Chapter-3
Java Database Programming

Relational Database Overview

- The relational model of data is based on the concept of relation (table).
- The relational model has established as the primary data model for commercial data processing application.
- The model is based on collections of tables.
- Users of the database can create tables, insert new tables or modify the existing tables using query language.
- A relational database is a database structured in relational model.
- A RDBMS is a collection of programs that can be used to create, maintain, modify and manipulate the relational database.
- We can use SQL queries to manipulate data in database. Some of the example of RDBMS are Oracle, MS-SQL Server, MS-Access etc.

Properties of RDBMS

1. It represents the data in the form two dimensional tables.
2. It shields the physical implementation from the user of database.
3. It gives information about content and structure of database.
4. It also supports null values.

A list of DML(Data manipulation language) commands are given below

1. SELECT commands

It is used to display all or selected records from a table.

Syntax: a. SELECT <field1>,<field2>…..<fieldN> FROM <table_name>
b. SELECT * FROM <table_name> // * represents all the field name defined for the particular table
c. SELECT * FROM <table_name> WHERE <Experssion>

E.g. SELECT id,name FROM tbl_student // display field id and name with all records
SELECT * FROM tbl_student // display all fields with all records
SELECT * FROM tbl_student where id<10 // display all the fields with only those records whose id is less than 10

2. INSERT INTO command

It is used to insert new record into a table.

Syntax: a. INSERT INTO <table_name> (field1,field2-----fieldN) VALUES (value1, value2----------valueN)
b. INSERT INTO <talb_name> values (value1, value2…….valueN)

E.g INSERT INTO tbl_student (id,name,roll,address) values (2,'Bhesh',5,'Pokhara')

3. UPDATE command

It is used to modify selected or all records from a table.

Syntax: a. UPDATE <table_name> SET field1=newvalue1, field2=newvalue2……fieldN=newvalueN
b. UPDATE <table_name> SET field1=newvalue2, field2=newvalue2……fielnN=newvaluN WHERE <Expression>

E.g. UPDATE tbl_student SET roll=5
UPDATE tbl_student SET roll=5 WHERE name='Bhesh'

4. DELETE command

It is used to delete all or selected records from a table.

Syntax: a. DELETE FROM <table_name> // deletes all records from table
b. DELETE FROM <table_name> WHERE <Expression>

E.g. DELETE FROM tbl_student
DELETER from tbl_student where id>10 // delete all records whose id value is greater than 10

A list of DDL(Data Defination language) commands are given below

1. CREATE DATABASE command

It is used to create a new database.

Syntax: CREATE DATABASE <database_name>
Eg. CREATE DATABASE db_MAC

2. CREATE TABLE command

It is used to create a table.

Syntax: CREATE TABLE <table_name>
(
Field1 datatype1,
Field2 datatype2,
.
.
.
FieldN datatypeN
)

E.g. CREATE TABLE student (id int, name char(20), roll int)

3. DROP DATABASE command

It is used to destroy the complete database structure.

Syntax: DROP DATABASE <database_name>

E.g. DROP DATABASE db_EEMC

4. DROP TABLE command

It is used to destroy the complete structure of table.

Syntax: DROP TABLE <table_name>

E.g. DROP TABLE student

5. Alter command

- It is used to change the structure of database object like table, stored procedure etc.

  i. add new column

  Syntax: ALTER TABLE <table_name> ADD <Newfield1 datatype1>,<Newfield2 datatype2>..<NewfieldN datatypeN>

  E.g. ALTER TABLE student ADD maths int, section char(20)

  ii. delete existing column

  Syntax: ALTER TABLE <table_name> DROP COLUMN <column_name1>,<column_name2>……<column_nameN>

  E.g. ALTER TABLE student DROP COLUMN maths,section

  iii. Changing the datatype of existing column

  Syntax: ALTER TABLE <table_name> ALTER COLUMN <column_name new datatype>

  E.g. ALTER TABLE tbl_student ALTER COLUMN ram nvarchar(50)

JDBC (Java Database Connectivity) API

- JDBC java database connectivity (JDBC) is an API (Application Programming Interface) for java that allows the Java programmer to access the Database.
- The JDBC API consists of a numbers of classes and interfaces, written in java programming language, they provide a numbers of methods for updating and querying a data in a database.
- It is a **relational database** oriented driver.
- It makes a bridge between java application and database. Java application utilizes the features of database using JDBC.
- JDBC provides the same capabilities as ODBC, allowing Java programs to contain database-independent code.
- The JDBC library includes APIs for each of the tasks commonly associated with database usage:
  - Making a connection to a database
  - Creating SQL or MySQL statements
  - Executing that SQL or MySQL queries in the database
  - Viewing & Modifying the resulting records

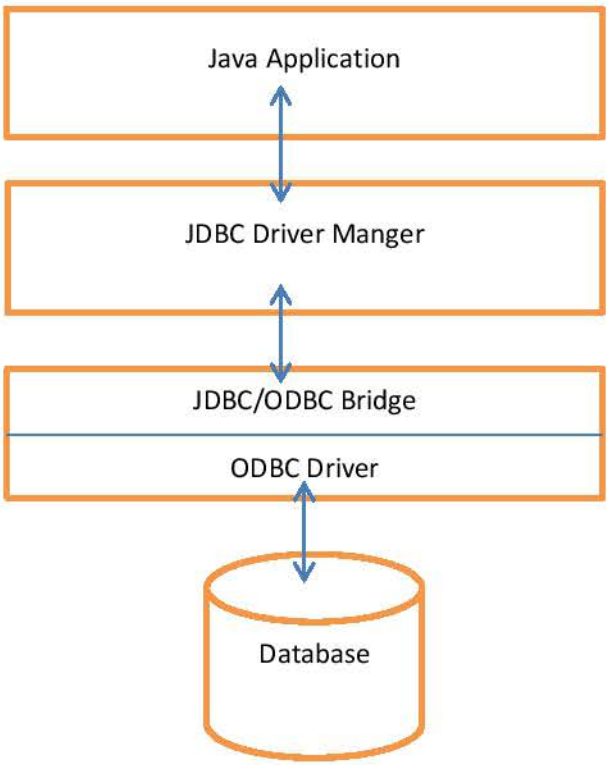


Fig: JDBC to Database communication path

Components of JDBC

The JDBC API provides the following interfaces and classes:

1. DriverManager: It manages interaction between the user-interface and the database driver used. This class manages a list of database drivers. Driver manager supports multiple drivers connecting to different database system. The first driver that recognizes a certain subprotocol under JDBC will be used to establish a database Connection.

2. JDBC Driver: This interface handles the communications with the database server. You will interact directly with Driver objects very rarely. Instead, you use DriverManager objects, which manages objects of this type.
JDBC driver are used to communicate with specific DBMS. The JDBC driver converts the queries to a form that a particular DBMS can understand and also retrieves the result of SQL queries, and convert it into the form the JDBC API classes and objects understand so that it can be used by the applications.

3. Connection : This interface with all methods for contacting a database. The connection object represents communication context, i.e., all communication with database is through connection object only.

4. Statement : You use objects created from this interface to submit the SQL statements to the database. Some derived interfaces accept parameters in addition to executing stored procedures.

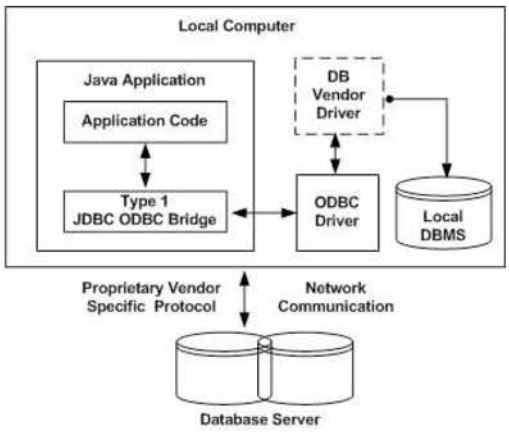
5. ResultSet: These objects hold data retrieved from a database after you execute an SQL query using Statement objects. It acts as an iterator to allow you to move through its data.

6. SQLException: This class handles any errors that occur in a database application.

JDBC Driver Types

JDBC driver implementations vary because of the wide variety of operating systems and hardware platforms in which Java operates. Sun has divided the implementation types into four categories, Types 1, 2, 3, and 4, which are explained below:

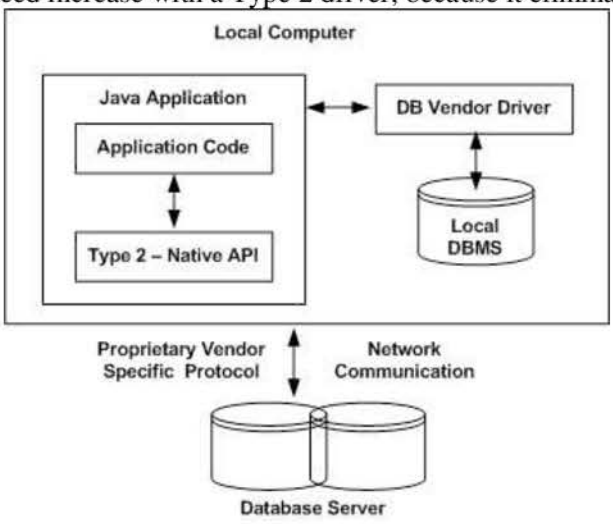
1. Type 1: JDBC-ODBC Bridge Driver (Bridge Driver)
   In a Type 1 driver, a JDBC bridge is used to access ODBC drivers installed on each client machine. Using ODBC requires configuring on your system a Data Source Name (DSN) that represents the target database.
   When Java first came out, this was a useful driver because most databases only supported ODBC access but now this type of driver is recommended only for experimental use or when no other alternative is available.



2. Type 2 : Native-API, Part Java Driver

   In a Type 2 driver, JDBC API calls are converted into native C/C++ API calls which are unique to the database. These drivers typically provided by the database vendors and used in the same manner as the JDBC-ODBC Bridge, the vendor-specific driver must be installed on each client machine.

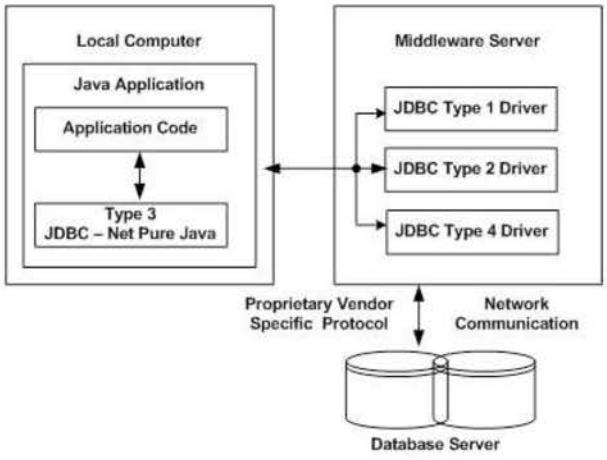
   If we change the Database we have to change the native API as it is specific to a database and they are mostly obsolete now but you may realize some speed increase with a Type 2 driver, because it eliminates ODBC's overhead.



3. Type 3: JDBC-Net pure Java Driver:

   In a Type 3 driver, a three-tier approach is used to accessing databases. The JDBC clients use standard network sockets to communicate with a middleware application server. The socket information is then translated by the middleware application server into the call format required by the DBMS, and forwarded to the database server.

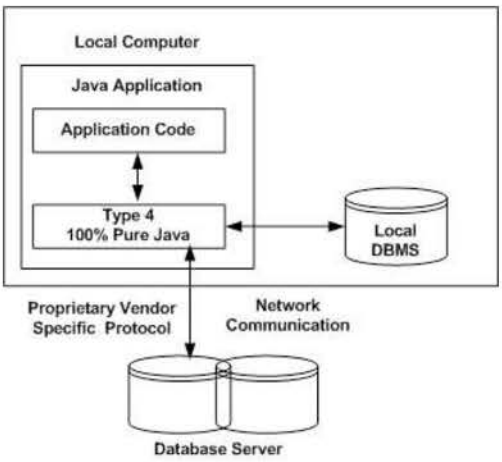
   This kind of driver is extremely flexible, since it requires no code installed on the client and a **single driver can actually provide access to multiple databases.**



4. Type 4: 100% pure Java:

   In a Type 4 driver, a pure Java-based driver that communicates directly with vendor's database through socket connection. This is the highest performance driver available for the database and is usually provided by the vendor itself.

   This kind of driver is extremely flexible; you don't need to install special software on the client or server. Further, these drivers can be downloaded dynamically.



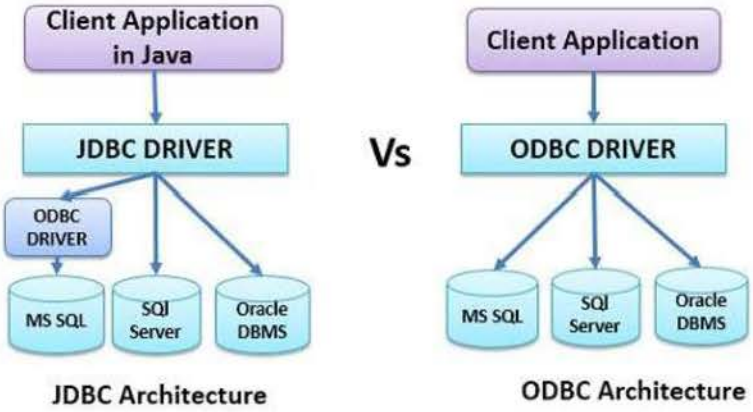
Which Driver should be used?

1. If you are accessing one type of database, such as Oracle, Sybase, or IBM, the preferred driver type is 4.
2. If your Java application is accessing multiple types of databases at the same time, type 3 is the preferred driver.
3. Type 2 drivers are useful in situations where a type 3 or type 4 driver is not available yet for your database.
4. The type 1 driver is not considered a deployment-level driver and is typically used for development and testing purposes only.

Introduction to ODBC (Open Database Connectivity)

Open Database Connectivity, a standard database access method developed by the SQL Access group. The goal of ODBC is to make it possible to access **any data from any application**, regardless of which database management system (DBMS) is handling the data. ODBC manages this by inserting a middle layer, called a database driver, between an application and the DBMS. The purpose of this layer is to translate the application's data queries into commands that the DBMS understands. For this to work, both the application and the DBMS must be ODBC- compliant -- that is, the application must be capable of issuing ODBC commands and the DBMS must be capable of responding to them.

JDBC VS ODBC



ODBC	JDBC
1) ODBC Stands for Open Database Connectivity	1) JDBC Stands for Java Database Connectivity
2) Introduced by Microsoft.	2) Introduced by Sun Micro Systems.
3) We can Use ODBC for any Languages like C, C++, Java, Etc.	3) We can Use JDBC only for Java Language.
4) We can use ODBC only for Windows Platforms.	4) We can use JDBC for any Platform.
5) Mostly ODBC Drivers are developed in Native Languages like C OR C++.	5) Mostly JDBC Drivers are developed in Java.
6) For Java Applications, it is not recommended to use ODBC because Performance will be Down due to Internal Conversions and Application will become Platform Dependent.	6) For Java Applications, it is highly recommended to use JDBC because there is no Performance Problems and Platform Dependency Problems.

Installing ODBC Driver
We will see it in lab.

Connecting Database using JDBC ODBC Driver

Following steps are used to create the java jdbc connection with the database

1. import java.sql.*;

2. Loading a database driver

In this step load the driver class by calling Class.forName() with the Driver class name as an argument. Once loaded, the Driver class creates an instance of itself. A client can connect to Database Server through JDBC Driver. The return type of the Class.forName (String ClassName) method is "Class" and resides in the java.lang package. To load the JDBC-ODBC driver, we use the following command.

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

2. Creating a jdbc Connection

The JDBC DriverManager class defines objects which can connect Java applications to a JDBC driver. DriverManager is considered the backbone of JDBC architecture. DriverManager class manages the JDBC drivers that are installed on the system. Its getConnection() method is used to establish a connection to a database. It uses a username, password, and a jdbc url to establish a connection to the database and returns a connection object.

Connection con=DriverManager.getConnection(url,"loginName","Password")

JDBC URL Syntax: jdbc : <subprotocol>:data source name
Eg: jdbc:odbc:ram Where odbc is a subprotocol and ram is the data source name,
Eg. Connection cn = DriverManager.getConnection("jdbc:odbc:ram","",""); Where, username and password field are empty. That is we don't use username and password in the database.

4. Creating a jdbc Statement object

Once a connection is obtained we can interact with the database. Connection interface defines methods for interacting with the database through the established connection. To execute SQL statements, we need to instantiate a Statement object from our connection object by using the createStatement() method.

Statement statement = con.createStatement();

A statement object is used to send and execute SQL statements to a database. Statement executes simple sql queries without parameters.

Note: We can also use PreparedStatement object and CallableStatement object for sending sql statements to database.

5. Executing a SQL statement with the Statement object, and returning a jdbc resultSet

Statement interface defines methods that are used to interact with database via the execution of SQL statements. The Statement class has three methods for executing statements: executeQuery(), executeUpdate(), and execute(). For a SELECT statement, the method to use is executeQuery . For INSERT, UPDATE, DELETE, CREATE, DROP statements, the method to use is executeUpdate(). execute() executes an SQL statement that is written as stored procedure.

ResultSet

- **ResultSet** provides access to a table of data generated by executing a select Statement.
- The table rows are retrieved in sequence.
- A ResultSet maintains a cursor pointing to its current row of data. **The next()** method is used to successively step through the rows of the tabular results.

```
Statement smt=Con.createStatement();
ResultSet rs=smt.executeQuery("Select * from tbl_Student");
```

- Some of the methods associated with ResultSet are
  1. getString(int column_index) or getString(String column_name)
  2. getInt(int column_index) or getInt(String column_name)
  3. getLong(int column_index) or getLong(String column_name)
  4. getDouble(int column_index) or getDouble(String column_name)
  5. getBoolean(int column_index) or getBoolean(String column_name)

Programming Example

1. **Program that insert data in the tbl_student table having fields id,name,address and section through java (Assume DSN as Bhesh)**

```
import java.sql.*;
class Insert
{
	public static void main(String arg[])
	{
		try
		{
			Class.forName("sun.jdbc.odbc.JdbcOdbcDriver"); // Load Driver
			Connection con=DriverManager.getConnection("jdbc:odbc:Bhesh","","");
			String str = "insert into tbl_student values(1,'ram','Kathmandu','VI')";
			Statement stat = con.createStatement();
			stat.executeUpdate(str);
			System.out.println("Record Inserted Successfully");

			con.close();
		}
		catch(ClassNotFoundException e)
		{
			System.out.println("Cannot insert into the table"+e);
		}
		catch(SQLException e)
		{
			System.out.println(e);
		}
	}

}
```

Or

```
import java.sql.*;
class Insert
{
	public static void main(String arg[]) throws Exception
	{
			Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
			Connection con=DriverManager.getConnection("jdbc:odbc:Bhesh","sa","1111");
			String str = "insert into tbl_student values(1,'ram','Kathmandu','VI')";
			Statement stat = con.createStatement();
			stat.executeUpdate(str);
			System.out.println("Record Inserted Successfully");
			con.close();
	}

}
```

2. **Java program that is used to insert multiple record in the above table.**

```
import java.sql.*;
public class InsertMultiple
{
        public static void main (String[] args) throws Exception
        {
                Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
                String url = "jdbc:odbc:Bhesh";   // Bhesh is Datasource name
                Connection conn = DriverManager.getConnection(url,"","");
                Statement st = conn.createStatement();
                st.executeUpdate("INSERT INTO tbl_student VALUES (1005, 'Ram','Kathmandu', 'V')");
                st.executeUpdate("INSERT INTO tbl_student VALUES (1007,'Shyam','Pokhara', 'Vi')");
                st.executeUpdate("INSERT INTO tbl_student VALUES (1008, 'Hari','Nepalgung','I')");
                st.executeUpdate("INSERT INTO tbl_student VALUES (1009,'Krishna','Dhangadhi','II')");


                /*
                //String sql=" INSERT INTO tbl_student VALUES (1005, 'Ram','Kathmandu', 'V') INSERT INTO
                tbl_student VALUES (1006, 'Ram','Kathmandu', 'V')  INSERT INTO tbl_student VALUES (1007,
                'Ram','Kathmandu', 'V')";
                st.executeUpdate(sql);
                */
                System.out.println("Records Inserted Successfully");
                conn.close();
        }
}
```

3. **Program that is used to display all the records in the above table of SQL database**

```
import java.sql.*;
 class Query1
 {
         public static void main (String[] args) throws Exception
         {
                 Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
                 String url = "jdbc:odbc:Bhesh";
                 Connection conn = DriverManager.getConnection(url,"","");
                 Statement stmt = conn.createStatement();
                 ResultSet rs;
                 rs = stmt.executeQuery("SELECT * FROM tbl_student");
                 while ( rs.next() )
                 {
                         int id=rs.getInt("id");
                         String Name = rs.getString("name");
                         String address=rs.getString("address");
                         String pn=rs.getString("section");
                         System.out.println(id+"\t"+Name+"\t"+address+"\t"+pn);
                 }
                 conn.close();
         }
}
```

4. **A database consists of table named tbl_login having fields id, username and password. Write a java program to display 'you are valid user' if user input valid username and password.**

```
import java.sql.*;
import java.util.Scanner;
class LoginValidation
{
	public static void main (String[] args) throws Exception
	{
		Scanner sc=new Scanner(System.in);
		System.out.println("Enter the username");
		String uname=sc.nextLine();
		System.out.println("Enter the password");
		String pwd=sc.nextLine();

		Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
		String url = "jdbc:odbc:Bhesh";
		Connection conn = DriverManager.getConnection(url,"","");
		Statement stmt = conn.createStatement();
		ResultSet rs;
		rs = stmt.executeQuery("SELECT * FROM tbl_login where username='"+uname+"' and
		password='"+pwd+"' ");
		if(rs.next()==false)
		{
			System.out.println("You are not valid user");
		}
		else
		{
			System.out.println("You are valid user");
		}
		conn.close();

	}
}
```

5. **A database consists of table named tbl_student having fields id, name, address and section. Write a java program to modify the record of 'ram' whose id is '1001' to new name 'Bheeshma'. Finally display the name after modification.**

```
import java.sql.*;
class UpdateQuery1
{
	public static void main (String[] args) throws Exception
	{
		Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
		String url = "jdbc:odbc:Bhesh";
		Connection conn = DriverManager.getConnection(url,"","");
		Statement stmt = conn.createStatement();
		ResultSet rs;
		stmt.executeUpdate("UPDATE tbl_student SET name = 'Bheeshma' WHERE id = 1001");
		rs = stmt.executeQuery("SELECT * from tbl_student where id = 1001");
		while ( rs.next() )
		{
		String Name = rs.getString(2); // 2 is column index of 'name' field and index starts from 1
		//or String Name=rs.getString("name");
		System.out.println(Name);
```

```
        }
        conn.close();
    }
}
```

6. **A database consists of table named tbl_student having fields id, name, address and section. Write a java program to delete the record a record whose id is '1001'.**

```
import java.sql.*;
class DeleteQuery
{
    public static void main (String[] args) throws Exception
    {
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        String url = "jdbc:odbc:Bhesh";
        Connection conn = DriverManager.getConnection(url,"","");
        Statement stmt = conn.createStatement();
        stmt.executeUpdate("DELETE from tbl_student WHERE id = 1001");
        System.out.println("Record Deleted Successfully");
        conn.close();
    }
}
```

7. **Write a java program to count the number of records in tbl_student.**

```
import java.sql.*;
class CountRecord
{
    public static void main (String[] args) throws Exception
    {
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        String url = "jdbc:odbc:Bhesh";
        Connection conn = DriverManager.getConnection(url,"","");
        Statement stmt = conn.createStatement();
        ResultSet rs;
        rs = stmt.executeQuery("SELECT * FROM tbl_student");
        int count=0;
        while ( rs.next() )
        {
            count++;
        }
        System.out.println("Total number of records is"+count);
        conn.close();
    }
}
```

8. **Write a java program to create a table having fields id, name, address and section.**

```
import java.sql.*;
class CreateTableQuery
{
    public static void main (String[] args) throws Exception
    {
        String url = "jdbc:odbc:Bhesh";
        Connection conn = DriverManager.getConnection(url,"sa","1111");
        Statement stmt = conn.createStatement();
        String sql = "create table tbl_student(id int Primary Key, name varchar(255), address varchar(255),
        section varchar(50))";
        stmt.executeUpdate(sql);
```

```
                System.out.println("Table is created in the given database");
                conn.close();

            }
    }
```

9. Write a GUI program to show entire record of table named tbl_std with fields id, name and address in JTable.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.table.*;
import java.sql.*;

public class Demo extends JFrame
{
        JScrollPane scroll;
        JTable table;
        DefaultTableModel model;
        JScrollPane scroll;
        public Demo()
        {
                try
                {
                setSize(800,400);
                model=new DefaultTableModel();
                table=new JTable(model);
                model.addColumn("ID");
                model.addColumn("NAme");
                model.addColumn("Address");
                scroll=new JScrollPane(table);
                Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
                Connection con=DriverManager.getConnection("jdbc:odbc:bhesh","","");
                String sql="select * from tbl_std";
                Statement st=con.createStatement();
                ResultSet rs=st.executeQuery(sql);
                while(rs.next())
                {
                        Object []data=new Object[3];
                        data[0]=rs.getInt(1);
                        data[1]=rs.getString(2);
                        data[2]=rs.getString(3);
                        model.addRow(data);

                        //model.addRow(new Object[]{rs.getInt(1),rs.getString(2),rs.getString(3)});
                }
                add(scroll);
                setVisible(true);
                con.close();
                }
                catch(Exception ex){}

        }
        public static void main(String[] args)
        {
                new Demo();
```

```
        }
}
```

10. Write a java program to show entire record of table named tbl_std with fields id, name and address under button click event.

Note:

- The DefaultTableModel stores the data for the JTable. The advantage of using DefaultTableModel is we don't have to code the methods like add, insert or delete rows and columns. They already exist to change the data held in the model. This makes it a quick and easy table model to implement.
- The constuuctor is

  DefaultTableModel defTableModel = DefaultTableModel();
- Some of the methods are
  1. addColumn(Object columnName) : Adds a column to the model.
  2. addRow(Object[] rowData) : Adds a row to the end of the model.
  3. getColumnCount() : Returns the number of columns in this data table.
  4. getColumnName(int column) : Returns the column name.
  5. getRowCount() :Returns the number of rows in this data table.
  6. getValueAt(int row, int column) : Returns an attribute value for the cell at row and column.
  7. setValueAt(Object aValue, int row, int column): Sets the object value for the cell at column and row.
  8. insertRow(int row, Object[] rowData): Inserts a row at row in the model.
  9. removeRow(int row): Removes the row at row from the model.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.table.*;
import java.sql.*;
public class Demo extends JFrame implements ActionListener
{
    JScrollPane scroll;
    JButton btn;
    DefaultTableModel model;
    JTable table;
    public Demo()
    {
            setLayout(new FlowLayout());
            setSize(800,400);
            btn=new JButton("Show");
            btn.addActionListener(this);
            model=new DefaultTableModel();
            table=new JTable(model);
            JScrollPane scroll=new JScrollPane(table);
            add(btn);
            add(scroll);
            setVisible(true);
    }
    public static void main(String[] args) throws Exception
    {
        new Demo();
    }
    public void actionPerformed(ActionEvent e)
    {
```

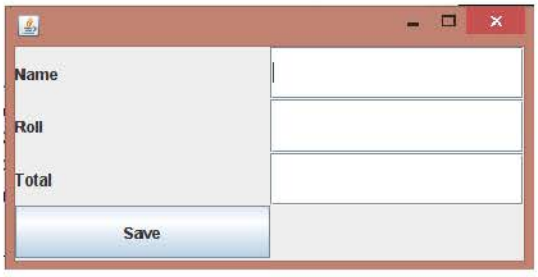
```
            try
            {
                    model.addColumn("ID");
                    model.addColumn("Name");
                    model.addColumn("Address");
                    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
                    Connection con=DriverManager.getConnection("jdbc:odbc:bhesh","","");
                    String sql="select * from tbl_std";
                    Statement st=con.createStatement();
                    ResultSet rs=st.executeQuery(sql);
                    while(rs.next())
                    {
                            Object []data=new Object[3];
                            data[0]=rs.getInt(1);
                            data[1]=rs.getString(2);
                            data[2]=rs.getString(3);
                            model.addRow(data);
                    }
                    con.close();
            }
            catch(Exception ex)
            {
            }
    }
}
```

1. **Make a form that accepts roll number, name and total marks of the student and save those values in the table tbl_mark when user clicks in the Save button.(Assume, DSN=Bhesh)**

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.sql.*;
public class InsertGui extends JFrame implements ActionListener
{
        JButton btn_save;
        JLabel lbl_name;
        JLabel lbl_roll;
        JLabel lbl_total;
        JTextField txt_name;
        JTextField txt_roll;
        JTextField txt_total;
        public InsertGui()
        {
                setSize(400,200);
                setLayout(new GridLayout(4,2));
                lbl_name=new JLabel("Name");
                lbl_roll=new JLabel("Roll");
                lbl_total=new JLabel("Total");
                txt_name=new JTextField(10);
                txt_roll=new JTextField(10);
                txt_total=new JTextField(10);
                btn_save=new JButton("Save");
                btn_save.addActionListener(this);
                add(lbl_name);
                add(txt_name);
                add(lbl_roll);
```

```
                add(txt_roll);
                add(lbl_total);
                add(txt_total);
                add(btn_save);
                show();
        }
        public static void main(String [] args)
        {
                new InsertGui();
        }
        public void actionPerformed(ActionEvent e)
        {
                if(e.getSource()==btn_save)
                {
                        try
                        {
                                String nm=txt_name.getText();
                                int roll=Integer.parseInt(txt_roll.getText());
                                double total=Double.parseDouble(txt_total.getText());
                                Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
                                 String url = "jdbc:odbc:Bhesh";
                                 Connection conn = DriverManager.getConnection(url,"","");
                                 Statement st = conn.createStatement();
                                 st.executeUpdate("INSERT INTO tbl_mark " + "VALUES
                                ('"+nm+"','"+roll+"','"+total+"')");
                                JOptionPane.showMessageDialog(this,"Successfully inserted one  record");
                                 conn.close();
                        }
                        catch(Exception er)
                        {
                                JOptionPane.showMessageDialog(this,"Failed to insert a record because of error"+er);
                        }
                }

        }
}
```

Output:

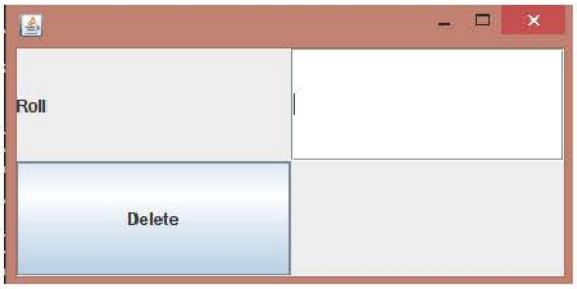


2. Make a form that accepts roll number of the student and delete those records in the table tbl_mark associated with roll number when user clicks in the Delete button.(Assume, DSN=Bhesh)

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.sql.*;
public class DeleteGui extends JFrame implements ActionListener
{
        JButton btn_Delete;
        JLabel lbl_roll;
```

```
JTextField txt_roll;
public DeleteGui()
{
        setSize(400,200);
        setLayout(new GridLayout(2,2));
        lbl_roll=new JLabel("Roll");
        txt_roll=new JTextField(10);
        btn_Delete=new JButton("Delete");
        btn_Delete.addActionListener(this);
        add(lbl_roll);
        add(txt_roll);
        add(btn_Delete);
        show();
}
public static void main(String [] args)
{
        new DeleteGui();
}
public void actionPerformed(ActionEvent e) //override method cant throw exception
{
        if(e.getSource()==btn_Delete)
        {
                try
                {
                        int rll=Integer.parseInt(txt_roll.getText());
                        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
                         String url = "jdbc:odbc:Bhesh";
                         Connection conn = DriverManager.getConnection(url,"","");
                         Statement st = conn.createStatement();
                         st.executeUpdate("DELETE from tbl_mark where roll='"+rll+"'");
                        JOptionPane.showMessageDialog(null,"Successfully Deleted");
                         conn.close();
                }
                catch(Exception er)
                {
                        JOptionPane.showMessageDialog(null,"Failed to Delete because of error"+er);
                }
        }

}
}
```

Output:



3. **Make a form that displays name, roll and total marks sequentially from the table tbl_mark when user clicks in the Next button.(Assume, DSN=Bhesh)**

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.sql.*;
```

```
public class ViewGui extends JFrame implements ActionListener
{
	ResultSet rs;
	Statement stmt;
	Connection conn;
	JButton btn_next;

	JLabel lbl_name;
	JLabel lbl_roll;
	JLabel lbl_total;
	JTextField txt_name;
	JTextField txt_roll;
	JTextField txt_total;
	public ViewGui()
	{
		try
		{
			Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
			String url = "jdbc:odbc:Bhesh";
			conn = DriverManager.getConnection(url,"sa","1111");
			stmt = conn.createStatement();
			rs = stmt.executeQuery("SELECT * FROM tbl_student");
		}


		catch(Exception y)
		{
			System.out.println(y.getMessage());
		}
		setSize(400,200);
		setLayout(new GridLayout(4,2));
		lbl_name=new JLabel("Name");
		lbl_roll=new JLabel("Roll");
		lbl_total=new JLabel("Total");
		txt_name=new JTextField(10);
		txt_roll=new JTextField(10);
		txt_total=new JTextField(10);
		btn_next=new JButton("Next");
		btn_next.addActionListener(this);
		add(lbl_name);
		add(txt_name);
		add(lbl_roll);
		add(txt_roll);
		add(lbl_total);
		add(txt_total);
		add(btn_next);
		show();
	}
	public static void main(String [] args)
	{
		new ViewGui();
	}
	public void actionPerformed(ActionEvent e) //override method cant throw exception
	{
		if(e.getSource()==btn_next)
		{
```

```
                try
                {
                        if(rs.next()==true)
                        {
                                int id=rs.getInt("id");
                                String Name = rs.getString("name");
                                String address=rs.getString("address");
                                String pn=rs.getString("phno");
                                txt_name.setText(Name);
                                txt_roll.setText(address);
                                txt_total.setText(pn);
                        }
                        else
                        {
                                JOptionPane.showMessageDialog(null,"End of file reached");
                        }
                }
                catch(Exception ee)
                {
                        JOptionPane.showMessageDialog(null,"Failed to retrieve data because of
                        error "+ee);
                }

        }

    }
}
```

Output:

Prepared Statement

The prepared statement is used to execute precompiled SQL statements thereby providing extremely efficient for repeated execution.

Features

- Easy to insert parameters using positional parameters(?) into the SQL statement.
- Easy to reuse the Prepared Statement with new parameter values.
- May increase performance of executed statements.
- Increase security (Prevents Database Injection Attack)
- Enables easier batch updates.

1. **JAVA program to insert a record and display the records using PreparedStatement.**

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.util.Scanner;
public class Prepared
```

```
{
	public Prepared()
	{
		try
		{
			Scanner sc=new Scanner(System.in);
			Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
			Connection cn=DriverManager.getConnection("jdbc:odbc:Bhesh","","");
			System.out.println("Enter the id");
			int id=sc.nextInt();
			sc.nextLine();//flushing
			System.out.println("Enter name");
			String name=sc.nextLine();
			System.out.println("Enter address");
			String add=sc.nextLine();
			System.out.println("Enter phonenumber");
			String ph=sc.nextLine();
			String sql1="insert into tbl_student values (?,?,?,?)";
			PreparedStatement smt1=cn.prepareStatement(sql1);
			smt1.setInt(1,id);
			smt1.setString(2,name);
			smt1.setString(3,add);
			smt1.setString(4,ph);

			int rowupdated=smt1.executeUpdate();
			System.out.println("REcords inserted "+rowupdated);

			//note we can again reuse the above statement to insert other records just by updating
			the parameters.

			String sql2="select * from tbl_student";
			PreparedStatement smt2=cn.prepareStatement(sql2);
			ResultSet rs=smt2.executeQuery();
			while(rs.next())
			{
				int i=rs.getInt(1);
				String n=rs.getString(2);
				String a=rs.getString(3);
				String p=rs.getString(4);
				System.out.println(i +" " + n + " " + a + " " +p);
			}
		}
		catch (Exception e)
		{
			System.out.println("Error Occured"+e);
		}
	}

	public static void main(String[] args)
	{
		new Prepared();
	}
}
```

2. **A database consists of table named tbl_std having fields id, name and address. Write a java program to modify the record of 'ram' whose id is '1' to new name 'Hari'. Finally display the name after modification. Use the concept of prepared statement.**

```
import java.sql.*;
class UpdateQuery1
{
	public static void main (String[] args) throws Exception
	{
		Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
		String url = "jdbc:odbc:bhesh";
		Connection conn = DriverManager.getConnection(url,"","");

		String sql1="update tbl_student set Name=? where id=?";
		PreparedStatement stmt1 = conn.prepareStatement(sql1);
		stmt1.setString(1,"hari");
		stmt1.setInt(2,1);
		stmt1.executeUpdate();

		String sql2="SELECT * from tbl_student where id = ?";
		PreparedStatement stmt2 = conn.prepareStatement(sql2);
		stmt2.setInt(1,1);

		ResultSet rs;
		rs = stmt2.executeQuery();
		while ( rs.next() )
		{
			String Name = rs.getString(2);
			System.out.println(Name);
		}
		conn.close();
	}
}
```

Metadata

Data about data is called metadata. Metadata describes the database or one of its part.

Scrollable and Updatable ResultSets

- Using Scrollable and Updatable ResultSet provided by JDBC 2 (JDK 1.2), we can move forward and backward through the result set and jump to any position in the result set.
- It also allows us to update the database by simply updating the result set entries.
- The common format of the createStatement() is given below.

  Statement stmt=<Connectin object>.createStatement(ResultSets types, Resultsets Concurrency);

ResultSet Types

1. ResultSet.TYPE_FORWARD_ONLY:
   - It is default ResutlSet which moves only from first row to last row via next() method.
2. ResultSet.TYPE_SCROLL_INSENSITIVE or ResultSet.TYPE_SCROLL_SENSITIVE
   - The ResultSet object has a cursor, which initially does not point to any record. Both of the above types are scrollable so that row cursor can move forward, backward or to an absolute or relative row using different methods such as
     1. first(): move the cursor to first row
     2. last(): move the cursor to last row
     3. previous(): move the cursor to previous row from current row
     4. next(): move the cursor to next row from current row
     5. absolute(rowNumber): move the cursor to specified row
     6. relative(rowNumber): Moves the cursor the given number of rows forward or backward, from where it is currently

pointing.

7. berforeFirst() : Moves the cursor just before the first row.
8. afterLast(): Moves the cursor just after the last row
9. moveToInsertRow(): Moves the cursor to a special row in the result set that can be used to insert a new row into the database.
10. moveToCurrentRow():Moves the cursor back to the current row if the cursor is currently at the insert row

- If the **ResultSet is Insensitive** then it is insensitive to changes i.e. the ResultSet doesn't reflect changes made by others to the database that occur after the resultset was created but if the ResultSet is made Sensitive then it is sensitive to changes. For example, if we read some table using **sensitive resultset** and suppose, from some other application, the table content has been modified, so in such case, this changes are reflected in our application automatically. But this won't be reflected in case of insensitive resultset.

- ResultSet has two concurrency
  1. ResultSet.CONCUR_READ_ONLY: it is default state of ResultSet in which it is read only. We can't update the underlying database using the ResultSet.
  2. ResultSet.CONCUR_UPDATABLE: It creates an updatable result set. The ResultSet object can be updated using the updateXXX() method and finally changes in the ResultSet is reflected in the underlying database.
- General syntax for creating updatable and scrollable result set is

```
Statement stmt = conn.createStatement( ResultSet.TYPE_SCROLL_INSENSITIVE,  ResultSet.CONCUR_UPDATABLE);
```

JAVA Program to demonstrate the scrollable ResultSet. Assume DSN: mydata, table name: tbl_std with field id, name and address.

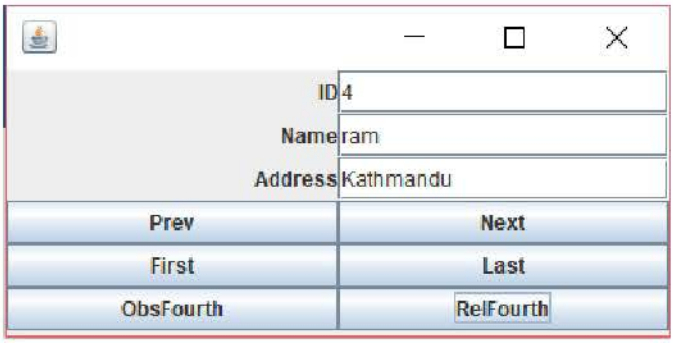
```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.sql.*;
public class Scrollable extends JFrame implements ActionListener
{
        ResultSet rs;
        Statement stmt;
        Connection conn;

        JButton btn_next;
        JButton btn_prev;
        JButton btn_first;
        JButton btn_last;
        JButton btn_Obsfourth;
        JButton btn_Relfourth;


        JLabel lbl_id;
        JLabel lbl_name;
        JLabel lbl_add;
        JTextField txt_id;
        JTextField txt_name;
        JTextField txt_add;

        public Scrollable()
        {
                try
                {
                        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
                        String url = "jdbc:odbc:bhesh";
```

```
                conn = DriverManager.getConnection(url,"","");


                stmt = conn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
                ResultSet.CONCUR_READ_ONLY);
                rs = stmt.executeQuery("SELECT * FROM tbl_student");
        }
        catch(Exception y)
        {
                 System.out.println("Error Occured"+y);
        }
        setSize(400,200);
        setLayout(new GridLayout(6,2));
        lbl_id=new JLabel("ID",JLabel.RIGHT);
        lbl_name=new JLabel("Name",JLabel.RIGHT);
        lbl_add=new JLabel("Address",JLabel.RIGHT);
        txt_id=new JTextField(10);
        txt_name=new JTextField(10);
        txt_add=new JTextField(10);
        btn_next=new JButton("Next");
        btn_next.addActionListener(this);
        btn_prev=new JButton("Prev");
        btn_prev.addActionListener(this);
        btn_first=new JButton("First");
        btn_first.addActionListener(this);
        btn_last=new JButton("Last");
        btn_last.addActionListener(this);
        btn_Obsfourth=new JButton("ObsFourth");
        btn_Obsfourth.addActionListener(this);
        btn_Relfourth=new JButton("RelFourth");
        btn_Relfourth.addActionListener(this);

        add(lbl_id);
        add(txt_id);
        add(lbl_name);
        add(txt_name);
        add(lbl_add);
        add(txt_add);
        add(btn_prev);
        add(btn_next);
        add(btn_first);
        add(btn_last);
        add(btn_Obsfourth);
        add(btn_Relfourth);
        show();
}
public static void main(String [] args)
{
      new Scrollable();
}

public void fillControl() throws Exception
{
        int id=rs.getInt(1);
        String Name = rs.getString(2);
        String address=rs.getString(3);
```

```
            txt_id.setText(Integer.toString(id));
            txt_name.setText(Name);
            txt_add.setText(address);
}

public void actionPerformed(ActionEvent e) //NOTE: override method cant throw exception
{
            try
            {
                        if(e.getSource()==btn_next)
                        {
                                    if(rs.next()==true)
                                    {
                                        fillControl();
                                    }
                                    else
                                    {
                                       JOptionPane.showMessageDialog(null,"End of file reached");
                                    }

                        }
                        else if(e.getSource()==btn_prev)
                        {
                                    if(rs.previous()==true)
                                    {
                                        fillControl();
                                    }
                                    else
                                    {
                                        JOptionPane.showMessageDialog(null,"End of file reached");
                                    }
                        }




                        else if(e.getSource()==btn_first)
                        {
                                    rs.first();
                                    fillControl();
                        }
                        else if(e.getSource()==btn_last)
                        {
                                    rs.last();
                                    fillControl();
                        }
                        else if(e.getSource()==btn_Obsfourth)
                        {
                                    rs.absolute(4);
                                    fillControl();
                        }
                        else if(e.getSource()==btn_Relfourth)
                        {
                                    rs.relative(4);
                                    fillControl();
                        }
```

```
                }

                catch(Exception ea)
                {
                    System.out.println("Error"+ea);
                }

            }
        }
```

Output:



Following table is the snapshot for tbl_std. Write java program for performing the following operations using scrollable and updatable Resultset.

	id	Name	address
	1	hari	PKR
	3	sunil	pokhara
▶	4	ram	Kathmandu
	5	dfdf	df
	6	jkj	j
*	NULL	NULL	NULL

1. Insert new record.
2. Delete Existing record based on id
3. Modify Existing record based on id
4. Display all the record
5. Exit

```
import java.sql.*;
import java.util.Scanner;
public class ResultsetUpdatable
{
    Connection conn;
    Statement stmt;
    ResultSet rs;
    public ResultsetUpdatable()
    {
            Scanner sc=new Scanner(System.in);
            try
            {
                Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
                 String url = "jdbc:odbc:Bhesh";
                conn = DriverManager.getConnection(url,"","");
                stmt =
                conn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_UPDATABLE);
                        rs=stmt.executeQuery("select * from tbl_student");
```

```
System.out.println("Choose 1-4");
System.out.println("1: insert new record");
System.out.println("2: Delete existing record based on id");
System.out.println("3: Modify existing record based on id");
System.out.println("4: Display all record");
System.out.println("5: exit");
int choice =sc.nextInt();

switch(choice)
{
case 1:
        System.out.println("Enter the id of student");
        int id=sc.nextInt();
        sc.nextLine();
        System.out.println("Enter the name of student");
        String name=sc.nextLine();
        System.out.println("Enter the address of the student");
        String add=sc.nextLine();
        rs.moveToInsertRow(); //insertRow() method can be invoked only if the cursor is pointing to
                                //current insert row
        rs.updateInt(1,id);
        rs.updateString(2,name);
        rs.updateString(3,add);
        rs.insertRow();
        System.out.println("Record inserted successfully");
break;




case 2:
        System.out.println("Enter the id of the student to be deleted");
        int idvalue=sc.nextInt();
        while(rs.next())
        {
                if(rs.getInt(1)==idvalue)
                {
                        rs.moveToCurrentRow();
                        rs.deleteRow();
                        break;
                }
        }
        System.out.println("Record Deleted Successfully");
break;

case 3:
        System.out.println("Enter the id of student whose record is to be updated");
        int x=sc.nextInt();
        sc.nextLine();
        System.out.println("Enter the name");
        String nm=sc.nextLine();
        System.out.println("Enter address");
        String adr=sc.nextLine();
```

```
                        while(rs.next())
                        {
                                if(rs.getInt(1)==x)
                                {
                                        rs.moveToCurrentRow();
                                        rs.updateString(2,nm);
                                        rs.updateString(3,adr);
                                        rs.updateRow();
                                        System.out.println("Successfully updated");
                                        break;
                                }
                        }
                break;

                case 4:
                        rs=stmt.executeQuery("select * from tbl_student");
                        while(rs.next())
                        {
                                int idval=rs.getInt(1);
                                String nam=rs.getString(2);
                                String addr=rs.getString(4);
                                System.out.println(idval + " " + nam + "  " + addr);
                        }
                break;
                case 5:
                        conn.close();
                        System.exit(0);
                break;

                default:
                        main(null);
                break;

                }
        }
         catch(Exception y)
        {
                System.out.println("Error"+y.getMessage());// to update result set it must be exception handled
        }
    }
    public static void main(String [] args)
    {
            new ResultsetUpdatable();
    }
}
```

Output:

Callable Statement

- JDBC allow the use of Stored procedure by the CallableStatement class.
- The use of stored procedure makes our system more secure, consistent and manageable multi-tier system.
- A CallableStatement object is created by the prepareCall () method in the connection method. The prepareCall() method take a string as a parameter.
- When a batch of SQL statements needs to be executed more than once, we need to recreate SQL statement and submit them to the server. This leads to an increase in the overhead, as the server needs to compile and create the execution plan for these statements again. Therefore, if we need to execute a batch multiples times, we can save it within a stored procedure.
- **A stored procedure is a precompiled object stored in a database data dictionary.**

JAVA program to insert and display the records using the CallableStatement.

```
import java.sql.*;
import java.util.*;
public class Callable
{
	public Callable()
	{
		try
		{
			Scanner sc=new Scanner(System.in);
			System.out.println("Enter id");
			int id=sc.nextInt();
			sc.nextLine();//flushing
			System.out.println("Enter name");
			String nm=sc.nextLine();
			System.out.println("Enter address");
			String add=sc.nextLine();
			System.out.println("Enter phone number");
			String pn=sc.nextLine();
			Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
			String url = "jdbc:odbc:Bhesh";
			Connection conn = DriverManager.getConnection(url,"","");
			CallableStatement cs=conn.prepareCall("{call proc_insert(?,?,?,?)}");
			cs.setInt(1,id);
			cs.setString(2,nm);
			cs.setString(3,add);
			cs.setString(4,pn);
			cs.execute();
			System.out.println("Record Inserted Successfully");

			CallableStatement cs1=conn.prepareCall("{call proc_display}");
			ResultSet rs=cs1.executeQuery();
			while(rs.next())
			{
				int i=rs.getInt(1);
				String nam=rs.getString(2);
				String addr=rs.getString(3);
				String phn=rs.getString(4);
				System.out.println(i+" " + nam+ " " + addr + " " +phn);
			}
		}
		catch(Exception e)
		{
			System.out.println("Error occured"+e.getMessage());
```

```
            }
        }
        public static void main(String [] args)
        {
            new Callable();
        }
    }
```

write the stored procedure in sql

```
create procedure proc_insert
@id int,
@name varchar(50),
@address varchar(50),
@phno varchar(50)
as
begin
insert into tbl_student values(@id,@name,@address,@phno);
end

create procedure proc_display
as
begin
select * from tbl_student
end
```

SQL Escapes

- The escape syntax gives you the flexibility to use database specific features unavailable to us by using standard JDBC methods and properties.
- The general SQL escape syntax format is as follow:

{keyword 'parameters'}

Following are various escape keywords in JDBC:

1. d, t, ts Keywords:
   - They help identify date, time, and timestamp literals.
   - As we know, no two DBMSs represent time and date the same way.
   - This escape syntax tells the driver to render the date or time in the target database's format

     {d 'yyyy-mm-dd'}

     Where yyyy = year, mm = month; dd = date.
     Using this syntax {d '2009-09-03'} is March 9, 2009.

     Example:

```
//Create a Statement object
stmt = conn.createStatement();
String sql="INSERT INTO STUDENTS VALUES" + "(1001, 'Ram,'Thapa', {d '2001-12-16'})";
stmt.executeUpdate(sql);
```

2. escape Keyword
   - This keyword identifies the escape character used in LIKE clauses. Useful when using the SQL wildcard %, which matches zero or more characters.

     Example:

```
String sql = "SELECT symbol FROM MathSymbols WHERE symbol LIKE '\%' {escape '\'}";
stmt.execute(sql);
```

     Note: The above sql will returl all symbols %. Since % is itself a wild card character, using escape symbol \ will treat % as regular literal, not wild card character.
     If we use the backslash character (\) as the escape character, you also have to use two backslash characters in your Java String literal, because the backslash is also a Java escape character.

3. fn Keyword
   - This keyword represents scalar functions used in a DBMS.
   - For example, WE can use SQL function *length* to get the length of a string

```
Statement stmt = conn.createStatement ();
        stmt.executeUpdate("SELECT * FROM tbl_std WHERE  {fn length(Name')}>3 ");
```

the above statement will return all the records of student whose name length is greater than 3.

Example:

```
Statement stmt = conn.createStatement ();
stmt.executeUpdate("UPDATE emp SET ename = {fn CONCAT('Ram', 'Thapa')}");
```

Row Sets and Cached Row Sets

- Scrollable result sets and updatable result sets which are powerful and flexible, but they have a drawback: it needs to keep the database connection open during the entire user interaction.
- This behaviour may affect your application's performance in case there are many concurrent connections to the database.
- In this situation, we can use a **row set** which doesn't need to keep the database connection always open, and is leaner than a result set.
- A row set contains all data from a result set, but it can be disconnected from the database.
- A row set may make a connection with a database and keep the connection open during its life cycle, in which case it is called **connected row set.**
- A row set may also make connection with a database, get data from it, and then close the connection. Such a row set is called **disconnected row set.** We can make changes to data in a disconnected row set, and commit changes to the database later (the row set must re-establish the connection with the database).
- In JDBC, a row set is represented by the **RowSet interface** which is defined in the **javax.sql** package.
- The javax.sql.rowset provides and interface named **CachedRowSet** that stores data in memory so you can work on the data without keeping the connection open all the time. CachedRowSet object is a container for rows of data that caches its rows in memory, which makes it possible to operate (scroll and update) without keeping the database connection open all the time. A CachedRowSet object makes use of a connection to the database only briefly: while it is reading data to populate itself with rows, and again while it is committing changes to the underlying database. So the rest of the time, a CachedRowSet object is disconnected, even while its data is being modified. Hence it is **called disconnected row set.**
- Being disconnected, a CachedRowSet object is much leaner than a ResultSet object, making it easier to pass a CachedRowSet object to another component in an application.
- We can modify data in a CachedRowSet object, but the modifications are not immediately reflected in the database. You need to make an explicit request to accept accumulated changes (insert, update and delete rows). The CachedRowSet then reconnects to the database and issues SQL statements to commit the changes.

Consider database db_student with table named tbl_student(id, name, address). WAP to display entire record and insert a new record using CachedRowSet

```
import java.sql.*;
import javax.sql.rowset.*;
public class CachedRowSetExample
{
  public static void main(String args[]) throws Exception
{
    //Registering the Driver
     Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver");

     String url="jdbc:sqlserver://localhost:49170/db_Student";
```

```
        //Creating the RowSet object
        RowSetFactory factory = RowSetProvider.newFactory();
        CachedRowSet rowSet = factory.createCachedRowSet();

        //Setting the URL
        rowSet.setUrl(url);

        //Setting the user name
        rowSet.setUsername("sa");

        //Setting the password
        rowSet.setPassword("*****");

        //Setting the query/command
        rowSet.setCommand("select * from tbl_student");

        rowSet.execute();

        System.out.println("Contents of the row set");
            while(rowSet.next())
            {
                System.out.print("ID: "+rowSet.getInt("ID")+", ");
                System.out.print(" Name: "+rowSet.getString("Name")+", ");
                System.out.print("Address: "+rowSet.getString("Address")+", ");
                System.out.println("");
            }

        //Inserting data into RowSet object

        rowSet.moveToInsertRow();
        rowSet.updateInt(1, 6001);
        rowSet.updateString(2, "Ram Thapa");
        rowSet.updateString(3, "Pokhara");
        rowSet.insertRow();
        System.out.println("One Row Inserted Successfully");

    }
}
```

Assignment

1. **WAP to demonstrate reading and writing LOB (Large Object: Example, Image) in java.**
2. **Briefly explain Transactions in java with suitable programming example.**