JDBC FUNDAMENTALS

POONAM DHAND
ASSISTANT PROFESSOR
COMPUTER SCIENCE DEPT.
GOVT. COLLEGE FOR GIRLS,LUDHIANA

JDBC – AN INTRODUCTION

JDBC stands for *Java Database Connectivity*. JDBC can also be defined as the platform-independent interface between a relational database and Java programming. JDBC is an *Application Programming Interface* (API) used to connect Java application and execute the query with the database. JDBC allows for accessing any form of tabular data from any source and can interact with various types of Databases such as *Oracle, MS Access, My SQL and SQL Server*. It allows java program to execute SQL statement and retrieve result from database.

FIG 1.1 describes the connection of the JDBC with the Database. JDBC API is used to access tabular data stored in any relational database. JDBC provides a complete set of interfaces that allows for portable access to an underlying database. Java can be used to write different types of executables, such as Java Applications, Java Applets, Java Servlets, Java Server Pages (JSPs), Enterprise JavaBeans (EJBs). These different executables are able to use a JDBC driver to access a database, and also used to store data. By the help of JDBC API, we can save, update, delete and fetch data from the database. It is like Open Database Connectivity (ODBC) provided by Microsoft, allowing Java programs to contain database-independent code.

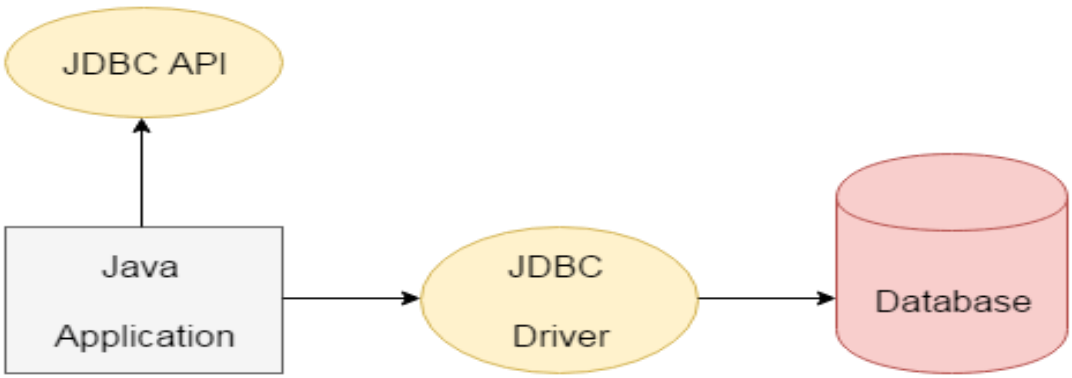


Fig. 1.1 (JDBC-Connection to Database)

JDBC helps you to write Java applications that manage these three programming activities:

- Connect to a data source, like a database
- Send queries and update statements to the database
- Retrieve and process the results received from the database in answer to your query.

Components of JDBC

JDBC includes following components:

- **JDBC API:** JDBC API provides access to relational data from the Java programming language and can execute SQL statements, retrieve results, and propagate changes back to an underlying data source . The JDBC API uses two packages **java.sql** and **javax.sql.**

- **JDBC DriverManager :** The JDBC **DriverManager** class manages various drivers which defines objects that connect Java applications to a JDBC driver. *javax.naming* and *javax.sql* packages are used to register a DataSource object and establish a connection with a data source.
- **Driver**: This interface handles the communications with the database server.
- **Connection:** The *Connectio***n** object represents communication context and provide all the methods and interfaces to communicate with database. All communication with database is through connection object only.
- **Statement:** *Statement* Object is used to create objects from this interface to query the SQL statements to the database. Some derived interfaces accept parameters in addition to executing stored procedures.
- **ResultSet:** *ResultSet* objects hold data retrieved from a database after executing an SQL query using Statement objects.
- **SQLException:** This class handles any errors that occur in a database application.

JDBC Drivers

JDBC drivers enable you to open database connections and to interact with it by sending SQL or database commands then receiving results with Java. The following are the different types of driver available in JDBC.

- **Type-1 Driver** or **JDBC-ODBC bridge** : **Type-1 Driver** act as a bridge between JDBC and other database connectivity mechanism(ODBC). This driver converts JDBC calls into ODBC calls and redirects the request to the ODBC driver.
- **Type-2 Driver** or **Native API Partly 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.

- **Type-3 Driver** or **Network Protocol Driver :** The Network Protocol driver uses middleware (application server) that converts JDBC calls directly or indirectly into the database specific calls. It is fully written in java. This driver translate the JDBC calls into a database server independent and Middleware server-specific calls.
- **Type-4 Driver** or **Thin Driver:** This is Driver called Pure Java Driver because. This driver interacts directly with database. It does not require any native database library that is why it is also known as Thin Driver. It is fully written in Java language.

ESTABLISHING CONNECTION AND CONNECTION INTERFACE

There are various steps which are used to connect any java application with the database using JDBC. These steps are given below:

- Establishing the Connection
  - Register the Driver class
  - Create connection
- Create statement
- Execute queries
- Process the ResultSet Object
- Close connection

Establishing the Connection

The First step of connectivity is to establish a connection with the data source you want to use. A data source can be a DBMS, a valid file system, or some other source of data with a corresponding JDBC driver. JDBC applications connect to a data source using the DriverManager class.

DriverManager Class

The *DriverManager* class acts as an interface between user application and drivers. *DriverManager* class connects an application to a data source, which is specified by a database URL. This class is used for establishing a connection between a database and the appropriate driver and also keeps track of the drivers that are available in the system. The *DriverManager* class maintains a list of Driver classes that have registered themselves by calling the method **DriverManager.registerDriver().** Various methods of DriverManager Class are given below

- *public static Connection getConnection(String url):* is used to establish the connection with the specified url.

- *public static Connection getConnection(String url,String userName,String password):* This method is used to establish the connection with the specified url, username and password.
- *public static void registerDriver(Driver driver):* This method is used to register the given driver with DriverManager.
- *public static void deregisterDriver(Driver driver):* This method is used to deregister the given driver (drop the driver from the list) with DriverManager.

java.sql.DriverManager class should be import to use these above mentioned database connection methods

Registering Driver Class

Registering the driver is the process by which the driver's class file is loaded into the memory. . You must register the driver in your program before you use it.So it can be utilized as an implementation of the JDBC interfaces. You need to do this registration only once in your program. You can register a driver in one of two ways.

- ***Class.forName()***

The ***java.lang.Class.forName()*** method is most common approach to register a driver that dynamically loads the driver's class file into memory. This method automatically registers drivers class. This method is preferable because it allows you to make the driver registration configurable and portable.

Example

```
a) Class.forName("sun.jdbc.odbc.JdbcOdbcDriver")
   // For Registering    JdbcOdbcDriver driver Class.

b) Class.forName("oracle.jdbc.driver.OracleDriver");
   // For Registering    OracleDriver driver Class.
```

- ***DriverManager.registerDriver()***

The second approach you can use to register a driver, is to use the static *DriverManager.registerDriver()* method. The DriverManager class maintains a list of Driver classes that have registered themselves by calling the method *DriverManager.registerDriver().*

Example

```
Driver myDriver = new oracle.jdbc.driver.OracleDriver();
DriverManager.registerDriver( myDriver );
```

Creating Connection

After you've loading and Registering the driver class , you can establish a connection using the **DriverManager.getConnection()** method.

Example:

```
Import java.sql;
Connection Con;
 Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
 con = DriverManager.getConnection(jdbc:odbc: url);
```

Connection Interface

java.sql.Connection interface represents a session between java application and database. The Connection interface is used to create the *java.sql.Statement, java.sql.PreparedStatement,* and *java.sql.DatabaseMetaData* objects which means the object of Connection can be used to get the objects of Statement and *DatabaseMetaData.* All SQL statements are executed and results are returned with in the context of a Connection object. . You can also use it to retrieve the metadata of a database like name of the database product, name of the JDBC driver, version of the database etc. The Connection interface provide many which are given below.

- **public Statement createStatement():** creates a statement object that can be used to execute SQL queries.
- **public Statement createStatement(int resultSetType, int resultSetConcurrency):** This method is used to Create a Statement object that will generate ResultSet objects with the given type and concurrency.
- **public void setAutoCommit(boolean status):** This method is used to set the commit status.By default it is true.
- **public void close():**This method closes the connection and Releases a JDBC resources immediately.
- **public void commit():**This method used to save the changes made since the previous commit/rollback permanent.
- **public void rollback():**This method is used to Drop all changes made since the previous commit/rollback.
-

WORKING WITH STATEMENTS OR CREATING & EXECUTING STATEMENTS

Once the connection has been established you can interact with the database. A *Statement* is an interface that represents a SQL statement. It provides methods to execute queries with the database. You can execute Statement objects, and they

generate ResultSet objects. ResultSet is a table of data representing a database result set. You need a Connection object to create a Statement object.

Creating JDBC Statements

JDBC API provides 3 different interfaces to execute the different types of SQL Queries. The JDBC *Statement, PreparedStatement* and *CallableStatement* interfaces define the methods and properties that enable you to send SQL or PL/SQL commands and receive data from your database and these statements are created by using createStatement(), prepareStatement() and prepareCall() respectively.

- ***Statement***: java.sql.Statement object is used to execute normal SQL queries with no parameters. You can't pass the parameters to SQL query at run time using this interface. Before you can use a Statement object to execute a SQL statement, *you need to create one using the Connection object's createStatement( ) method*. Statement interface is used for DDL statements like CREATE, ALTER, DROP etc.

  Example

```
Connection Con;
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
con = DriverManager.getConnection(jdbc:odbc: url);
Statement stmt=con.CreateStatement();
Con.executeUpdate("CREATE TABLE STUDENT ( ROLLNO NUMBER,
STUDENT_NAME VARCHAR");
```

- ***PreparedStatement*** (Extends Statement.): java.sql.PreparedStatement is used to execute dynamic or parameterized SQL queries. This statement is used for precompiling SQL statements that might contain input parameters. PreparedStatement extends Statement interface. You can pass the parameters to SQL query at run time using this interface.

```
PreparedStatement stmt=con.prepareStatement("UPDATE STUDENT SET
STUDENT_NAME=? WHERE ROLLNO= ?");
stmt.setString(101,"Geeta");
stmt.executeUpdate();
```

- ***CallableStatement***:

  java.sql.CallableStatement is used to execute the stored procedures that may contain both input and output parameters. Three types of parameters exist: IN, OUT, and INOUT. The PreparedStatement object only uses the IN parameter. Before calling the stored procedure, you must register OUT parameters using ***registerOutParameter()*** method of CallableStatement. **Connection.prepareCall()** method is used to instantiate a **CallableStatement** object.

Example

```
CallableStatement  stmt=con.prepareCall(“ Call Procedure1(?,?,?)”);
stmt.execute ();
```

Execute JDBC Statements

The important methods for executing the Statement interface are as follows:

- **public ResultSet executeQuery(String sql):** This method is used for SQL statements which retrieve some data from the database. This method is used to execute SELECT query. It returns the object of ResultSet which contains the data returned by the query.This method is meant to be used for select queries which fetch some data from the database.
- **public int executeUpdate(String sql):** This method is used to execute specified for SQL statements query, it may be create, drop, insert, update, delete etc. is used which update the database in some way. All these statements are DML(Data Manipulation Language) statements. This method can also be used for DDL(Data Definition Language) statements which return nothing.
- **public boolean execute(String sql):** This method is used to execute queries that may return multiple results. This method returns a **boolean** value. **TRUE** indicates that statement has returned a ResultSet object and **FALSE** indicates that statement has returned an int value or returned nothing.
- **public int[] executeBatch():** This method is used to execute batch of commands

Difference between executeQuery(),executeUpdate(),excute()

executeQuery()	**executeUpdate()**	**execute()**
This method is use to execute the sql statements which reterive some data from the database	This method is use to execute the sql statements which update or modify data from the database	This method execute any type of statement.
This method returns a ResultSet object which contains the results returned by the query	This method returns an int value which represents the number of rows affected by the query. This value will be 0 for the statements which return nothing.	This method returns a **boolean** value. **TRUE** indicates that statement has returned a ResultSet object and **FALSE** indicates that statement has returned an int value or returned nothing.
This method used with	This method used to	This method used to

only select queries. Example :Select	execute non select queries . Example DML: INSERT UPDATE and DELETE DDL: CRETE,ALTER	execute for both select and non select queries used for SQL statements .

Difference between Statement, PreparedStatement and CallableStatement

Statement	**PreparedStatement**	**CallableStatement**
It is used to execute normal SQL queries.	It is used to execute parameterized or dynamic SQL queries.	It is used to call the stored procedures.
It is preferred when a particular SQL query is to be executed only once.	It is preferred when a particular query is to be executed multiple times.	It is preferred when the stored procedures are to be executed.
You cannot pass the parameters to SQL query using this interface.	You can pass the parameters to SQL query at run time using this interface.	You can pass 3 types of parameters using this interface. They are – IN, OUT and IN OUT.
This interface is mainly used for DDL statements like CREATE, ALTER, DROP etc.	It is used for any kind of SQL queries which are to be executed multiple times.	It is used to execute stored procedures and functions.
The performance of this interface is very low.	The performance of this interface is better than the Statement interface (when used for multiple execution of same query).	

RESULTSET INTERFACE

A table of data representing a database result set, which is usually generated by executing a statement that queries the database. The result of the query after execution of database statement is returned as table of data according to rows and columns and this data is accessed using the **ResultSet** interface. The *java.sql.ResultSet* interface represents the result set of a database query. A ResultSet object maintains a cursor that points to the current row in the result set. The term "result set" refers to the row and column data contained in a ResultSet object.A default **ResultSet** object is not updatable and the cursor moves only in forward direction.

Types of ResultSet Interface

- **ResultSet.TYPE_FORWARD_ONLY:** The ResultSet can only be navigating forward.
- **ResultSet.TYPE_SCROLL_INSENSITIVE:** The ResultSet can be navigated both in forward and backward direction. It can also jump from current position to another position. The ResultSet is not sensitive to change made by others.
- **ResultSet.TYPE_SCROLL_SENSITIVE :** The ResultSet can be navigated in both forward and backward direction. It can also jump from current position to another position. The ResultSet is sensitive to change made by others to the database.

Methods of ResultSet Interface

Various methods of *ResultSet Interface* is given below.

Methods	**Description**
public boolean absolute(int row)	Moves the cursor to the specified row in the ResultSet object.
public void beforeFirst( )	It moves the cursor just before the first row i.e. front of the ResultSet.
public void afterLast()	Moves the cursor to the end of the ResultSet object, just after the last row.
public boolean first()	Moves the cursor to first value of ResultSet object.
public boolean last( )	Moves the cursor to the last row of the ResultSet object.
public boolean previous ( )	Just moves the cursor to the previous row in the ResultSet object.
public boolean next( )	It moves the curser forward one row from its current position.
public void relative(int rows)	It moves the cursor to a relative number of rows.
public void updateRow()	Updates the current row by updating the corresponding row in the database.
public void deleteRow()	Deletes the current row from the database
public void refreshRow()	Refreshes the data in the result set to reflect any recent changes in the database.
public void cancelRowUpdates()	Cancels any updates made on the current row.
public void insertRow()	Inserts a row into the database. This method can only be invoked when the cursor is pointing to the insert row.
public void updateString(int columnIndex, String s)	Changes the String in the specified column to the value of s.

public void updateString(String columnName, String s)	Similar to the previous method, except that the column is specified by its name instead of its index.
public int getInt(intcolIndex)	It retrieves the value of the column in current row as int in given ResultSet object.
public String getString (intcolIndex)	It retrieves the value of the column in current row as int in given ResultSet object.
public String getString(intcolIndex)	It retrieves the value of the column in current row as int in given ResultSet object.

Creating ResultSet Interface

To execute a *Statement or PreparedStatement,* You create *ResultSet* object. JDBC provides the following connection methods to create statements with desired ResultSet are createStatement(), prepareStatement(), prepareCall(). Example

```
Statement stmt = connection.createStatement();
ResultSet result = stmt.executeQuery("select * from Students");
Or
String sql = "select * from Students";
PreparedStatementstmt = con.prepareStatement(sql);
ResultSet result = stmt.executeQuery();
```

WORKING WITH SQL STATEMENTS

A Statement object sends SQL statements to a database. There are three kinds of Statement objects. Each is specialized to send a particular type of SQL statement:

- A Statement object is used to execute a simple SQL statement with noparameters
- A PreparedStatement object is used to execute a pre-compiled SQL statement with or without IN parameters.
- A CallableStatement object is used to execute a call to a database stored procedure.

You must construct a Statement object before executing an SQL statement. The Statement object offers a way to send a SQL statement to the server (and gain access to the result set). Each Statement object belongs to a Connection; use the *createStatement()* method to ask the Connection to create the Statement object and executeUpdate(), executeQuery() is used for execution.

JDBC : Insert Record

Insert command in SQL is used to insert record in the Database. Example

```
Connection con;
 url="jdbc:odbc:java1";
 Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
   con= DriverManager.getConnection(url);
```

```
  stmt=conn.createStatement();
  String sqlI = "INSERT INTO EMP_INFO " + "(EMPNO, DEPT, ENAME, FATHERNAME)"+" VALUES ('"+eno+"','"+dept+"','"+empname+"','"+fname+"')";
stmt.executeUpdate(sqlI);
```

JDBC : Update Record

Update command in SQL is used to update or modify values in the Database. Example

```
Connection con;
 url="jdbc:odbc:java1";
 Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
  con= DriverManager.getConnection(url);
  stmt=conn.createStatement();
  String sql = "UPDATE EMP_INFO " + "SET  EMPNO= '"+eno+"', DEPT= '"+dept+"', ENAME='"+empname+"', FATHERNAME = '"+fname+"'  WHERE EMPNO='"+eno+"'" ;
stmt.executeUpdate(sql);
```

JDBC : Delete Record

Delete command in SQL is used to delete record in the Database. Example

```
Connection con;
 url="jdbc:odbc:java1";
 Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
  con= DriverManager.getConnection(url);
  stmt=conn.createStatement();
  String  sql = "DELETE FROM  EMP_INFO  WHERE EMPNO ='"+eno+"'";
stmt.executeUpdate(sql);
```

JDBC : Select Record from Database

Delete command in SQL is used to delete record in the Database. Example

```
Connection con;
 url="jdbc:odbc:java1";
 Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
  con= DriverManager.getConnection(url);
  stmt=conn.createStatement();
  String  sql="SELECT EMPNO from EMP_INFO WHERE  EMPNO='"+eno+"'";
stmt.executeQuery(sql);
```

STEPS TO CONNECT A JAVA APPLICATION TO DATABASE USING JDBC-ODBC Bridge

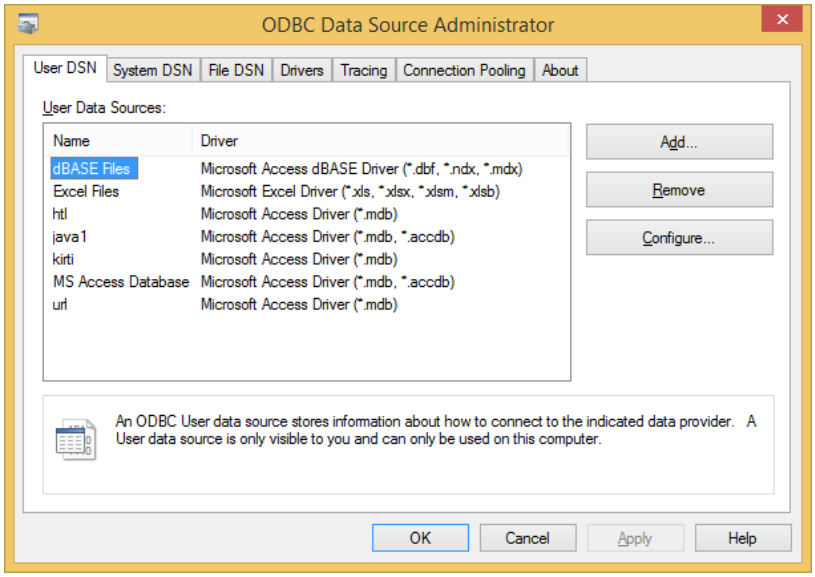
To Connect the JDBC and ODBC we should have a *database. Then we would be required to create a DSN to use JDBC ODBC Bridge driver. The DSN (Data Source Name) specifies the connection of an ODBC to a specific server.* As its name JDBC-

ODBC bridge, it acts like a bridge between the Java Programming Language and the ODBC to use the JDBC API. To use the JDBC API with the existing ODBC Sun Microsystems (Now Oracle Corporation) provides the driver named **JdbcOdbcDriver**. Full name of this class is sun.jdbc.odbc.JdbcOdbcDriver.

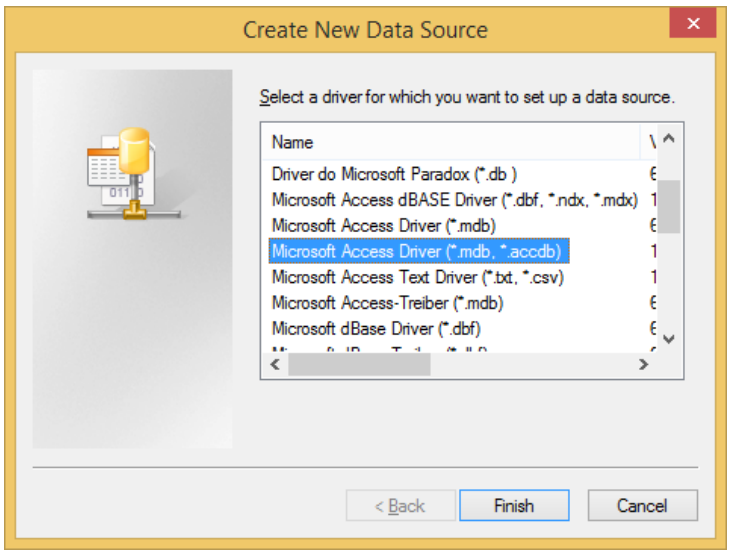
Steps to Create DSN in java

To create a DSN we would be required to follow some basic steps. Before creating DSN to use MS-Access with JDBC technology the database file should be created in advance These steps are as follows :.

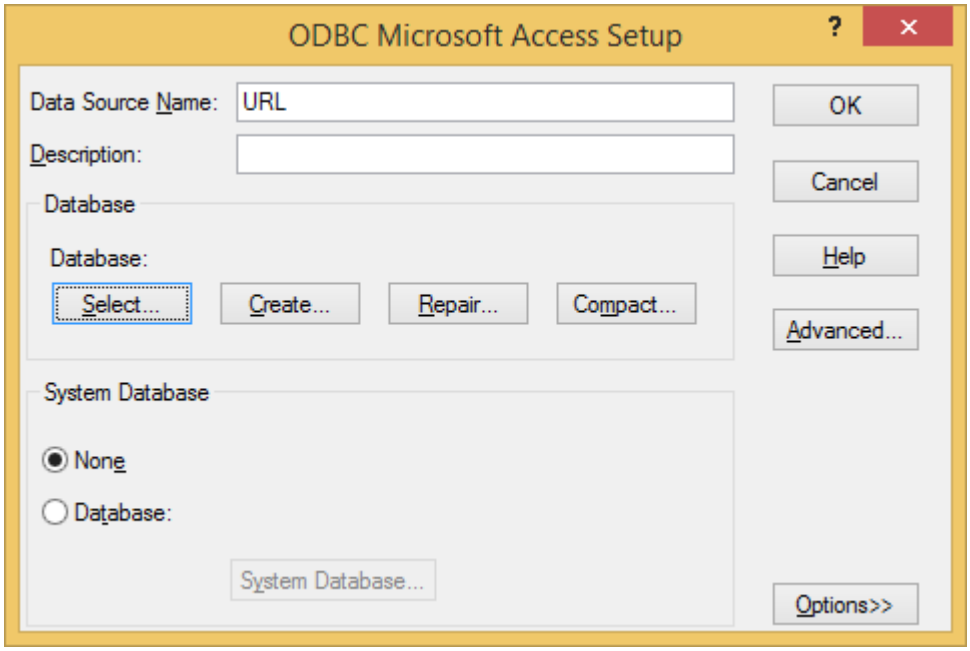
- First step is to go through the *Control Panel* → *Administrative Tools* → Data Sources (ODBC).



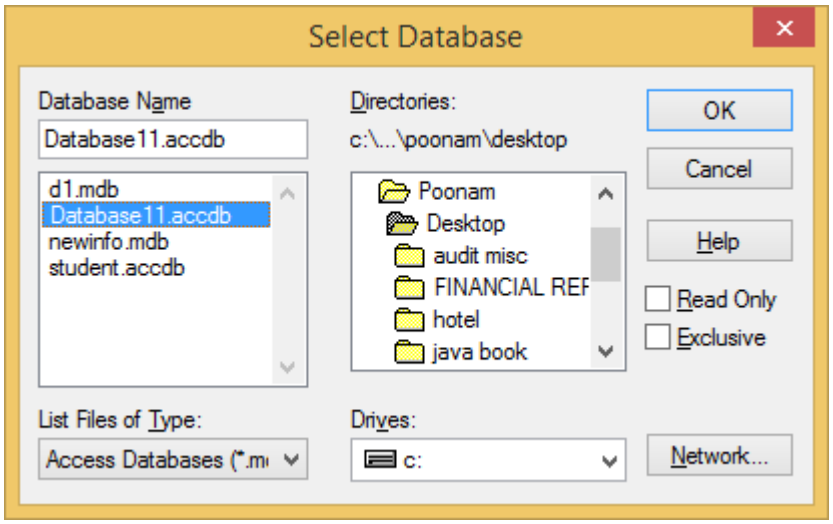
- Go to the tab System DSN in the ODBC Data Source Administrator dialog box then, Click on Add button→ select MS Access Driver (*.mdb) OR MS Access Driver (*.accdb, *.mdb) -> click on Finish button.



- In the next step a dialog box ODBC Microsoft Access Setup will be opened then provide the field's values corresponding to the field's label (i.e. provide the DSN name as you wish) and then click on the "Select" button.



- In the next step a new dialog box "Select database" will be opened.



Here you have to select the directory where you have stored your file and provide the name in the place of the Database Name and then press ok → Ok.

Steps to Create a Database Connection

There are following steps to create the database connection with Java application**:**

- Import JDBC packages
- Register the Driver

- Create Connection
- Create SQL Statement
- Execute SQL Queries
- Close the Connection

1) Import JDBC packages : First step to include the packages containing the JDBC classes needed for database programming. Most often, using *import java.sql.** package.

2) Register the driver

The **Class.forName()** method is used to register the driver class dynamically.

For example:

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver")
                        or
Class.forName("oracle.jdbc.odbc.JdbcOdbcDriver");
```

3) Create the Connection Object

The **DriverManager** class provides the **getConnection()** method to establish connection object. It requires to pass a database url, username and password.

Example : Creating connection with jdbc-odbc driver

```
Import java.sql;
  Connection Con;
   Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
   con = DriverManager.getConnection(jdbc:odbc: url);//here url is dsn which is created by user
```

4) Create SQL Statement

The **Connection interface** provides the **createStatement()** method to create SQL statement.

Example:

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

5) Execute SQL Queries

The **Statement interface** provides the **executeQuery( )** , executeUpdate() and execute ()method to execute SQL statements.

Example

```
ResultSet rs = stmt.executeQuery("select * from students");
while (rs.next())
{
    System.out.println (rs.getInt(1)+" "+rs.getString(2)+" "+rs.getFloat(3));
}
```

6) Closing the Connection

The **Connection interface** provides **close( )** method, used to close the connection. It is invoked to release the session after execution of SQL statement.

Syntax:

public void close( ) throws SQLException

Example:

con.close( );

Program for creating a Employee information form having attributes EmpNo,department, name and father name and also done the connectivity with access database using JDBC –ODBC connectivity.

```
import java.lang.*;
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
import java.sql.*;
import java.util.*;
public class Emp_Record t extends JFrame implements ActionListener
{
 String url,sql,eno,dept,empname,fname,doav,sqlI,sqlU;
 String[] rwData;

 int response,stmtval,rwcnt,rec,coln,flag,reply,nrows;
 Vector dataRows;
 Font fnt,head;
 JButton jbtnaddrec;
 JButton jbtndel;
 JButton jbtnupdate;
 JButton jbtnview;
 JButton jbtnfirst;
 JButton jbtnprev;
 JButton jbtnnext;
 JButton jbtnlast;
 JButton jbtnsave;
 JButton jbtnexit;

 JLabel jlblfrmhead;
 JLabel lbleno;
 JLabel lbldept;
 JLabel lblname;
 JLabel lblfname;
```

```
 JTextField jtxteno;
 JTextField jtxtdept;
 JTextField jtxtname;
 JTextField jtxtfname;

 Connection conn;
 Statement stmt;
 ResultSet rs;
 Container container;

public Emp_ record()
 {
  super(" EMPLOYEE  FORM");
  container=this.getContentPane();
  jbtnaddrec = new JButton("Insert");
  jbtndel = new JButton("Delete");
  jbtnupdate = new JButton("Update");
  jbtnview = new JButton("View");
  jbtnfirst = new JButton("First");
  jbtnprev = new JButton("<<");
  jbtnnext = new JButton(">>");
  jbtnlast = new JButton("Last");
  jbtnsave = new JButton("Save");
  jbtnexit = new JButton("Exit");


  jbtnaddrec.setMnemonic('I');
  jbtnaddrec.setToolTipText("Adds a new Record to the table");

  jbtnupdate.setMnemonic('U');
  jbtnupdate.setToolTipText("Update a record in the table");

  jbtndel.setMnemonic('D');
  jbtndel.setToolTipText("Delete an Existing Record from the table");

  jbtnview.setMnemonic('V');
  jbtnview.setToolTipText("TO view the records in the table");

  jbtnfirst.setMnemonic('F');
  jbtnfirst.setToolTipText("To view the first record in the table");

  jbtnprev.setMnemonic('P');
  jbtnprev.setToolTipText("To view the Previous record");
```

```
jbtnnext.setMnemonic('N');
jbtnnext.setToolTipText("To view the next record ");

jbtnlast.setMnemonic('L');
jbtnlast.setToolTipText("To view the Last record in the table");

jbtnsave.setMnemonic('S');
jbtnsave.setToolTipText("Save records");

jbtnexit.setMnemonic('X');
jbtnexit.setToolTipText("Exit Application");

jlblfrmhead = new JLabel("  EMPLOYEE REGISTRATION FORM");
lbleno = new JLabel("Emp_No.:");
lbldept = new JLabel("Department:");
lblname = new JLabel("Name:");
lblfname = new JLabel("Father_Name:");

jtxteno=new JTextField(10);
jtxtdept = new JTextField(20);
jtxtname = new JTextField(15);
jtxtfname = new JTextField(15);

dataRows = new Vector();

jbtnaddrec.addActionListener(this);
jbtndel.addActionListener(this);
jbtnupdate.addActionListener(this);
jbtnview.addActionListener(this);
jbtnfirst.addActionListener(this);
jbtnprev.addActionListener(this);
jbtnnext.addActionListener(this);
jbtnlast.addActionListener(this);
jbtnsave.addActionListener(this);
jbtnexit.addActionListener(this);
container.setLayout(null);
fnt = new Font("TIMES NEW ROMAN",Font.BOLD,15);

jbtnaddrec.setFont(fnt);
jbtndel.setFont(fnt);
jbtnupdate.setFont(fnt);
jbtnview.setFont(fnt);
```

```
jbtnfirst.setFont(fnt);
jbtnprev.setFont(fnt);
jbtnnext.setFont(fnt);
jbtnlast.setFont(fnt);
jbtnsave.setFont(fnt);
jbtnexit.setFont(fnt);

lbleno.setFont(fnt);
lbldept.setFont(fnt);
lblname.setFont(fnt);
lblfname.setFont(fnt);

head= new Font("TIMES NEW ROMAN",Font.BOLD,15);
jlblfrmhead.setBounds(455,25,375,25);
jlblfrmhead.setFont(head);
container.add(jlblfrmhead);

jbtnaddrec.setBounds(450,80,75,25);
container.add(jbtnaddrec);

jbtnupdate.setBounds(540,80,75,25);
container.add(jbtnupdate);

jbtndel.setBounds(630,80,75,25);
container.add(jbtndel);

jbtnview.setBounds(720,80,75,25);
container.add(jbtnview);

jbtnfirst.setBounds(450,120,75,25);
container.add(jbtnfirst);

jbtnprev.setBounds(540,120,75,25);
container.add(jbtnprev);

jbtnnext.setBounds(630,120,75,25);
container.add(jbtnnext);

jbtnlast.setBounds(720,120,75,25);
container.add(jbtnlast);

lbleno.setBounds(420,220,100,25);
container.add(lbleno);
```

```
 jtxteno.setBounds(560,220,100,25);
 container.add(jtxteno);

 lbldept.setBounds(420,255,100,25);
 container.add(lbldept);

 jtxtdept.setBounds(560,255,100,25);
 jtxtdept.setToolTipText("Department");
 container.add(jtxtdept);

 lblname.setBounds(420,290,100,25);
 container.add(lblname);

 jtxtname.setBounds(560,290,100,25);
 jtxtname.setToolTipText("Employee name");
 container.add(jtxtname);

 lblfname.setBounds(420,325,100,25);
 container.add(lblfname);

 jtxtfname.setBounds(560,325,100,25);
 jtxtfname.setToolTipText("Father's name");
 container.add(jtxtfname);

 jbtnsave.setBounds(420,400,75,25);
 container.add(jbtnsave);

 jbtnexit.setBounds(560,400,75,25);
 container.add(jbtnexit);

 setSize(1300,1300);
 show();
 connection();
 System.out.println("\n Connection Called");
 setResizable(false);
}

public void actionPerformed(ActionEvent actEvt)
{
 if(actEvt.getSource()==jbtnaddrec)
 {
  addRcrd();
  flag=1;
```

```
}
if(actEvt.getSource()==jbtnupdate)
{
 updtRcrd();
 flag=2;
}
if(actEvt.getSource()==jbtndel)
{
delRcrd();
}
if(actEvt.getSource()==jbtnview)
{
 jbtnview.setEnabled(false);
 vwRcrd();
firstRcrd();
}
if(actEvt.getSource()==jbtnfirst)
{
 firstRcrd();
}
if(actEvt.getSource()==jbtnprev)
{
 prevRcrd();
}
if(actEvt.getSource()==jbtnnext)
{
 nextRcrd();
}
if(actEvt.getSource()==jbtnlast)
{
 lastRcrd();
}
if(actEvt.getSource()==jbtnsave)
{
applyRcrd();
}
if(actEvt.getSource()==jbtnexit)
{
 try
 {
  stmt.close();
  conn.close();
  System.exit(0);
```

```
   }
   catch(SQLException sqlexcep)
   {
    System.out.println("Error:" +sqlexcep);
   }
  }
 }

 public void connection()
 {

  try
  {
url="jdbc:odbc:java1";
   Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
   conn = DriverManager.getConnection(url);
   stmt=conn.createStatement();

   System.out.println("Successfully Connected!!!");
   disableBtns();
   jbtnsave.setEnabled(false);
   disableFlds();
  }
  catch(SQLException sqlExcep)
  {
   System.out.println("ERROR:" +sqlExcep);
  }
  catch(ClassNotFoundException clsntExcp)
  {
   System.out.println("ERROR:" +clsntExcp);
  }
 }

 public void addRcrd()
 {
  jbtnsave.setEnabled(true);
  try
  {
   eno=JOptionPane.showInputDialog(null,"Enter Employee number");
   System.out.println(eno);
   if((eno.trim()).length()==0)
   {
    disableFlds();
```

```
    return;
   }
   else
   {
    for(int i=0;i<eno.length();i++)
    {
     if(!Character.isDigit(eno.charAt(i)))
     {
       JOptionPane.showMessageDialog(null,"Enter Digits Only in employee id
field !!!","Note",2);
       return;
     }
    }
   }
   sql="SELECT EMPNO from EMP_INFO WHERE  EMPNO='"+eno+"'";

   try
   {
    rs = stmt.executeQuery(sql);

while(rs.next())
    {
     JOptionPane.showMessageDialog(null,"Record Exists!!!","Note",2);
     disableFlds();
     return;
    }
    enableFlds();
    clsAllFlds();
    jtxteno.setText(eno);
    jtxtdept.requestFocus();
   }
   catch(SQLException sqlexcep)
   {
    System.out.println("ERROR:"+sqlexcep);
   }
  }
  catch(NullPointerException nullPntrExcep)
  {
   return;
  }
 }

 public void updtRcrd()
```

```
 {
  eno=jtxteno.getText();
  if(eno.length()==0)
  {
   JOptionPane.showConfirmDialog(null,"Nothing to Update Please Press the View
Button !!","Note",2);
   jbtnview.setEnabled(true);
  }
  else
  {
   enableFlds();
   jbtnsave.setEnabled(true);
   jtxtdept.requestFocus();
  }
 }

 public void delRcrd()
 {
  int stmtVal;
  disableFlds();
  eno= jtxteno.getText();
  if (eno.length() == 0)
  {
   JOptionPane.showConfirmDialog(null,"Nothing to Delete Please press the View
button First !!!","Note",2);
  }
  else
  {
   response = JOptionPane.showConfirmDialog(null,"Do You Really Want To Delete
The Record?");
   if (response ==JOptionPane.YES_OPTION)
   {
    sql = "DELETE FROM  EMP_INFO  WHERE EMPNO ='"+eno+"'";
    try
    {
     stmtVal=stmt.executeUpdate(sql);
     if(stmtVal==1)
     {
      clsAllFlds();
      JOptionPane.showMessageDialog(null,"Record Deleted !! Press the view button
to refresh the Form!!!","Note",2);
     }
    }
```

```
    catch(SQLException sqlExcep)
    {
     System.out.println("Error " + sqlExcep);
    }
   }
   if (response == JOptionPane.NO_OPTION)
   {
    return ;
   }
  }
  jbtnview.setEnabled(true);
 }
//for Viewing Record
 public void vwRcrd()
 {
  int rwCnt;
  dataRows.removeAllElements();
  rec=0;
  sql = "SELECT * from  EMP_INFO";
  rwCnt=1;
  try
  {
   rs = stmt.executeQuery(sql);
   coln=rs.getMetaData().getColumnCount();
   while(rs.next())                    // For each row.....................................
   {
    rwData=new String[coln];
    for (int i=0; i<coln ; i++)
    {
     rwData[i]= rs.getString(i+1);
    }
    dataRows.addElement(rwData);
    rwCnt++;
   }
  }
  catch (SQLException sqlExcep)
  {
   System.out.println("ERROR :"+sqlExcep);
  }
  enableBtns();
 }

// Method for viewing the first record.
```

```
 public void firstRcrd()
 {
  rec=0;
  disableFlds();
  jbtnnext.setEnabled(true);
  jbtnlast.setEnabled(true);
  jbtnprev.setEnabled(false);
  jbtnfirst.setEnabled(false);

// display the rows contents (columns).
   eno= ((String []) (dataRows.elementAt (rec))) [0];
   dept = ((String []) (dataRows.elementAt (rec))) [1];
   empname = ((String []) (dataRows.elementAt (rec))) [2];
   fname= ((String []) (dataRows.elementAt (rec))) [3];

   popFlds();
 }
//Method for viewing previous record
 public void prevRcrd()
 {
  rec=rec-1;
  disableFlds();
  enableBtns();
  if (rec >= 0)
  {
   if (rec==0)
   {
    jbtnprev.setEnabled(false);
    jbtnfirst.setEnabled(false);
   }
   eno= ((String []) (dataRows.elementAt (rec))) [0];
   dept = ((String []) (dataRows.elementAt (rec))) [1];
   empname= ((String []) (dataRows.elementAt (rec))) [2];
   fname = ((String []) (dataRows.elementAt (rec))) [3];

  }
  popFlds();
 }

// Method for viewing the Next Record
 public void nextRcrd()
 {
```

```
  rec=rec+1;
  disableFlds();
  enableBtns();

// display the rows contents (columns).
  if (rec < dataRows.size())
  {
   if (rec == dataRows.size()-1)
   {
    jbtnnext.setEnabled(false);
    jbtnlast.setEnabled(false);
   }
   eno= ((String []) (dataRows.elementAt (rec))) [0];
   dept = ((String []) (dataRows.elementAt (rec))) [1];
   empname= ((String []) (dataRows.elementAt (rec))) [2];
   fname = ((String []) (dataRows.elementAt (rec))) [3];

  }
  popFlds();
 }


// Method for viewing the last record.

 public void lastRcrd()
 {
  enableFlds();
  jbtnnext.setEnabled(false);
  jbtnlast.setEnabled(false);
  jbtnprev.setEnabled(true);
  jbtnfirst.setEnabled(true);
  rec=dataRows.size()-1;
  eno= ((String []) (dataRows.elementAt (rec))) [0];
  dept = ((String []) (dataRows.elementAt (rec))) [1];
  empname= ((String []) (dataRows.elementAt (rec))) [2];
  fname = ((String []) (dataRows.elementAt (rec))) [3];
   popFlds();
 }

// Method for insert and update the record after clicking the save button
 public void applyRcrd()
 {
  getProdVal();          //.....................................
```

```
  if(eno.length() == 0)
  {
   JOptionPane.showMessageDialog(null,"Employee id Cannot Be
Blank !!!","Note",2);
   jtxteno.requestFocus();
   return;
  }
  if (dept.length() == 0)
  {
   JOptionPane.showMessageDialog(null,"Department Cannot Be
Blank !!!","Note",2);
   jtxtdept.requestFocus();
   return;
  }
  if (empname.length() == 0)
  {
   JOptionPane.showMessageDialog(null,"Name Cannot be left blank !!!","Note",2);
   jtxtname.requestFocus();
   return;
  }
  else
  {
   for(int i=0;i<empname.length();i++)
   {
    if(Character.isDigit(empname.charAt(i)))
    {
     JOptionPane.showMessageDialog(null,"Enter Characters Only in Name
Field !!!","Note",2);
     jtxtname.requestFocus();
     return;
    }
   }
  }
  if (fname.length() == 0)
  {
   JOptionPane.showMessageDialog(null," Father's Name field cannot be
blank !!!","Note",2);
   jtxtfname.requestFocus();
   return;
  }
  else
  {
   for(int i=0;i<fname.length();i++)
```

```
   {
    if(Character.isDigit(fname.charAt(i)))
    {
     JOptionPane.showMessageDialog(null,"Enter Characters Only in Father's Name
Field!!!","Note",2);
     jtxtfname.requestFocus();
     return;
    }
   }
  }

  String sqlI = "INSERT  INTO  EMP_INFO " +
"(EMPNO,DEPT,ENAME,FATHERNAME)"+" VALUES
('"+eno+"','"+dept+"','"+empname+"','"+fname+"')";

 String sqlU = "UPDATE EMP_INFO " + "SET  EMPNO= '"+eno+"', DEPT= '"+dept+"',
ENAME='"+empname+"', FATHERNAME = '"+fname+"'  WHERE EMPNO='"+eno+"'" ;
 try
 {
  enableBtns();
  if ( flag == 1 )
  {
   reply = JOptionPane.showConfirmDialog(null, "Insert Changes ??");
   if (reply ==JOptionPane.YES_OPTION)
   {
    nrows = stmt.executeUpdate(sqlI);
    JOptionPane.showConfirmDialog(null,"Changes Saved !!  want to
Continue !!!","Note",2);
    if(reply==JOptionPane.YES_OPTION)
    { clsAllFlds();

    }
    else
    {
     clsAllFlds();
    }
   }
   if (reply == JOptionPane.NO_OPTION)
   {
    return;
   }
   jbtnview.setEnabled(true);
   disableFlds();
```

```
  }
  if(flag == 2)
  {
   reply = JOptionPane.showConfirmDialog(null,"Save Changes ??");
   if(reply == JOptionPane.YES_OPTION)
   {
    getProdVal();
    nrows = stmt.executeUpdate(sqlU);
    JOptionPane.showMessageDialog(null,"Changes Saved !! Press the view button
to refresh the Records!!!","Note",2);
    clsAllFlds();
   }
   if(reply == JOptionPane.NO_OPTION)
   {
    return;
   }
   jbtnview.setEnabled(true);
   disableFlds();
   jbtnsave.setEnabled(false);
  }
 }
 catch(SQLException sqlExcep)
 {
  System.out.println("Error "+ sqlExcep);
 }
}


public void disableFlds()
{
 jtxteno.setEnabled(false);
 jtxtdept.setEnabled(false);
 jtxtname.setEnabled(false);
 jtxtfname.setEnabled(false);
 }

public void enableBtns()
{
 jbtnfirst.setEnabled(true);
 jbtnnext.setEnabled(true);
 jbtnprev.setEnabled(true);
 jbtnlast.setEnabled(true);
}
```

```
public void disableBtns()
{
 jbtnfirst.setEnabled(false);
 jbtnnext.setEnabled(false);
 jbtnprev.setEnabled(false);
 jbtnlast.setEnabled(false);
}

public void enableFlds()
{
 jtxteno.setEnabled(true);
 jtxtdept.setEnabled(true);
 jtxtname.setEnabled(true);
 jtxtfname.setEnabled(true);

}

public void clsAllFlds()
{
 jtxteno.setText("");
 jtxtdept.setText("");
 jtxtname.setText("");
 jtxtfname.setText("");

}

public void popFlds()
{
 jtxteno.setText(eno);
 jtxtdept.setText(dept);
 jtxtname.setText(empname);
 jtxtfname.setText(fname);

}

public void getProdVal()
{
 eno =jtxteno.getText();
 dept =jtxtdept.getText();
 empname =jtxtname.getText();
 fname=jtxtfname.getText();

}
```

```
public static void main(String args[])
{
 Emp_Record  ER= new Emp_Record  ();
WindowListener l1=new WindowAdapter()
 {
  public void windowClosing(WindowEvent winEvt)
  {System.exit(0);}
 };
 ER.addWindowListener(l1);
 }
}
```

Output Screens

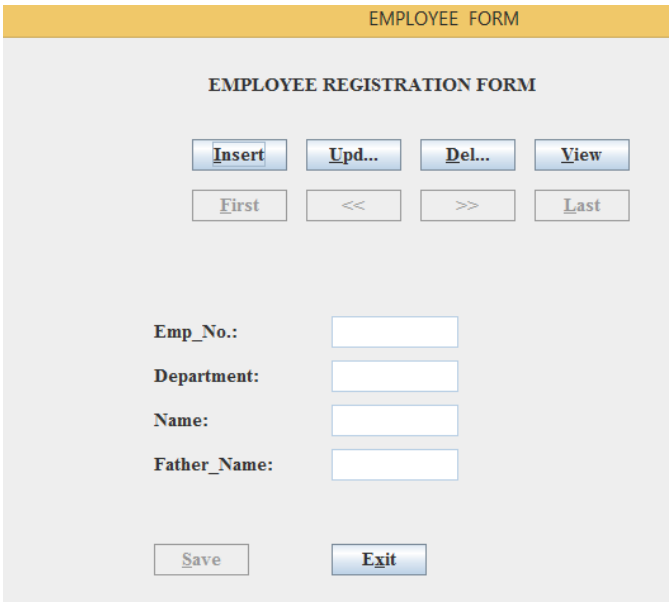
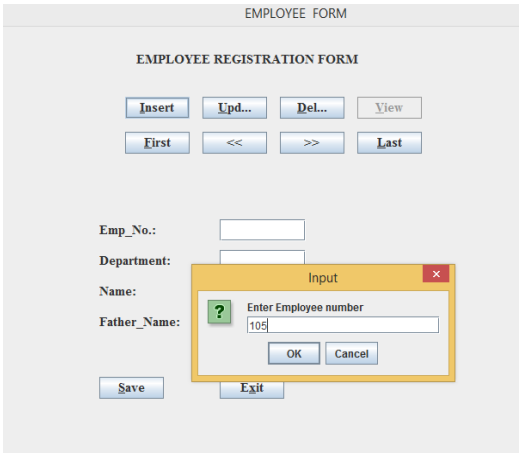


Fig. (Employee information system-Main Layout)



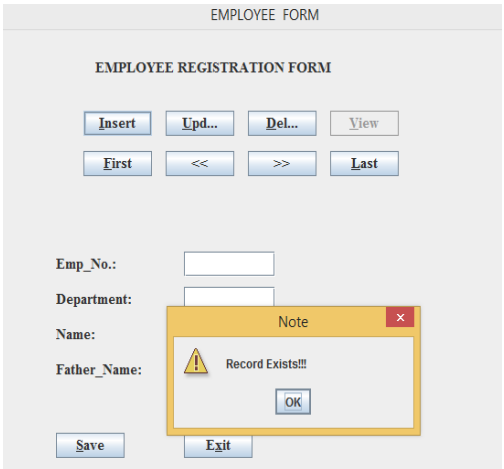


Fig.(Employee information System: Insert Record)

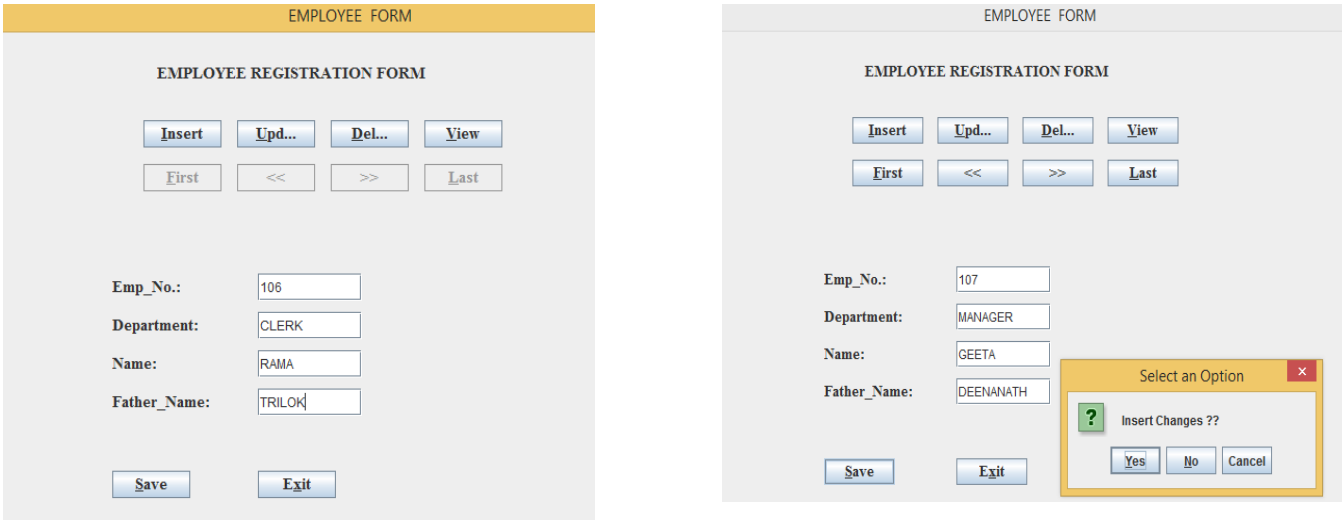


Fig. (Employee information system: Insert Record (Next Step))

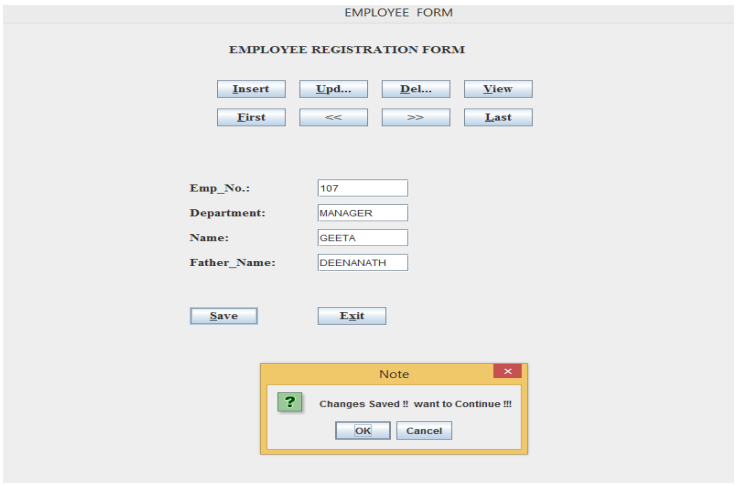


Fig. (Employee information system: Insert Record (Next Step))

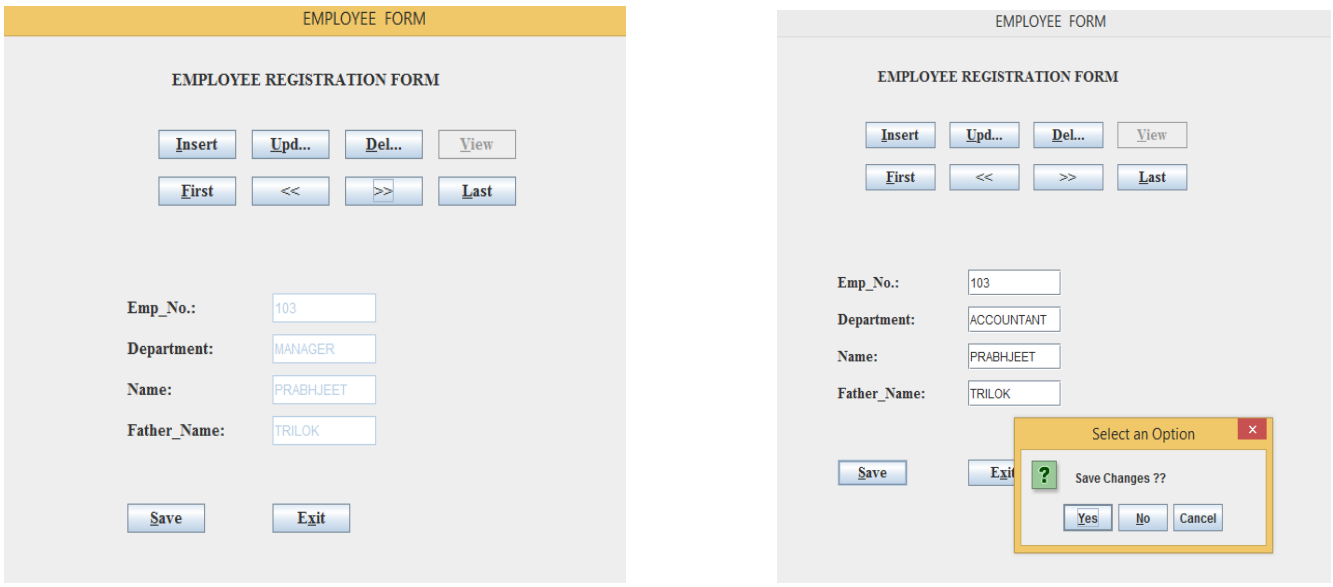


Fig. (Employee information system: Update Record)

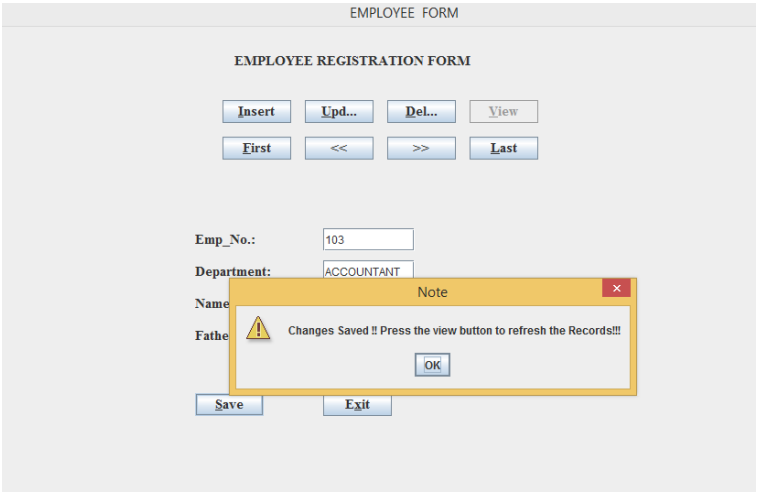


Fig. (Employee information system: Update Record (Next Step))

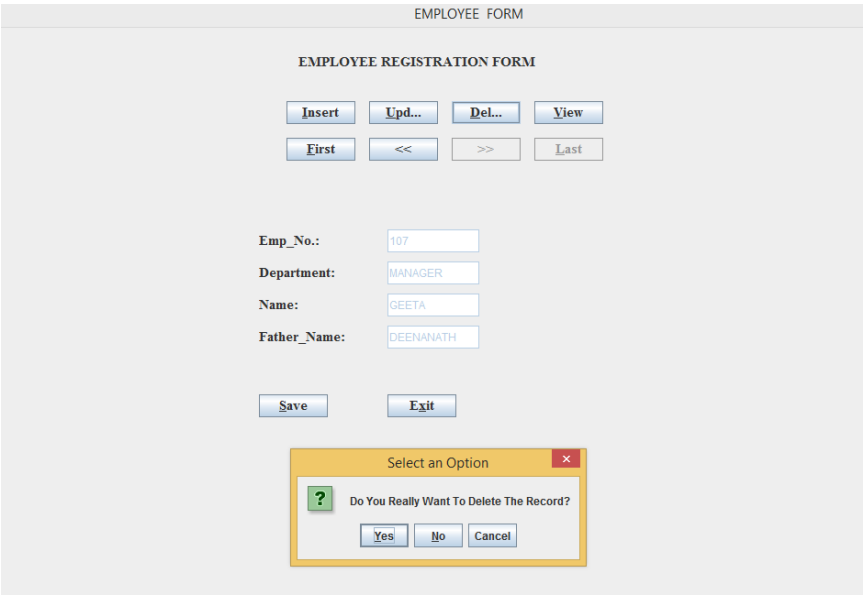


Fig. (Employee information system: Delete Record)

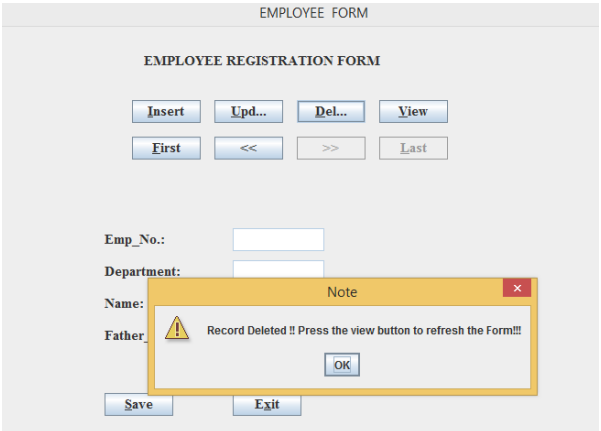


Fig. (Employee information system: Delete Record (Next Step))

EMP_INFO

EMPNO	DEPT	ENAME	FATHERNAME
101	MANAGER	RAM PARKASH	CHANDRPAL
102	ACCOUNT	POOJA	PUNEET
103	ACCOUNTANT	PRABHJEET	TRILOK
104	MANAGER	RIDHIMA	BRIJ MOHAN
105	ACCOUNT	PARAMJEET	GURPREET SINGH

Fig. (Access Database-Employee(Emp_Info Table Name)

QUESTIONS

➢ SHORT ANSWER QUESTIONS

Q1. Define JDBC.

Ans: JDBC stands for *Java Database Connectivity*. JDBC can also be defined as the platform-independent interface between a relational database and Java programming. JDBC is an *Application Programming Interface* (API) used to connect Java application and execute the query with the database. JDBC allows for accessing any form of tabular data from any source and can interact with various types of Databases such as *Oracle, MS Access, My SQL and SQL Server*. It allows java program to execute SQL statement and retrieve result from database.

Q2. Define DriverManager.

Ans: The JDBC **DriverManager** class manages various drivers which defines objects that connect Java applications to a JDBC driver. *javax.naming* and *javax.sql* packages are used to register a DataSource object and establish a connection with a data source.

Q3. What is the use of executeUpdate()?

Ans: *The DSN (Data Source Name) specifies the connection of an ODBC to a specific server.* As its name JDBC-ODBC bridge, it acts like a bridge between the Java Programming Language and the ODBC to use the JDBC API.

Q4. Define PreparedStatement?

Ans: PreparedStatement is used to execute dynamic or parameterized SQL queries. This statement is used for precompiling SQL statements that might contain input parameters. PreparedStatement extends Statement interface. You can pass the parameters to SQL query at run time using this interface.

Q5. Define ResultSet Interface?

Ans: The result of the query after execution of database statement is returned as table of data according to rows and columns and this data is accessed using the **ResultSet** interface. The *java.sql.ResultSet* interface represents the result set of a database query. A ResultSet object maintains a cursor that

points to the current row in the result set. The term "result set" refers to the row and column data contained in a ResultSet object.A default **ResultSet** object is not updatable and the cursor moves only in forward direction.

Q6. Explain Various Methods of Connection Inetface.

Ans: The Connection interface provides various methods which are given below.

- **public Statement createStatement():** creates a statement object that can be used to execute SQL queries.
- **public Statement createStatement(int resultSetType, int resultSetConcurrency):** This method is used to Create a Statement object that will generate ResultSet objects with the given type and concurrency.
- **public void setAutoCommit(boolean status):** This method is used to set the commit status.By default it is true.
- **public void close():**This method closes the connection and Releases a JDBC resources immediately.
- **public void commit():**This method used to save the changes made since the previous commit/rollback permanent.
- **public void rollback():**This method is used to Drop all changes made since the previous commit/rollback.

Q7. What is executeUpdate()?

Ans. This method is used to execute specified for SQL statements query, it may be create, drop, insert, update, delete etc. is used which update the database in some way. All these statements are DML(Data Manipulation Language) statements. This method can also be used for DDL(Data Definition Language) statements which return nothing.

➢ Long Answer Questions

Q1. How will u create and execute various Statemnts in JDBC?

Ans: Refer Section 6.3

Q2. What will you establish connection in JDBC

Ans: Refer Section 6.2

Q3. Difference b/w executeQuery(), executeUpdate() and execute().

Ans:

executeQuery()	**executeUpdate()**	**execute()**
This method is use to execute the sql statements which reterive some data from the database	This method is use to execute the sql statements which update or modify data from the database	This method execute any type of statement.
This method returns a ResultSet object which	This method returns an int value which	This method returns a **boolean** value. **TRUE**

contains the results returned by the query	represents the number of rows affected by the query. This value will be 0 for the statements which return nothing.	indicates that statement has returned a ResultSet object and **FALSE** indicates that statement has returned an int value or returned nothing.
This method used with only select queries. Example :Select	This method used to execute non select queries . Example DML: INSERT UPDATE and DELETE DDL: CRETE,ALTER	This method used to execute for both select and non select queries used for SQL statements .

EXERCISE

Q1. What is JDBC? Explain various steps to create a database connection?

Q2. Create a Hotel Management Application using JDBC .

Q3. Explain the following:

a) Components of JDBC

b) Driver Types

c) DriverManager Class

□□□□□