods of ConnectionInterface:

1) public Statement createStatement(): creates a statement object that can be used to execute SQL queries.

2) public void commit(): saves the changes made since the previous commit/rollback permanent.

3) public void rollback(): Drops all changes made since the previous commit/rollback.

4) public void close(): closes the connection and Releases a JDBC resources immediately.

5.4 List and know about JDBC drivers:

What do you mean by driver?

JDBC Driver is a software component that enables java application to interact with the database.
The driver is loaded with the help of a static method.

```
Class.forName(drivername)
```

Every database has its own driver. The following table describes names for few databases:

DataBase name	Driver Name
MS ACCESS	sun.jdbc.odbc.JdbcOdbcDriver
Oracle	oracle.jdbc.driver.OracleDriver
Microsoft SQL server 2000	com.Microsoft.sqlserver.jdbc.SqlServerDriver
MySQL	org.gjt.mm.mysql.Driver

There are 4 types of JDBC drivers:

1. JDBC-ODBC bridge driver
2. Native-API driver (partially java driver)
3. Network Protocol driver (fully java driver)
4. Thin driver (fully java driver)

1. JDBC-ODBC bridge driver: The JDBC-ODBC bridge driver uses ODBC driver to connect to the database. The JDBC-ODBC bridge driver converts JDBC method calls into the ODBC function calls. This is now discouraged because of thin driver.

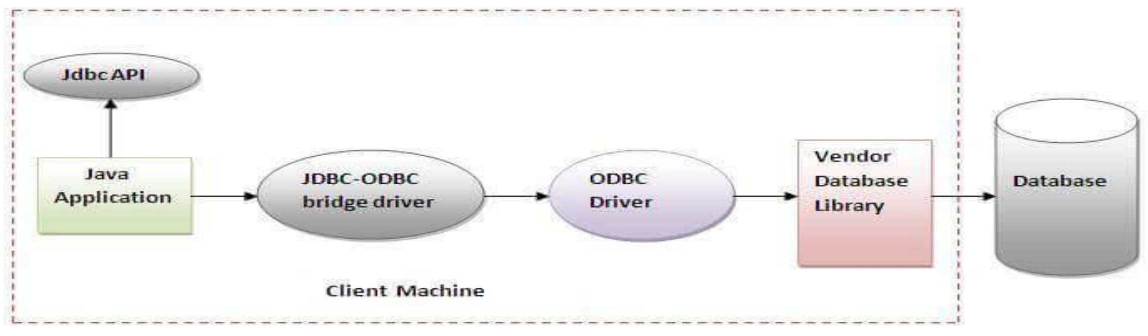


Figure- JDBC-ODBC Bridge Driver

Oracle does not support the JDBC-ODBC Bridge from Java 8. Oracle recommends that you use JDBC drivers provided by the vendor of your database instead of the JDBC-ODBC Bridge.

Advantages:

- easy to use.
- can be easily connected to any database.

Disadvantages:

- Performance degraded because JDBC method call is converted into the ODBC function calls.
- The ODBC driver needs to be installed on the client machine.

2. Native-API driver (partially java driver): The Native API driver uses the client-side libraries of the database. The driver converts JDBC method calls into native calls of the database API. It is not written entirely in java.

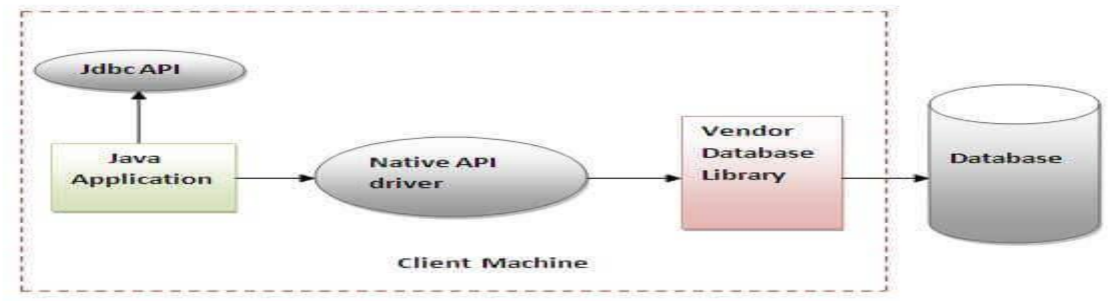


Figure- Native API Driver

Advantage:

- Performance upgraded than JDBC-ODBC bridge driver.

Disadvantage:

- The Native driver needs to be installed on the each client machine.
- The Vendor client library needs to be installed on client machine.

3. Network Protocol driver (fully java driver): The Network Protocol driver uses middleware (application server) that converts JDBC calls directly or indirectly into the vendor-specific database protocol. It is fully written in java.

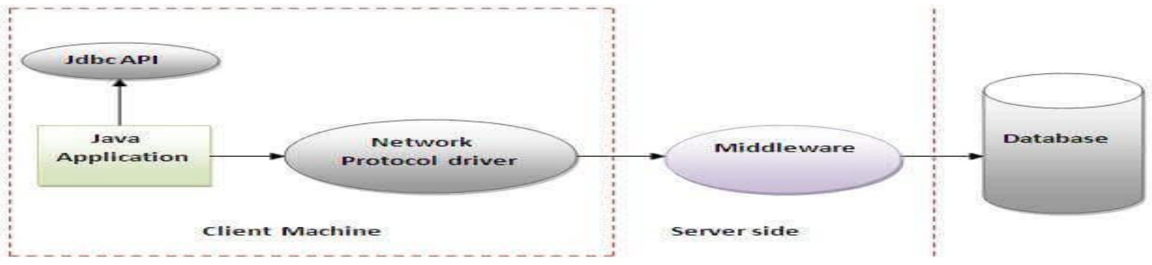


Figure- Network Protocol Driver

Advantage:

- No client side library is required because of application server that can perform many tasks like auditing, load balancing, logging etc.

Disadvantages:

- Network support is required on client machine.
- Requires database-specific coding to be done in the middle tier.
- Maintenance of Network Protocol driver becomes costly because it requires database-specific coding to be done in the middle tier.

4. Thin driver (fully java driver): The thin driver converts JDBC calls directly into the vendor-specific database protocol. That is why it is known as thin driver. It is fully written in Java language.

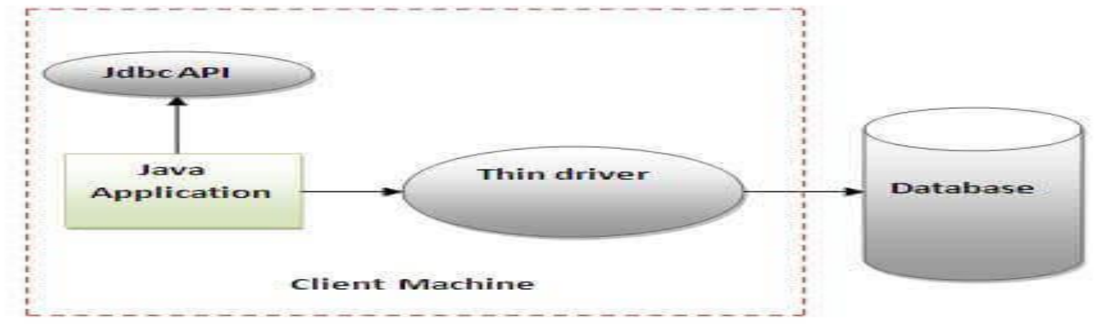


Figure- Thin Driver

Advantage:

- Better performance than all other drivers.
- No software is required at client side or server side.

Disadvantage:

- Drivers depend on the Database.

5.5 Know about 3 kinds of Statements: Statement , Prepared Statement and Callable Statement:

The connection (after being established) is used to send SQL statements to the database. There`re 3 types interfaces in java.sql package used for sending SQL statements to databases, namely **Statement** and its 2 sub interfaces **PreparedStatement** and **CallableStatement.** There methods of the connection objects are used to return objects of these statements.

- A **Statemen**t object is used to send a simple **SQL statement to the database with no parameters.** Its object returned by using the Connection object is used to return objects of these statements.
  Syntax: **Statement st = con.createStatement();**
- A **Prepared** Statement object sends **precompiled statements with or without IN parameters**. Normally, we insert rows of data into the databases using the insert SQL statement. For example, we write an SQL statement which is sent to the database. If n rows need to be inserted, then the same statements get compiled n times. In Prepared Statement, only the values that have to be inserted are sent to the database again and again.
  Syntax: **PreparedStatement ps= con.preparedStatement(String query);**
- A **CallableStatement** object is used to call stored procedures.
  Syntax: **CallableSatement cs= com.prepareCall(String query);**

5.6 UnderStand the steps in connecting to database using JDBC:

Steps to connect to a database:

1. Load the driver
2. Establish the connection.
3. Create the statements.
4. Execute query and obtain the results.
5. Iterate through the results.
6. Close the connection

5.6.1 Know how to establish connection:

- A connection to the database is established using the static method **getConnection(databaseURL)** of the **DriverManager** Class.
- The DriverManager class is a class for managing JDBC Drivers.
- The database URL takes the following shape
  jdbc:subprotocol:subname.
- If any problem occurs during accessing database, an SQLException is generated, else a connection reference is returned which refers to a connection to the database Connection is an interface in java.sql package.

Syntax: **Connection con= DriverManager.getconnection(databaseURL);**

- The following table shows the various database URLs for connecting to various databases.

DataBase	databaseURL
MS ACCESS	jdbc:odbc:<DSN>
Oracle thin driver	jdbc:oracle:thin:@<HOST>:<PORT>:<SID>
MICROSOFT SQL SERVER	jdbc:Microsoft:sqlservver://<HOST>:<PORT>[;DATABASE NAME=<DB>]

A sample example program on establishing a connection:

```
/*Before execution of programs to install oracle software into your system and set the following paths in
command prompt set path= C:\Program Files (x86)\Java\jdk1.7.0_01\bin
                 set classpath=C:\oraclexe\app\oracle\product\10.2.0\server\jdbc\lib\ojdbc14.jar;.;   */
import java.sql.*;
public class OracleCon
{
public static void main(String args[])throws SQLException
  {
try
 {
   //step1 load the driver class
   Class.forName("oracle.jdbc.driver.OracleDriver");
   //step2 create  the connection object
  Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","manager");
  System.out.println("connection established");
 }
 catch(Exception e)
  {
        System.out.println(e);
  }
 }
}
```

output:

```
connection established
```

5.6.2 Create a statement: refer 5.5 topic

Example Program:

```
import java.sql.*;
class OracleConnection {
  public static void main(String args[]){
        try{
            //step1 load the driver class
                  Class.forName("oracle.jdbc.driver.OracleDriver");
```

```
                    //step2 create  the connection object
 Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","manager");
                     //step3 create the statement object
                    PreparedStatement ps=con.prepareStatement("create table student(sno number(10),sname
                    varchar2(30),smarks number(10))");
                    Statement st=con.createStatement();
                    String insert="insert into student values(10,'madhu',86)";
                    String update="update student set sname='subbu' where sno=10";
                     //step4 execute queries
                    ps.executeUpdate();
                    System.out.println("inserted rows are="+st.executeUpdate(insert));
                    System.out.println("updated rows are="+st.executeUpdate(update));
                    ResultSet rs=st.executeQuery("select * from student");
                    System.out.println("student table records are=");
                    while(rs.next())
                    {
                      System.out.println(rs.getInt(1)+"\t"+rs.getString(2)+"\t"+rs.getInt(3));
                    }
                     //step5 close the connection object
                    con.close();
              }
              catch(Exception e){
                    System.out.println(e);
                  }
      }
}
```

<u>output:</u>

```
inserted rows are=1
updated rows are=1
student table records are=
10  subbu  86
```

<u>5.6.3 Execute the Query:</u>

→The SQL statements are executed with the help of 3-methods.

- ResultSet**executeQuery(**String sqlquery) throws SQLException.
- Int**executeUpdate**(String sqlquery) throws SQLException.
- Boolean **execute**(String sqlquery) throws SQLException.

<u>1. executeQuery():</u>

- It is used for executing SQL statements that returns a single ResultSet, e.g: select statement.
- The rows fetched from database are returned as a single ResultSet object.

ResultSetrs=stmt.executeQuery("select *from student");

2. executeUpdate():

- It is used for DDL, DML SQL statements like insert, update, delete & create.
- This method returns an integer value for DML to indicate the number of rows affected/inserted and 0- for DDL statements which don't return anything.

PreparedStatementps=con.prepareStatement("insert into student values (?,?,?,?,?)");

- The values of IN(?) is sent using:

```
ps.setInt(1,1000);

ps.setString(2,"abc");

ps.executeUpdate();
```

3 Execute() method:

It is used for callable statement when the statement may return more than one ResultSet or update counts or combination of both. This happens when stored procedures are executed.

Example program:

```
import java.sql.*;
class ResultSetExample
{
   public static void main(String args[])
    {
      try
       {
         Class.forName("oracle.jdbc.driver.OracleDriver");
  Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","manager");
       Statement st=con.createStatement();
       ResultSet rs=st.executeQuery("select * from student");
       ResultSetMetaData rsmd=rs.getMetaData();
       int c=rsmd.getColumnCount();
       System.out.println("student table records are=");
       while(rs.next())
        {
             System.out.println(rs.getInt(1)+"  "+rs.getString(2)+"  "+rs.getInt(3));
         }
       System.out.println("no of columns in a student table ="+c);
      System.out.println("column name type="+rsmd.getColumnTypeName(2));
      con.close();
    }
    catch(Exception e){ }
  }
}
```

Output:

```
student table records are=
10  subbu  86
no of columns in a student table =3
column name type=VARCHAR2
```

5.6.4 Process the ResultObject

The object of ResultSet maintains a cursor pointing to a row of a table. Initially, cursor points to before the first row.

```
 Resultsetrs =st.executeQuery(sql query);
Ex: Resultsetrs =st.executeQuery(select *from student);
```

Iterate through the results.

- The resultset is iterated with the help of method 'next' which returns a Boolean value to indicate that the ResultSet has more rows to be iterated.
- The next method moves the cursor to the next row.
- The individual column data is obtained by using accessor i.e., getX-method

```
while(rs.next())
{
    System.out.println(rs.getInt(1)+"\t"+ rs.getString(2));
}
```

<u>**ResultSetMetaData object**</u>:

- This object is used to obtain metadata about the ResultSet that include number of columns, types of columns……..
- The method **getColumnCount()** –method returns number of columns in the ResultSet.
- The method **getColumnTypeName(int index)-**returns type of the data the column holds.

Types of Resultsets

In JdbcResultSet Interface are classified into two types;.

- Non-Scrollable ResultSet in JDBC
- Scrollable ResultSet

By default a ResultSet Interface is Non-Scrollable, In non-scrollable ResultSet we can move only in forward direction (that means from first record to last record), but not in Backward Direction, If you want to move in backward direction use **Scrollable Interface**.

Difference between Scrollable ResultSet and Non-Scrollable ResultSet

	Non-Scrollable ResultSet	**Scrollable ResultSet**
1	Cursor move only in forward direction	Cursor can move both forward and backward direction
2	Slow performance, If we want to move nth record then we need to n+1 iteration	Fast performance, directly move on any record.
3	Non-Scrollable ResultSet cursor cannot move randomly	Scrollable ResultSet cursor can move randomly

<u>**Create Scrollable ResultSet:**</u>To create a Scrollable ResultSet, create Statement object with two parameters.

Syntax: Statement stmt=con.CreateStatement(param1, param2); // parm1 type and param2 mode

These type and mode are predefined in ResultSet Interface of Jdbc like below which is static final.

Type:

- public static final int TYPE_FORWARD_ONLY
- public static final int TYPE_SCROLL_INSENSITIVE
- public static final int TYPE_SCROLL_SENSITIVE

Mode:

- public static final int CONCUR_READ_ONLY
- public static final int CONCUR_UPDATABLE

Methods: It has the following methods.

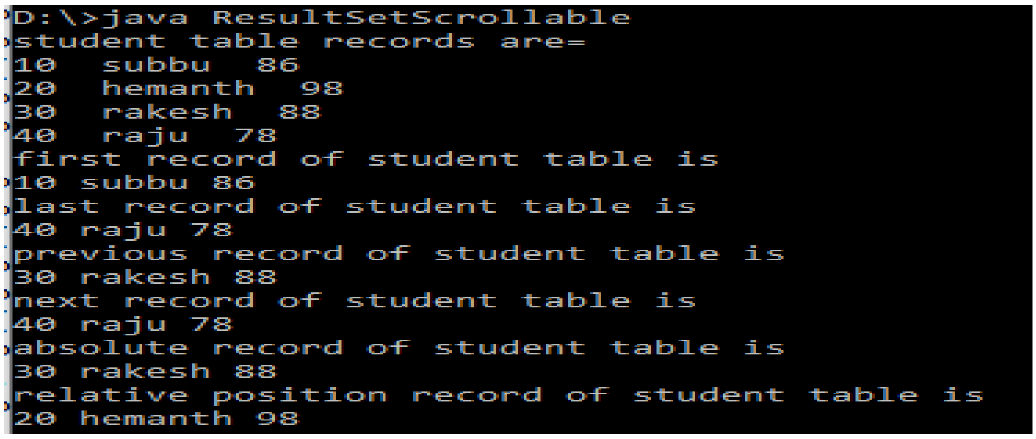
1) public boolean next():	is used to move the cursor to the one row next from the current position.
2) public boolean previous():	is used to move the cursor to the one row previous from the current position.
3) public boolean first():	is used to move the cursor to the first row in result set object.
4) public boolean last():	is used to move the cursor to the last row in result set object.
5) public boolean absolute(int row):	is used to move the cursor to the specified row number in the ResultSet object.
6) public boolean relative(int row):	is used to move the cursor to the relative row number in the ResultSet object, it may be positive or negative.
7) public intgetInt(int columnIndex):	is used to return the data of specified column index of the current row as int.
8) public intgetInt(String columnName):	is used to return the data of specified column name of the current row as int.
9) public String getString(int columnIndex):	is used to return the data of specified column index of the current row as String.
10) public String getString(String columnName):	is used to return the data of specified column name of the current row as String.

<u>Sample program on Scrollable ResultSet:</u>

```
import java.sql.*;

class ResultSetScrollable

{

  public static void main(String args[])

{

   try{

        Class.forName("oracle.jdbc.driver.OracleDriver");

  Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","manager");

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

          ResultSet rs=st.executeQuery("select * from student");//select command execution

          System.out.println("student table records are=");

         while(rs.next())

         {

          System.out.println(rs.getInt(1)+"  "+rs.getString(2)+"  "+rs.getInt(3));

         }

         System.out.println("first record of student table is");

          rs.first();

         System.out.println(rs.getInt(1)+" "+rs.getString(2)+" "+rs.getInt(3));

         System.out.println("last record of student table is");
```

```
      rs.last();

      System.out.println(rs.getInt(1)+" "+rs.getString(2)+" "+rs.getInt(3));

      System.out.println("previous record of student table is");

      rs.previous();

      System.out.println(rs.getInt(1)+" "+rs.getString(2)+" "+rs.getInt(3));

      System.out.println("next record of student table is");

      rs.next();

      System.out.println(rs.getInt(1)+" "+rs.getString(2)+" "+rs.getInt(3));

      System.out.println("absolute record of student table is");

      rs.absolute(3);

      System.out.println(rs.getInt(1)+" "+rs.getString(2)+" "+rs.getInt(3));

      System.out.println("relative position record of student table is");

      rs.relative(-1);

      System.out.println(rs.getInt(1)+" "+rs.getString(2)+" "+rs.getInt(3));

      con.close();

    }catch(Exception e){ }

}

}
```

Output:



```
D:\>java ResultSetScrollable
student table records are=
10   subbu   86
20   hemanth   98
30   rakesh   88
40   raju   78
first record of student table is
10 subbu 86
last record of student table is
40 raju 78
previous record of student table is
30 rakesh 88
next record of student table is
40 raju 78
absolute record of student table is
30 rakesh 88
relative position record of student table is
20 hemanth 98
```

5.6.5 Close the Connection:

public void close(): closes the connection and Releases a JDBC resources immediately.

Ex: con.close();

Here, con is a Connection object reference variable.

5.7 Example programs of JDBC:

Note: Refer 5.6 topic programs.

/* example program for alter drop and delete command*/

```
import java.sql.*;
class JdbcExamples
{
    public static void main(String args[])
    {
      try
      {
         Class.forName("oracle.jdbc.driver.OracleDriver");
Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","manager");
         PreparedStatement ps=con.prepareStatement("alter table student add address varchar2(30)");
         PreparedStatement ps1=con.prepareStatement("delete from student where sno=40");
         PreparedStatement ps2=con.prepareStatement("drop table student");
         ps.executeUpdate();
         System.out.println("table altered="+ps.executeUpdate());
         System.out.println("deleting records="+ps1.executeUpdate());
         System.out.println("droping table is="+ps2.executeUpdate());
         con.close();
      }
      catch(Exception e){ }
  }
}
```