Database setup

Initially the ‘department_db’ database is created using command prompt console after logging into the mysql system. In this case XAMP is used as the mysql server. Below figures shows the commands used to create the two tables ‘department’ and ‘dept_locations’ filled with provided data.

```
MariaDB [(none)]> CREATE DATABASE department_db;
Query OK, 1 row affected (0.001 sec)

MariaDB [(none)]> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| abc_uni            |
| department_db      |
| hospital_db        |
| information_schema |
| mysql              |
| performance_schema |
| phpmyadmin         |
| university         |
+--------------------+
8 rows in set (0.001 sec)

MariaDB [(none)]> USE department_db
Database changed
MariaDB [department_db]> CREATE TABLE department(
    -> Dname varchar(20) NOT NULL,
    -> Dnumber int(5) NOT NULL,
    -> Mgr_ssn int(50) NOT NULL,
    -> Mgr_start_date date NOT NULL,
    -> PRIMARY KEY(Dnumber));
Query OK, 0 rows affected (2.787 sec)

MariaDB [department_db]> CREATE TABLE dept_locations(
    -> Dnumber int(5) NOT NULL,
    -> Dlocation varchar(20) NOT NULL,
    -> PRIMARY KEY(Dnumber,Dlocation),
    -> FOREIGN KEY(Dnumber) REFERENCES department(Dnumber)
    -> );
Query OK, 0 rows affected (0.018 sec)
```

Figure 1 : *creating ‘department_db’ and intialzing the Two tabels with appropriate data types.*

```
MariaDB [department_db]> INSERT INTO department VALUES
    -> ('Research',5,333445555,'1998-05-22'),
    -> ('Administration',4,987654321,'1995-01-01'),
    -> ('Headquaters',1,888665555,'1981-06-19');
Query OK, 3 rows affected (0.005 sec)
Records: 3  Duplicates: 0  Warnings: 0

MariaDB [department_db]> INSERT INTO dept_locations VALUES
    -> (1,'Houston'),
    -> (4,'Stafford'),
    -> (5,'Bellaire'),
    -> (5,'Sugarland'),
    -> (5,'Houston');
Query OK, 5 rows affected (0.005 sec)
Records: 5  Duplicates: 0  Warnings: 0
```

Figure 2: *filling up the tables with given data.*

```
MariaDB [department_db]> SELECT * FROM department
    -> ;
+----------------+---------+-----------+----------------+
| Dname          | Dnumber | Mgr_ssn   | Mgr_start_date |
+----------------+---------+-----------+----------------+
| Headquaters    |       1 | 888665555 | 1981-06-19     |
| Administration |       4 | 987654321 | 1995-01-01     |
| Research       |       5 | 333445555 | 1998-05-22     |
+----------------+---------+-----------+----------------+
3 rows in set (0.000 sec)

MariaDB [department_db]> SELECT * FROM dept_locations;
+---------+-----------+
| Dnumber | Dlocation |
+---------+-----------+
|       1 | Houston   |
|       4 | Stafford  |
|       5 | Bellaire  |
|       5 | Houston   |
|       5 | Sugarland |
+---------+-----------+
5 rows in set (0.000 sec)
```

Figure 3: *Output of the two table after the creation.*

Data types used

Attribute	Data types (used in SQL)	Data types (used in Java)
Dname	varchar	String
Dnumber	integer	Integer
Mgr_ssn	integer	Integer
Mgr_start_date	date	String / java.sql.Date
Dlocation	varchar	String

In this assignment I build a Graphical User Interface (GUI) to intract with the database easily. User typed data is obtained using text field which parses the data as String type. And for Int data type the relavent data from the text field is converted to the Integer type in Java. For 'Mgr_start_date' I used String data type and the execption handling for mismatches is done through the code.

GUI implementation of the system

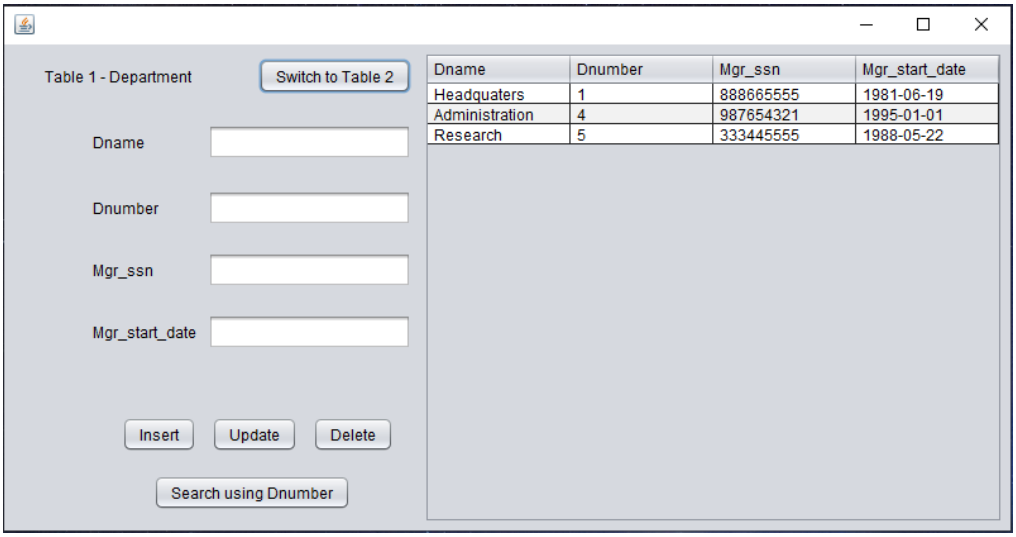

Figure 4: *GUI implementation of 'department' table.*

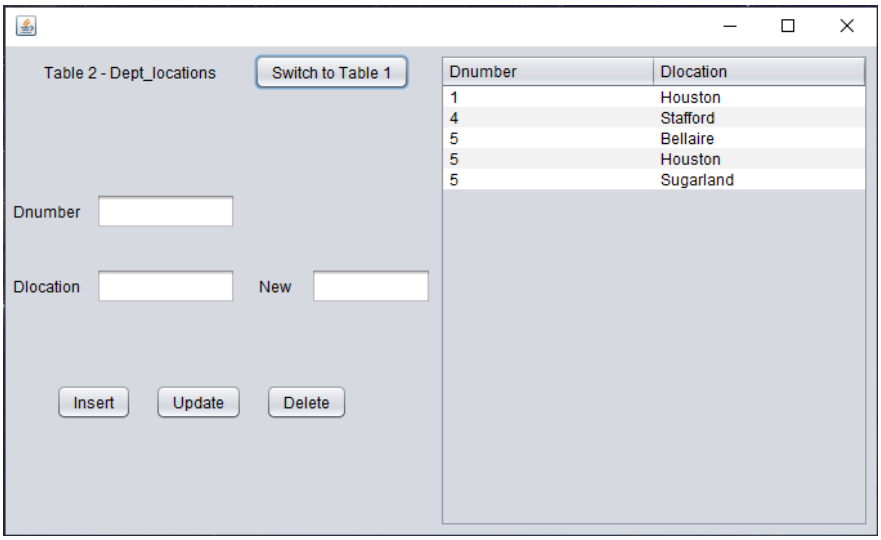

Figure 5: *GUI implementation of 'dept_locationt' table.*

GUI implementation is done using Java Swing library. Here I provided 2 separate Jforms for the two tables and connected them using a Jbutton.

Libraries used in this system.

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import javax.swing.JOptionPane;
import javax.swing.table.DefaultTableModel;
```

Figure 6: *Libraries used for the implementation.*

Connector used

In this system a middleware is needed to connect the written Java code with the SQL database that was created initially inorder to perform CREATE , UPDATE , DELETE operations. For this purpose, JDBC driver (version – 8.0.29) is used. This conncetor is added to the project depandencies as a JAR file to get the code connect to the database.

Code implementation

databaseConnection( ) function is created in because of a connection needed to perform any given task. (this function is created inside both the Jframes for table 1 and table 2)

```
//Function to initiate a connection to the database
private Connection databaseConnection(){
    try{
        con = DriverManager.getConnection("jdbc:mysql://localhost:3306/department_db", "root","");

        if(con != null){
            System.out.println("Connected to database successfully!");
            return con;
        }else{
            System.out.println("Failed to connect to the database.");
            return null;
        }
    }catch(SQLException ex ){
        JOptionPane.showMessageDialog(null, ex, "SQL connection error!", JOptionPane.ERROR_MESSAGE);
        return null;
    }
}
```

Figure 7: *databaseConnection fuction to esteblish a connection using JDBC driver.*

```
private Connection con = null;
private PreparedStatement pst = null;
private ResultSet rs = null;
```

Figure 8: *Common varibles used (declared inside both the Jframes)*

In this system data is provided in table form for visualization to the users. This is done by a Jtable and code for this is shown below.

```
//Load and display data in a table
public void showData(){
    con = databaseConnection();

    try{
      String sql = "SELECT * FROM department";
      pst = con.prepareStatement(sql);
      rs = pst.executeQuery();

      jTable1.setModel(new DefaultTableModel(null, new String[]{"Dname", "Dnumber", "Mgr_ssn", "Mgr_start_date"}));

      while(rs.next()){
          String Dname = rs.getString("Dname");
          String Dnumber = String.valueOf(rs.getInt("Dnumber"));
          String Mgr_ssn = String.valueOf(rs.getInt("Mgr_ssn"));
          String Mgr_start_date = rs.getString("Mgr_start_date");

          String data[] = {Dname, Dnumber, Mgr_ssn, Mgr_start_date};
          DefaultTableModel tableModel = (DefaultTableModel)jTable1.getModel();
          tableModel.addRow(data);
      }

    }catch(SQLException ex){
        JOptionPane.showMessageDialog(null, ex, "Data Loading Error!", JOptionPane.ERROR_MESSAGE);
    }
}
```

Figure 9: *implementation of showData( ) fucntion in Jframe1 for data visualization.*

```
//Load and display data in a table
public void showData(){
    con = databaseConnection();

    try{
      String sql = "SELECT * FROM dept_locations";
      pst = con.prepareStatement(sql);
      rs = pst.executeQuery();

      jTable2.setModel(new DefaultTableModel(null, new String[]{"Dnumber", "Dlocation"}));

      while(rs.next()){
          String Dnumber = String.valueOf(rs.getInt("Dnumber"));
          String Dlocation = rs.getString("Dlocation");

          String data[] = {Dnumber, Dlocation};
          DefaultTableModel tableModel = (DefaultTableModel)jTable2.getModel();
          tableModel.addRow(data);
      }

    }catch(SQLException ex){
        JOptionPane.showMessageDialog(null, ex, "Data Loading Error!", JOptionPane.ERROR_MESSAGE);
    }
}
```

Figure 10: *implementation of showData( ) fucntion in Jframe2 for data visualization.*

SEARCH operation

Search operation is only implemented for Jframe1(Table1) to populate relavant data for paticular Dnumber field. So that editing the record will be easy. In case of table 2 there is no search operation provided since there are only two columns and both the data act as combined primary keys (Since there are duplicate values in both columns)

```
private void SearchUsingDnumberActionPerformed(java.awt.event.ActionEvent evt) {
    String Dnumber = txtDnumber.getText();

    if(Dnumber.equals("")){
        JOptionPane.showMessageDialog(null,"Dnumber field is required for searching!", "Field Empty!", JOptionPane.ERROR_MESSAGE);
    }else{
        con = databaseConnection();
        try{
           String sql = "SELECT * FROM department WHERE Dnumber = ?";
           pst = con.prepareCall(sql);
           pst.setString(1, Dnumber);
           rs = pst.executeQuery();

           String Dname = "";
           while(rs.next()){

               Dname = rs.getString("Dname");
               String Mgr_ssn = rs.getString("Mgr_ssn");
               String Mgr_start_date = rs.getString("Mgr_start_date");

               txtDname.setText(Dname);
               txtMgr_ssn.setText(Mgr_ssn);
               txtMgr_start_date.setText(Mgr_start_date);
           }

           if("".equals(Dname)){
            txtDname.setText("");
            txtMgr_ssn.setText("");
            txtMgr_start_date.setText("");

             JOptionPane.showMessageDialog(null, "No data found!", "Invalid Entry", JOptionPane.ERROR_MESSAGE);
            }
        }catch(SQLException ex){
            JOptionPane.showMessageDialog(null, ex, "Access Error!", JOptionPane.ERROR_MESSAGE);
            showData();
        }
```

Figure 11: *implementation of SearchUsingDnumberActionPerformed( ) fucntion for table 1*

INSERT operation

```
private void InsertActionPerformed(java.awt.event.ActionEvent evt) {
    String Dname = txtDname.getText();
    String Dnumber = txtDnumber.getText();
    String Mgr_ssn = txtMgr_ssn.getText();
    String Mgr_start_date = txtMgr_start_date.getText();

    if(Dname.equals("") || Dnumber.equals("") || Mgr_ssn.equals("") || Mgr_start_date.equals("")){
        JOptionPane.showMessageDialog(null,"Please fill every field.", "Empty Field(s)!", JOptionPane.ERROR_MESSAGE);
    }else{
        con = databaseConnection();
        jTable1.setModel(new DefaultTableModel(null, new String[]{"Dname", "Dnumber", "Mgr_ssn", "Mgr_start_date"}));

        try{
            String sql = "INSERT INTO department" +
                    "(Dname, Dnumber, Mgr_ssn, Mgr_start_date)" +
                    "VALUES (?,?,?,?)";

            pst = con.prepareStatement(sql);
            pst.setString(1, Dname);
            pst.setInt(2, Integer.parseInt(Dnumber));
            pst.setInt(3,Integer.parseInt(Mgr_ssn));
            pst.setString(4, Mgr_start_date);

            pst.executeUpdate();
            JOptionPane.showMessageDialog(null, "Inserted Successfully.");
            showData();

            txtDname.setText("");
            txtDnumber.setText("");
            txtMgr_ssn.setText("");
            txtMgr_start_date.setText("");

        }catch(SQLException ex ){
            JOptionPane.showMessageDialog(null, ex);
            showData();
        }
```

Figure 12: *implementation of InsertActionPerformed ( ) fucntion for table 1*

```
private void InsertActionPerformed(java.awt.event.ActionEvent evt) {

    String Dnumber = txtDnumber.getText();
    String Dlocation = txtDlocation.getText();

    if(Dnumber.equals("") || Dlocation.equals("")){
        JOptionPane.showMessageDialog(null,"Please fill every field.", "Empty Field(s)!", JOptionPane.ERROR_MESSAGE);
    }else{
        con = databaseConnection();
        jTable2.setModel(new DefaultTableModel(null, new String[]{"Dnumber", "Dlocation"}));

        try{
            String sql = "INSERT INTO dept_locations" +
                    "(Dnumber, Dlocation)" +
                    "VALUES (?,?)";

            pst = con.prepareStatement(sql);

            pst.setInt(1, Integer.parseInt(Dnumber));
            pst.setString(2, Dlocation);

            pst.executeUpdate();
            JOptionPane.showMessageDialog(null, "Inserted Successfully.");
            showData();

            txtDnumber.setText("");
            txtDlocation.setText("");

        }catch(SQLException ex){
            JOptionPane.showMessageDialog(null, ex);
            showData();
        }
    }
}
```

Figure 13: *implementation of InsertActionPerformed ( ) fucntion for table 2*

UPDATE operation

```
private void UpdateActionPerformed(java.awt.event.ActionEvent evt) {
    String Dname = txtDname.getText();
    String Dnumber = txtDnumber.getText();
    String Mgr_ssn = txtMgr_ssn.getText();
    String Mgr_start_date = txtMgr_start_date.getText();

    if(Dname.equals("") || Dnumber.equals("") || Mgr_ssn.equals("") || Mgr_start_date.equals("")){
        JOptionPane.showMessageDialog(null,"Please fill every field to update.", "Empty Field(s)!", JOptionPane.ERROR_MESSAGE);
    }else{
        con = databaseConnection();

        try{
            String sql = "UPDATE department SET Dname= ? , Mgr_ssn= ? , Mgr_start_date= ? WHERE Dnumber= ? ";

            pst = con.prepareStatement(sql);
            pst.setString(1, Dname);
            pst.setInt(2, Integer.parseInt(Mgr_ssn));
            pst.setString(3, Mgr_start_date);
            pst.setInt(4, Integer.parseInt(Dnumber));

            int rowsEffected = pst.executeUpdate();
            if(rowsEffected > 0){
               JOptionPane.showMessageDialog(null, "Data updated Successfully.");
            }
            showData();
            txtDname.setText("");
            txtDnumber.setText("");
            txtMgr_ssn.setText("");
            txtMgr_start_date.setText("");

        }catch(SQLException ex ){
            JOptionPane.showMessageDialog(null, ex);
            showData();
        }
    }
}
```

Figure 14: *implementation of UpdateActionPerformed ( ) fucntion for table 1*

```
private void UpdateActionPerformed(java.awt.event.ActionEvent evt) {

    String Dnumber = txtDnumber.getText();
    String Dlocation = txtDlocation.getText();
    String newDlocation = txtNewDlocation.getText();

    if(Dnumber.equals("") || Dlocation.equals("") || newDlocation.equals("")){
        JOptionPane.showMessageDialog(null,"Please fill every field to update.", "Empty Field(s)!", JOptionPane.ERROR_MESSAGE);
        if(newDlocation.equals("")){
            JOptionPane.showMessageDialog(null,"Please add new location to update", "Add new location", JOptionPane.ERROR_MESSAGE);
        }
    }else{
        con = databaseConnection();

        try{
            String sql = "UPDATE dept_locations SET Dnumber= ? , Dlocation= ?  WHERE Dnumber= ? AND Dlocation= ?";

            pst = con.prepareStatement(sql);

            pst.setInt(1, Integer.parseInt(Dnumber));
            pst.setString(2, newDlocation);
            pst.setInt(3, Integer.parseInt(Dnumber));
            pst.setString(4, Dlocation);

            int rowsEffected = pst.executeUpdate();
            if(rowsEffected > 0){
               JOptionPane.showMessageDialog(null, "Data updated Successfully.");
            }

            showData();
            txtDnumber.setText("");
            txtDlocation.setText("");
            txtNewDlocation.setText("");

        }catch(SQLException ex ){
            JOptionPane.showMessageDialog(null, ex);
            showData();
        }
    }
}
```

Figure 15: *implementation of UpdateActionPerformed ( ) fucntion for table 2*

DELETE operation

```
private void DeleteActionPerformed(java.awt.event.ActionEvent evt) {
    String Dnumber = txtDnumber.getText();

    if(Dnumber.equals("")){
        JOptionPane.showMessageDialog(null,"Dnumber field is required for deleting a data", "Dnumber required!", JOptionPane.ERROR_MESSAGE);
    }else{
        con = databaseConnection();

        try{
          String sql = "DELETE FROM department WHERE Dnumber = ?";

          pst = con.prepareStatement(sql);
          pst.setString(1, Dnumber);
          int rowsAffected = pst.executeUpdate();

          if(rowsAffected > 0){
             JOptionPane.showMessageDialog(null, "Data deleted Successfully.");
          }

          showData();
          txtDname.setText("");
          txtDnumber.setText("");
          txtMgr_ssn.setText("");
          txtMgr_start_date.setText("");

         }catch(SQLException ex ){
            JOptionPane.showMessageDialog(null, ex);
            showData();
         }
    }
}
```

Figure 16: *implementation of I DeleteActionPerformed ( ) fucntion for table 1*

```
private void DeleteActionPerformed(java.awt.event.ActionEvent evt) {
    String Dnumber = txtDnumber.getText();
    String Dlocation = txtDlocation.getText();

    if(Dnumber.equals("") || Dlocation.equals("")){
        JOptionPane.showMessageDialog(null,"Dnumber and Dlocation fields are required for deleting a data", "Fill required fields!", JOptionPane.ERROR_MES
    }else{
        con = databaseConnection();

        try{
          String sql = "DELETE FROM dept_locations WHERE Dnumber = ? AND Dlocation = ?";

          pst = con.prepareStatement(sql);
          pst.setInt(1, Integer.parseInt(Dnumber));
          pst.setString(2, Dlocation);

          int rowsAffected = pst.executeUpdate();

          if(rowsAffected > 0){
             JOptionPane.showMessageDialog(null, "Data deleted Successfully.");
          }

          showData();

          txtDnumber.setText("");
          txtDlocation.setText("");

         }catch(SQLException ex ){
            JOptionPane.showMessageDialog(null, ex);
            showData();
         }
    }
```

Figure 17: *implementation of I DeleteActionPerformed ( ) fucntion for table 2*

How to avoid mismatch impedence

Mismatch impedence is the problems that arise due to difference in database model and programming model

1) Datatype mismatch - attribute data type in the Java language may differ from the attribute data type in the SQL database. To reduce the risks most appropriate data types were used in Java code implementation.
   Ex: Java – String, SQL– varchar, also used Integer.parse(String <variable>) to change the input string to integers when needed.

2) Majority of queries are sets of tuples, with each tuple consisting of a sequence of attribute values. For processing, the program requires access to the individual data values within individual tuples. Looping is implemented over tuples in query result to access a single tuple at a time and to get individual values from tuple. This is done in search method, to get the attribute, looping is implemented by next method of ResultSet object,

   while (rs.next()){
   Dname = rs.getString(“Dname”);
   ……
   }

Sample output

- Create

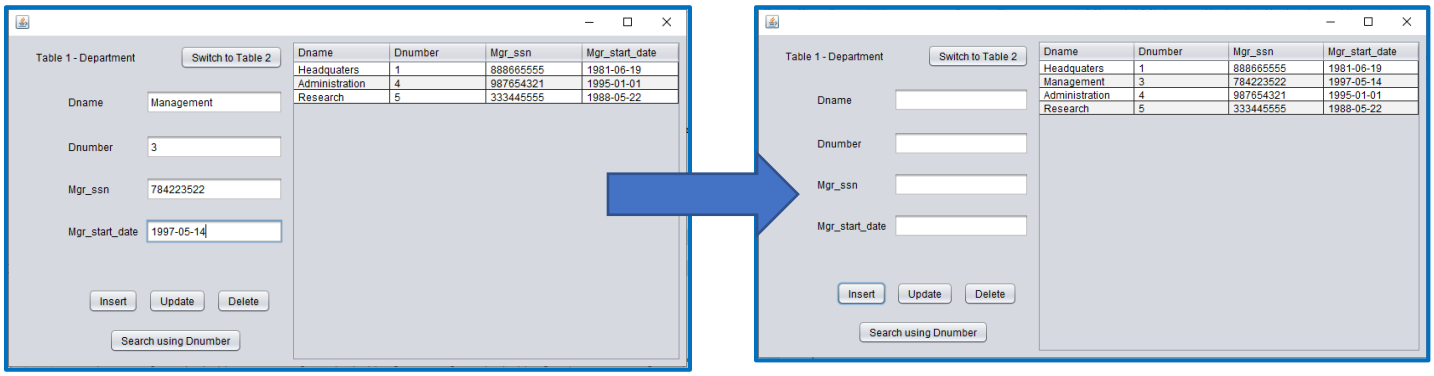

Figure 18: *Demonstration of insert operation in table 1 (Inserting new entry 'Management')*

- Search

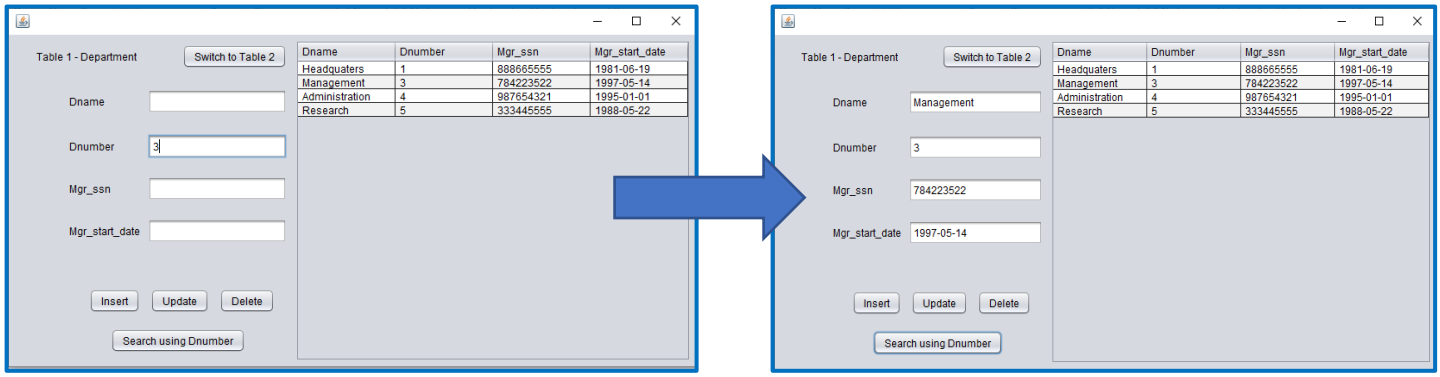

Figure 19: *Demonstration of search operation in table 1 (Searching using Dnumber = 3)*

- Update

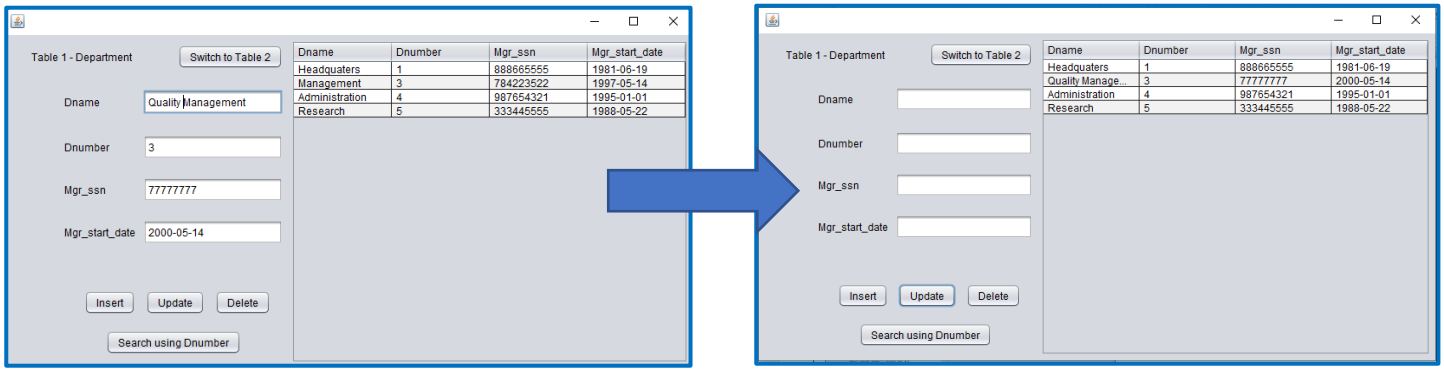

Figure 20: *Demonstration of update operation in table 1 (changing values of Dname, Mgr_ssn and Mgr_start_date)*

- Delete

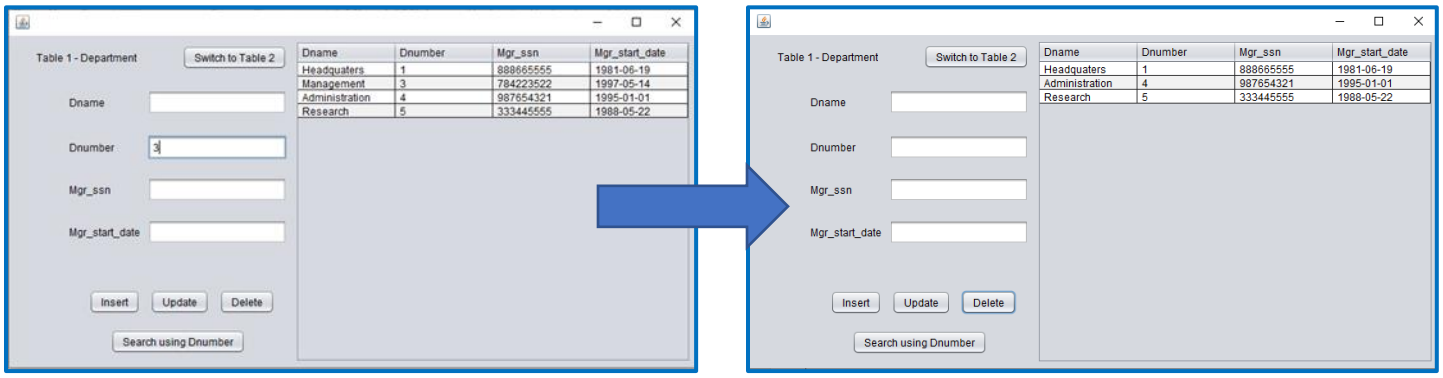

Figure 21: *Demonstration of delete operation in table 1 (Searching using Dnumber = 3 and deleting the entry)*

After performing each of these operation a message dialog box will appear indicating the state of the operation or possible errors. All the possible errors are handle including SQLexceptions.