# ASSIGNMENT 03

|  |  |
|---|---|
| NAME | : G.D.M. PRAMODI |
| INDEX NO. | : 21/ENG/028 |
| REGISTRATION NO. | : 102829 |
| SUBMISSION DATE | : 09/07/2024 |

## Data Types

Table 01: Data types of attributes of DEPARTMENT Table

| Attribute | Data Type in SQL | Data Type in Java |
|---|---|---|
| Dname | Varchar | String |
| Dnumber | Int | Int |
| Mgr_ssn | Int | Int |
| Mgr_start_date | Date | String |

Table 01: Data types of attributes of DEPT_LOCATION Table

| Attribute | Data Type in SQL | Data Type in Java |
|---|---|---|
| Dnumber | Int | Int |
| Dlocation | Varchar | String |

## Creating the Database using MySQL

A database was created named CompanyDB.

Then two tables were created as DEPARTMENT and DEPT_LOCATIONS.

Set the Primary keys and Foreign keys for each table.

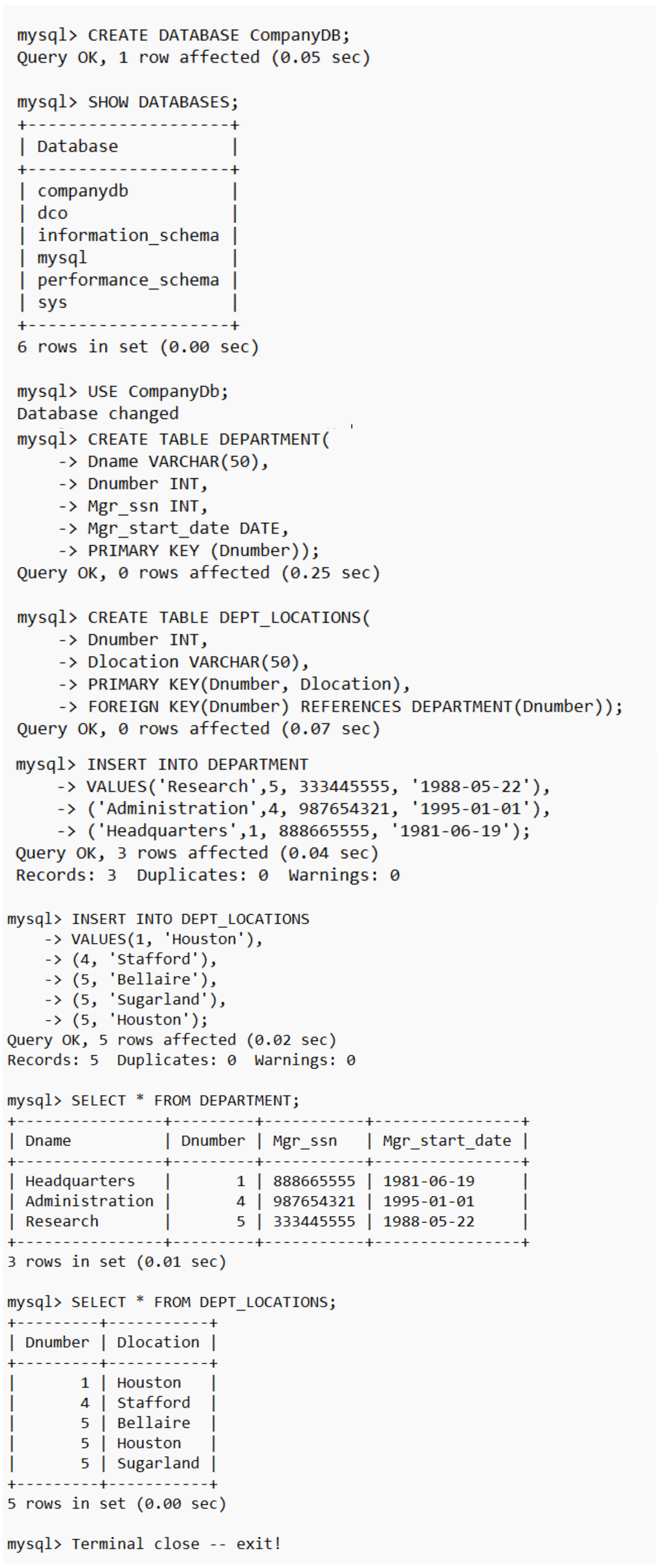
DEPARTMENT Table

Primary Key – Dnumber

DEPT_LOCATIONS

Primary Key – Dnumber + Dlocation

Foreign Key - Dnumber



```
mysql> CREATE DATABASE CompanyDB;
Query OK, 1 row affected (0.05 sec)

mysql> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| companydb          |
| dco                |
| information_schema |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
6 rows in set (0.00 sec)

mysql> USE CompanyDb;
Database changed
mysql> CREATE TABLE DEPARTMENT(
    -> Dname VARCHAR(50),
    -> Dnumber INT,
    -> Mgr_ssn INT,
    -> Mgr_start_date DATE,
    -> PRIMARY KEY (Dnumber));
Query OK, 0 rows affected (0.25 sec)

mysql> CREATE TABLE DEPT_LOCATIONS(
    -> Dnumber INT,
    -> Dlocation VARCHAR(50),
    -> PRIMARY KEY(Dnumber, Dlocation),
    -> FOREIGN KEY(Dnumber) REFERENCES DEPARTMENT(Dnumber));
Query OK, 0 rows affected (0.07 sec)

mysql> INSERT INTO DEPARTMENT
    -> VALUES('Research',5, 333445555, '1988-05-22'),
    -> ('Administration',4, 987654321, '1995-01-01'),
    -> ('Headquarters',1, 888665555, '1981-06-19');
Query OK, 3 rows affected (0.04 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql> INSERT INTO DEPT_LOCATIONS
    -> VALUES(1, 'Houston'),
    -> (4, 'Stafford'),
    -> (5, 'Bellaire'),
    -> (5, 'Sugarland'),
    -> (5, 'Houston');
Query OK, 5 rows affected (0.02 sec)
Records: 5  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM DEPARTMENT;
+----------------+---------+-----------+----------------+
| Dname          | Dnumber | Mgr_ssn   | Mgr_start_date |
+----------------+---------+-----------+----------------+
| Headquarters   |       1 | 888665555 | 1981-06-19     |
| Administration |       4 | 987654321 | 1995-01-01     |
| Research       |       5 | 333445555 | 1988-05-22     |
+----------------+---------+-----------+----------------+
3 rows in set (0.01 sec)

mysql> SELECT * FROM DEPT_LOCATIONS;
+---------+-----------+
| Dnumber | Dlocation |
+---------+-----------+
|       1 | Houston   |
|       4 | Stafford  |
|       5 | Bellaire  |
|       5 | Houston   |
|       5 | Sugarland |
+---------+-----------+
5 rows in set (0.00 sec)

mysql> Terminal close -- exit!
```

Figure 01: MySQL Database Tee File

## Java Database Connectivity (JDBC)

SQL can be called from the Java object-oriented programming language. It is important to download install and import necessary class libraries which are called java.sql before JDBC function calls.

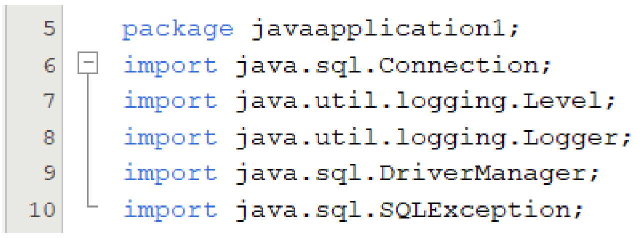
1. Add MySQL Connector to project

   Download MySQL Connector/J and add the JAR file to my NetBeans project.

   mysql – connector – j-9.0.0.jar

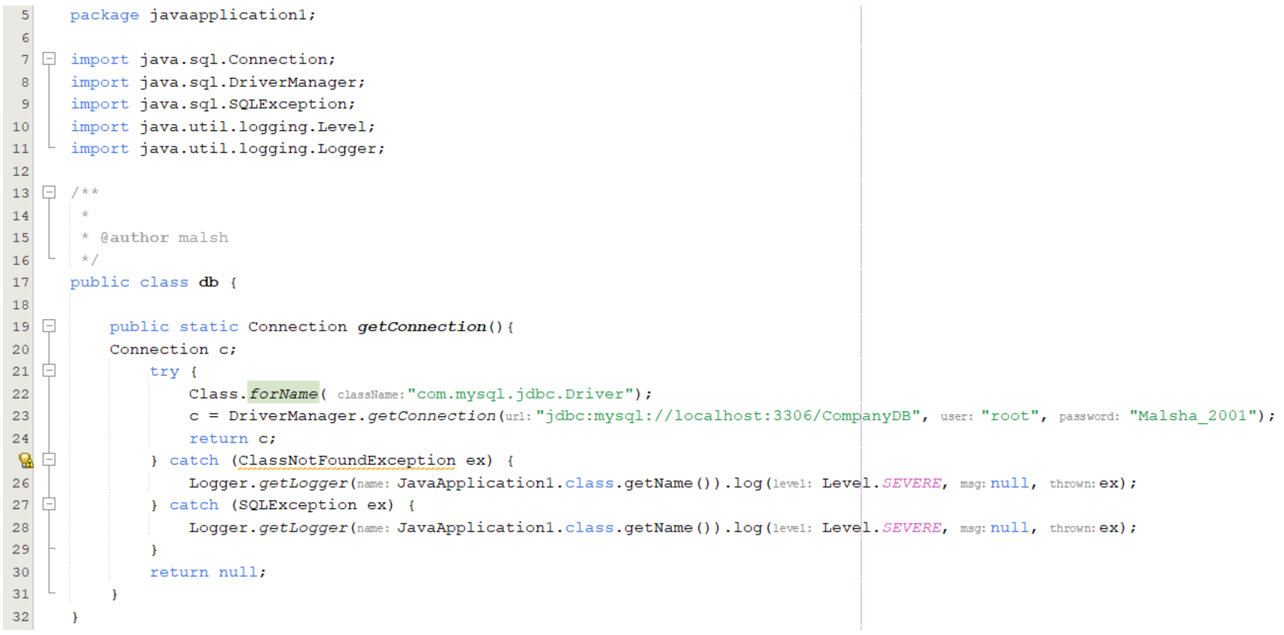
2. Establish the connection

   Import the necessary files.



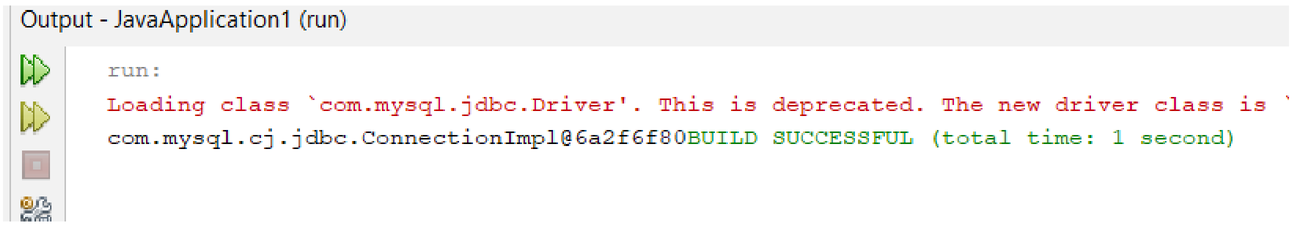
```java
package javaapplication1;
import java.sql.Connection;
import java.util.logging.Level;
import java.util.logging.Logger;
import java.sql.DriverManager;
import java.sql.SQLException;
```

Then created an object named ‘c’ and established the connection using jdbc Driver in a class file named as “db”.



```java
package javaapplication1;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.util.logging.Level;
import java.util.logging.Logger;

/**
 *
 * @author malsh
 */
public class db {

    public static Connection getConnection(){
    Connection c;
        try {
            Class.forName( className:"com.mysql.jdbc.Driver");
            c = DriverManager.getConnection(url: "jdbc:mysql://localhost:3306/CompanyDB", user: "root", password: "Malsha_2001");
            return c;
        } catch (ClassNotFoundException ex) {
            Logger.getLogger(name: JavaApplication1.class.getName()).log(level: Level.SEVERE, msg: null, thrown: ex);
        } catch (SQLException ex) {
            Logger.getLogger(name: JavaApplication1.class.getName()).log(level: Level.SEVERE, msg: null, thrown: ex);
        }
        return null;
    }
}
```

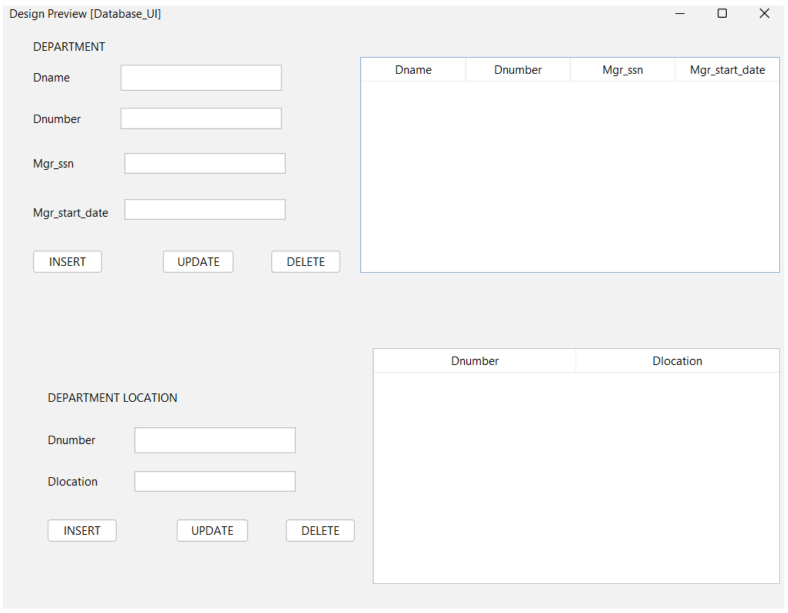
Run the code to check the status of the connection. The connection was built successfully as shown below.



Output - JavaApplication1 (run)

```
run:
Loading class `com.mysql.jdbc.Driver'. This is deprecated. The new driver class is `
com.mysql.cj.jdbc.ConnectionImpl@6a2f6f80BUILD SUCCESSFUL (total time: 1 second)
```

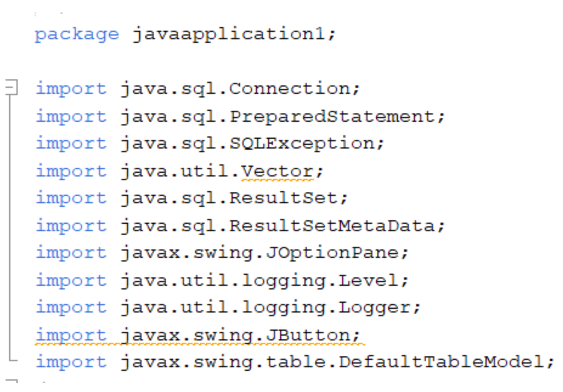
Designed a simple UI for easy interaction using JFrames.



Developed the functions for each button to perform the process Insert, Update and Delete data in a particular table. All the functions are developed within the relevant Button Action listener using JAVA in the NetBeans IDE.

**INSERT**

First, the necessary libraries are imported into the code file as shown below.

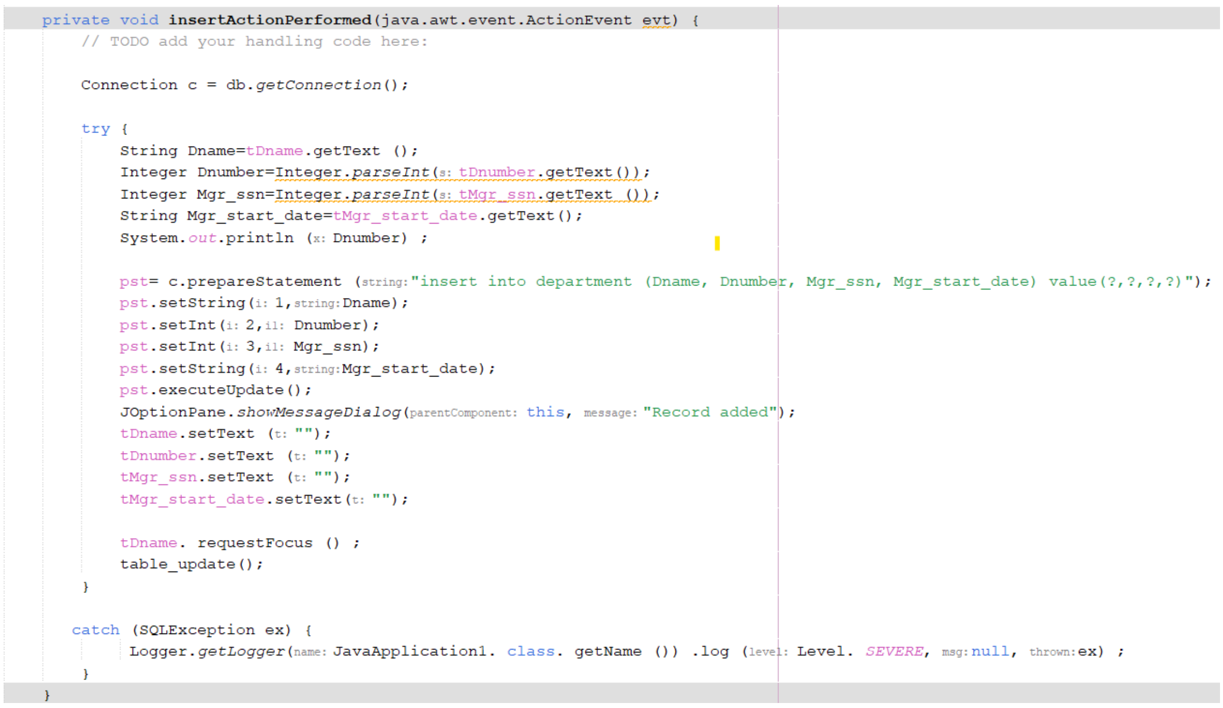


```
package javaapplication1;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import java.util.Vector;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import javax.swing.JOptionPane;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.JButton;
import javax.swing.table.DefaultTableModel;
```

Next, the two Insert functions are developed for the two INSERT buttons in each table.

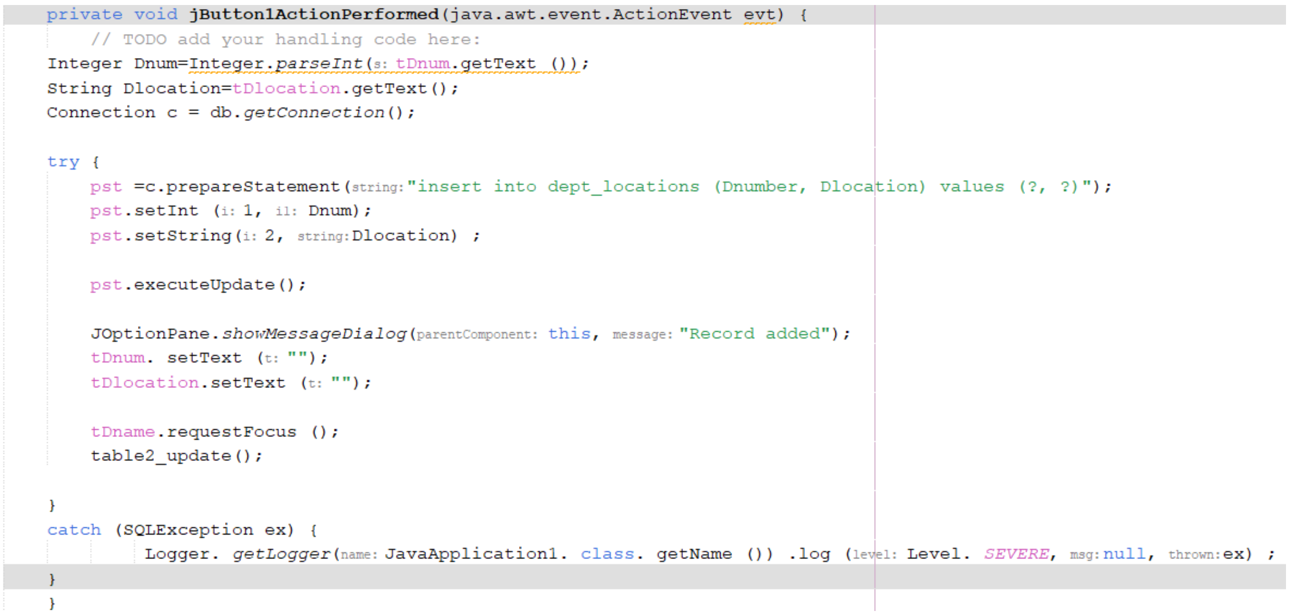
Insert Function for the Department Table:

The method ‘insertActionPerformed’ is triggered when the insert button is clicked. Within that method, a connection to the database is established using ‘db.getConnection()’ where db is the custom database connection class created at the beginning. The text fields in the UI (‘tDname’, ‘tDnumber’, ‘tMgr_ssn’, ‘tMgr_start_date’) are used to get the user input for all the functions used in every method. A ‘prepared statement’ (defined as ‘pst’) is used to construct and execute an SQL ‘INSERT’ query to insert the values to the ‘DEPARTMENT’ Table. 'pet.setString’, ‘pet.setInt’ methods are used to set the values in the query. Finally, the ‘INSERT’ statement is executed by ‘executeUpdate()’ method. For further confirmation, a confirmation message is displayed using ‘JOptionPane.showMessageDialog’. Records are displayed in the Table in the UI using the table_update() function. A similar process is applied to the Dept_Location Insert Function.



```
private void insertActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:

    Connection c = db.getConnection();

    try {
        String Dname=tDname.getText ();
        Integer Dnumber=Integer.parseInt(s: tDnumber.getText());
        Integer Mgr_ssn=Integer.parseInt(s: tMgr_ssn.getText ());
        String Mgr_start_date=tMgr_start_date.getText();
        System.out.println (x: Dnumber) ;

        pst= c.prepareStatement (string:"insert into department (Dname, Dnumber, Mgr_ssn, Mgr_start_date) value(?,?,?,?)");
        pst.setString(i: 1,string:Dname);
        pst.setInt(i: 2,i1: Dnumber);
        pst.setInt(i: 3,i1: Mgr_ssn);
        pst.setString(i: 4,string:Mgr_start_date);
        pst.executeUpdate();
        JOptionPane.showMessageDialog(parentComponent: this, message: "Record added");
        tDname.setText (t: "");
        tDnumber.setText (t: "");
        tMgr_ssn.setText (t: "");
        tMgr_start_date.setText(t: "");

        tDname. requestFocus () ;
        table_update();
    }

   catch (SQLException ex) {
         Logger.getLogger(name: JavaApplication1. class. getName ()) .log (level: Level. SEVERE, msg:null, thrown:ex) ;
    }
}
```

Figure 02: INSERT Function for Department table

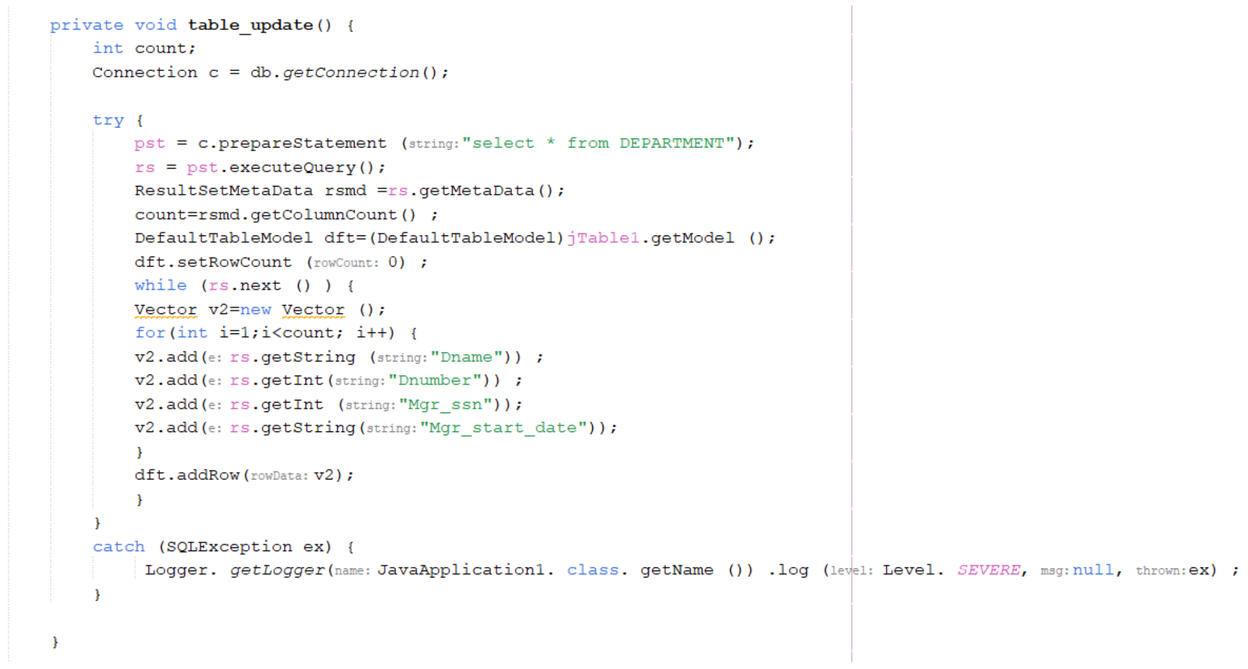


```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
Integer Dnum=Integer.parseInt(s: tDnum.getText ());
String Dlocation=tDlocation.getText();
Connection c = db.getConnection();

try {
    pst =c.prepareStatement(string:"insert into dept_locations (Dnumber, Dlocation) values (?, ?)");
    pst.setInt (i: 1, i1: Dnum);
    pst.setString(i: 2, string:Dlocation) ;

    pst.executeUpdate();

    JOptionPane.showMessageDialog(parentComponent: this, message: "Record added");
    tDnum. setText (t: "");
    tDlocation.setText (t: "");

    tDname.requestFocus ();
    table2_update();

}
catch (SQLException ex) {
        Logger. getLogger(name: JavaApplication1. class. getName ()) .log (level: Level. SEVERE, msg:null, thrown:ex) ;
}
}
```

Figure 03: INSERT Function for Dept_location Table

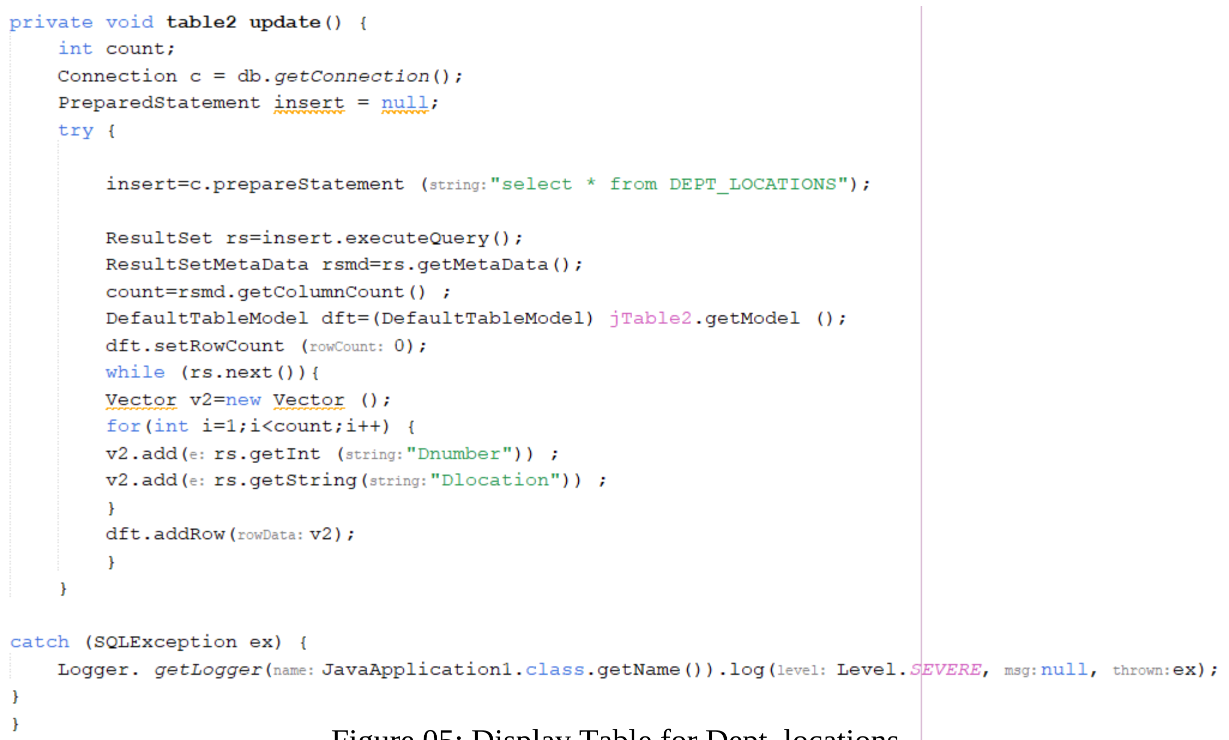
Table_update method in JAVA used to update the current records from the two tables in MySQL database. ‘rs = pst.executeQuery()’ executes the query and stores the result set in ‘rs’. ‘ResultSetMetadata’ is used to get the number of columns in the result set. The ‘while’ loop iterates through the result set and the ‘for’ loop adds each column value from the result set to the vector.

‘import javax.swing.table.DefaultTableModel’ to manage and update the table model.



```
private void table_update() {
    int count;
    Connection c = db.getConnection();

    try {
        pst = c.prepareStatement (string:"select * from DEPARTMENT");
        rs = pst.executeQuery();
        ResultSetMetaData rsmd =rs.getMetaData();
        count=rsmd.getColumnCount() ;
        DefaultTableModel dft=(DefaultTableModel)jTable1.getModel ();
        dft.setRowCount (rowCount: 0) ;
        while (rs.next () ) {
        Vector v2=new Vector ();
        for(int i=1;i<count; i++) {
        v2.add(e: rs.getString (string:"Dname")) ;
        v2.add(e: rs.getInt(string:"Dnumber")) ;
        v2.add(e: rs.getInt (string:"Mgr_ssn"));
        v2.add(e: rs.getString(string:"Mgr_start_date"));
        }
        dft.addRow(rowData: v2);
        }
    }
    catch (SQLException ex) {
        Logger. getLogger(name: JavaApplication1. class. getName ()) .log (level: Level. SEVERE, msg:null, thrown:ex) ;
    }

}
```

Figure 04: Display Table for Department

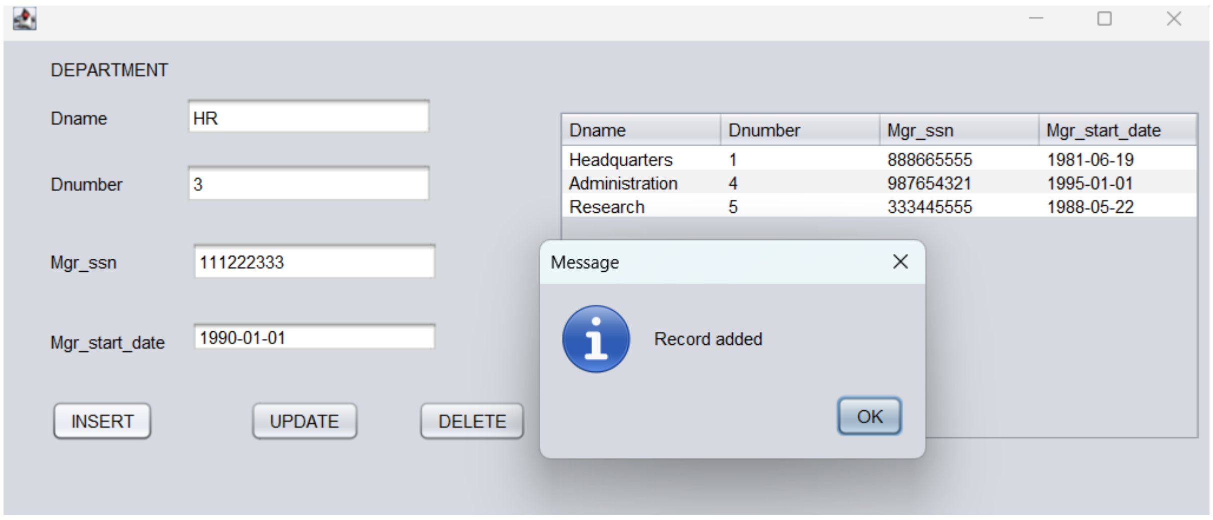


```
private void table2 update() {
    int count;
    Connection c = db.getConnection();
    PreparedStatement insert = null;
    try {

        insert=c.prepareStatement (string:"select * from DEPT_LOCATIONS");

        ResultSet rs=insert.executeQuery();
        ResultSetMetaData rsmd=rs.getMetaData();
        count=rsmd.getColumnCount() ;
        DefaultTableModel dft=(DefaultTableModel) jTable2.getModel ();
        dft.setRowCount (rowCount: 0);
        while (rs.next()){
        Vector v2=new Vector ();
        for(int i=1;i<count;i++) {
        v2.add(e: rs.getInt (string:"Dnumber")) ;
        v2.add(e: rs.getString(string:"Dlocation")) ;
        }
        dft.addRow(rowData: v2);
        }
    }

catch (SQLException ex) {
    Logger. getLogger(name: JavaApplication1.class.getName()).log(level: Level.SEVERE, msg:null, thrown:ex);
}
}
```

Figure 05: Display Table for Dept_locations

## Output

When the file is executed, the following output can be obtained by entering the relevant data into the file. Since this is a relational database, it is allowed to enter data to the Dept_Location table if there is a relevant record stored in the Department Table referring the Dnumber. When the data is entered and press the INSERT button is will provide a confirmation message as shown in Figure 06.


DEPARTMENT

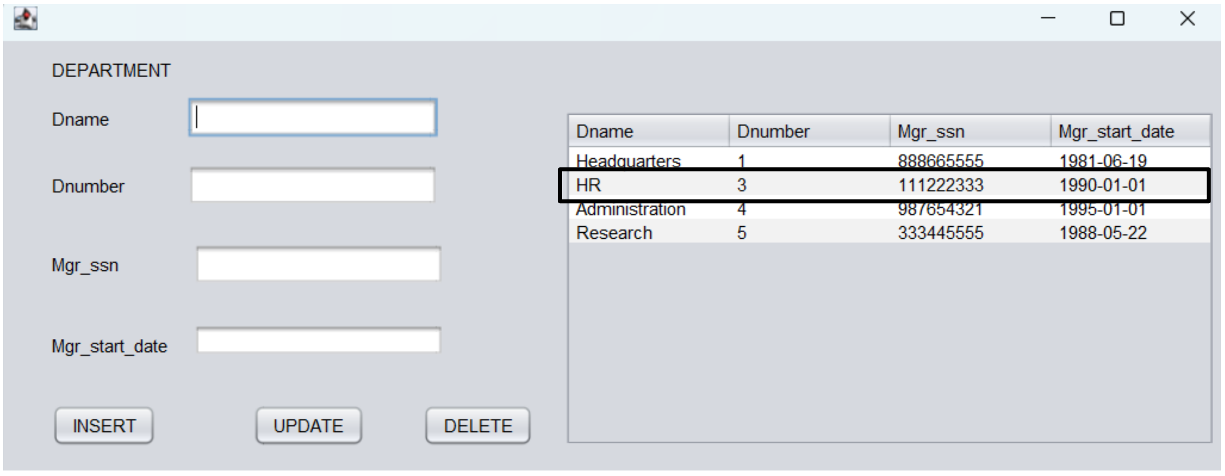
| | |
|---|---|
| Dname | HR |
| Dnumber | 3 |
| Mgr_ssn | 111222333 |
| Mgr_start_date | 1990-01-01 |

| Dname | Dnumber | Mgr_ssn | Mgr_start_date |
|---|---|---|---|
| Headquarters | 1 | 888665555 | 1981-06-19 |
| Administration | 4 | 987654321 | 1995-01-01 |
| Research | 5 | 333445555 | 1988-05-22 |

Message

Record added

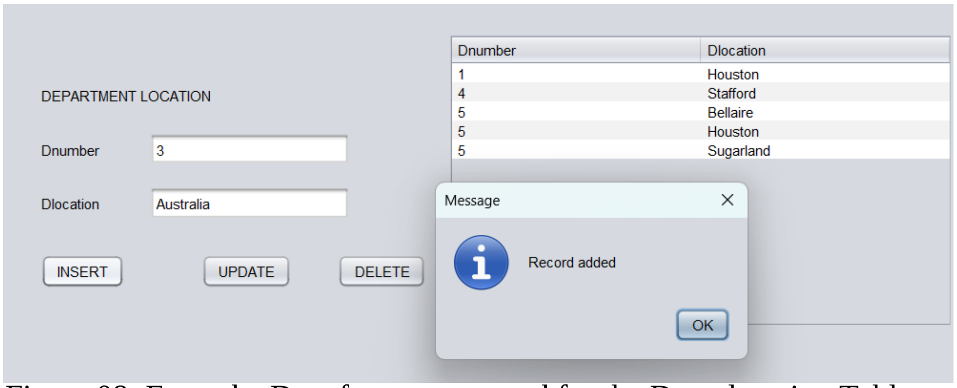
Figure 06: Enter the Data for a new record for the Department Table


DEPARTMENT

| | |
|---|---|
| Dname | |
| Dnumber | |
| Mgr_ssn | |
| Mgr_start_date | |

| Dname | Dnumber | Mgr_ssn | Mgr_start_date |
|---|---|---|---|
| Headquarters | 1 | 888665555 | 1981-06-19 |
| HR | 3 | 111222333 | 1990-01-01 |
| Administration | 4 | 987654321 | 1995-01-01 |
| Research | 5 | 333445555 | 1988-05-22 |

Figure 07: New Record added successfully for the Department Table


DEPARTMENT LOCATION

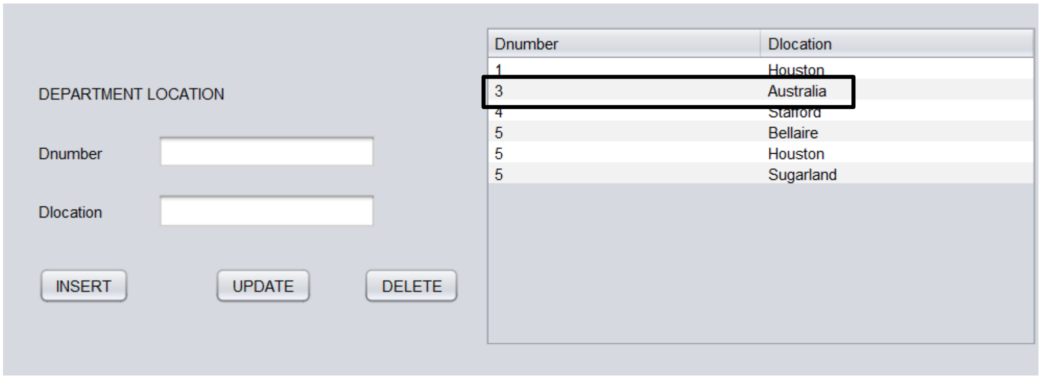
| | |
|---|---|
| Dnumber | 3 |
| Dlocation | Australia |

| Dnumber | Dlocation |
|---|---|
| 1 | Houston |
| 4 | Stafford |
| 5 | Bellaire |
| 5 | Houston |
| 5 | Sugarland |

Message

Record added

Figure 08: Enter the Data for a new record for the Dept_location Table


DEPARTMENT LOCATION

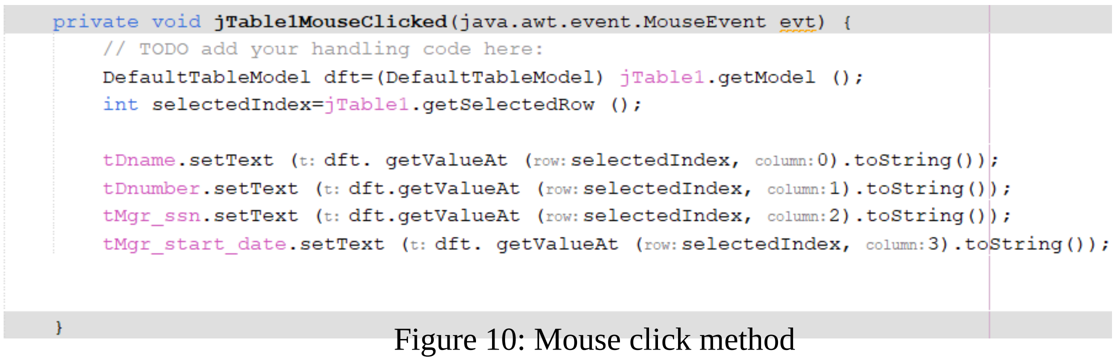
| | |
|---|---|
| Dnumber | |
| Dlocation | |

| Dnumber | Dlocation |
|---|---|
| 1 | Houston |
| 3 | Australia |
| 4 | Stafford |
| 5 | Bellaire |
| 5 | Houston |
| 5 | Sugarland |

Figure 09: New Record added successfully for the Dept_location Table

## UPDATE

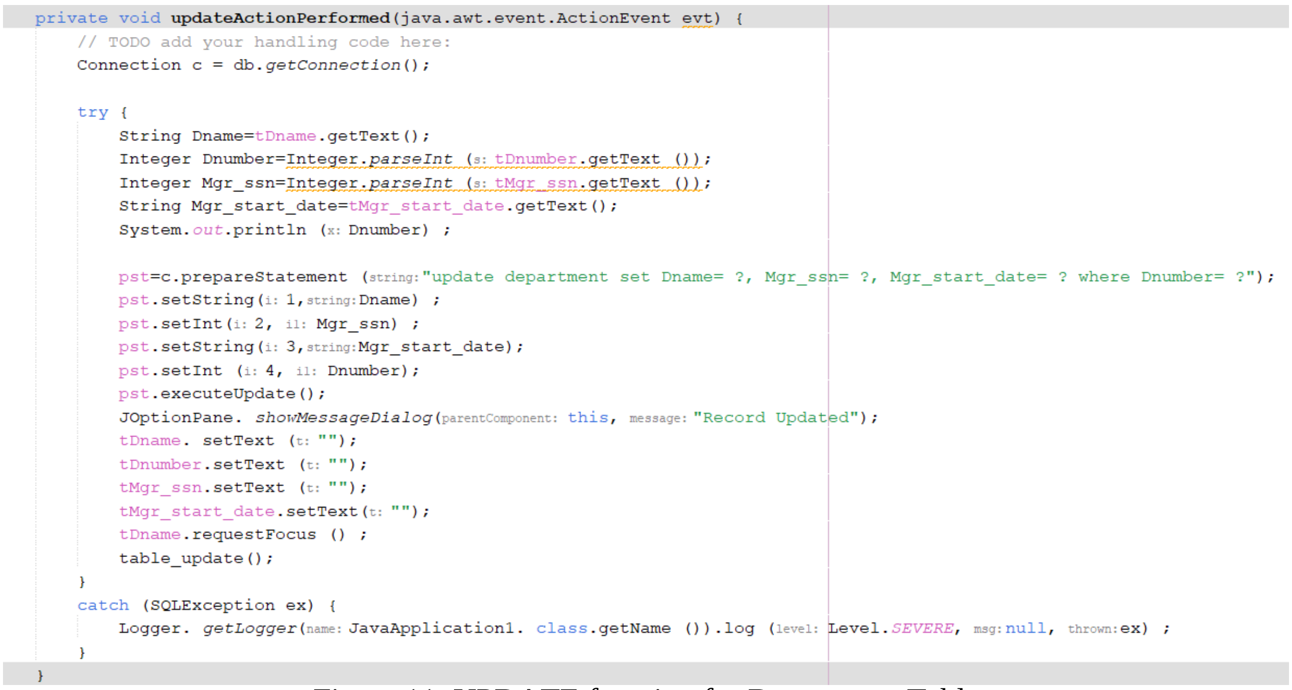
To make the update method more user friendly the Mouse Click Event is used for both tables in a similar manner as shown in Figure 10. This method populates the form fields with the values from the selected row in the table which allows the user to see the details of the relevant record and make necessary changes.



```
private void jTable1MouseClicked(java.awt.event.MouseEvent evt) {
    // TODO add your handling code here:
    DefaultTableModel dft=(DefaultTableModel) jTable1.getModel ();
    int selectedIndex=jTable1.getSelectedRow ();

    tDname.setText (t: dft. getValueAt (row: selectedIndex, column: 0).toString());
    tDnumber.setText (t: dft.getValueAt (row: selectedIndex, column: 1).toString());
    tMgr_ssn.setText (t: dft.getValueAt (row: selectedIndex, column: 2).toString());
    tMgr_start_date.setText (t: dft. getValueAt (row: selectedIndex, column: 3).toString());


}
```

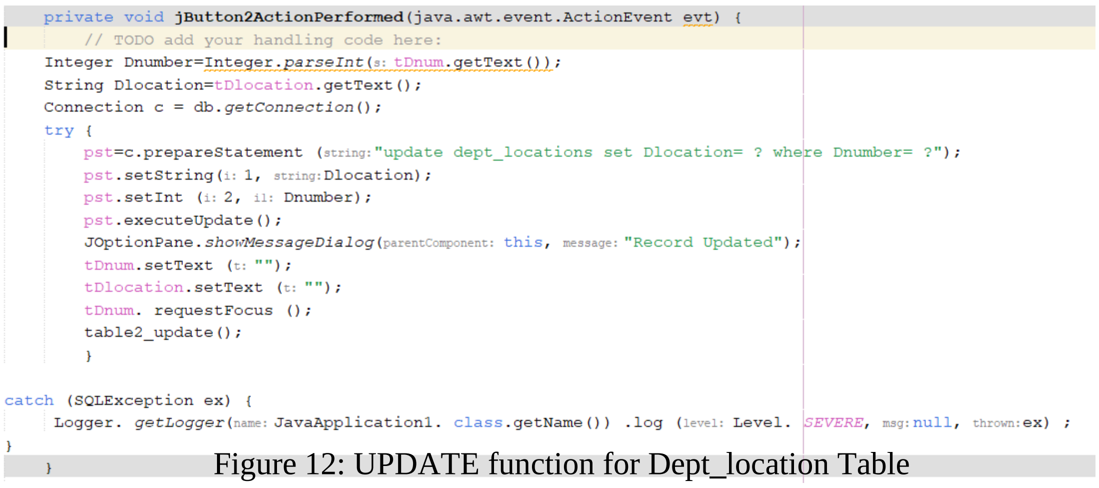
Figure 10: Mouse click method

For the update method, a preparedStatement is used as before to execute the SQL ‘UPDATE’ query to update a record in the ‘DEPARTMENT’ table based on the ‘Dnumber’. Similar method is used for the ‘DEPT_LOCATION’ table as well.



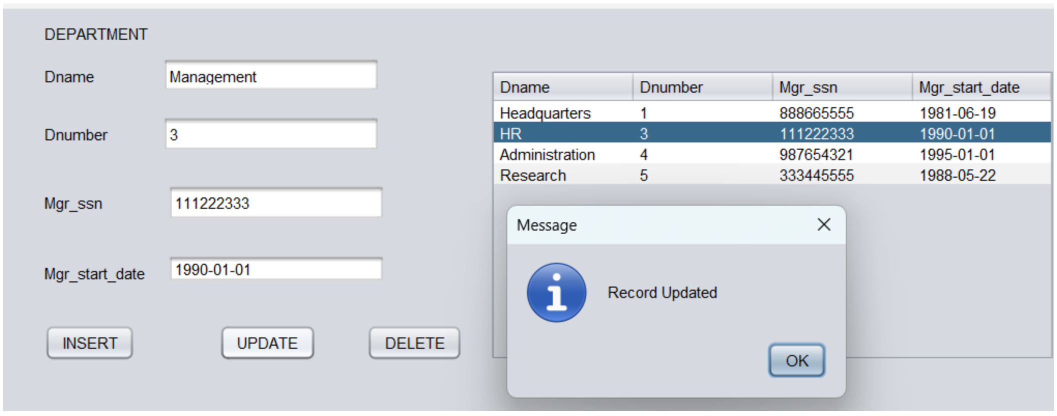
```
private void updateActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    Connection c = db.getConnection();

    try {
        String Dname=tDname.getText();
        Integer Dnumber=Integer.parseInt (s: tDnumber.getText ());
        Integer Mgr_ssn=Integer.parseInt (s: tMgr_ssn.getText ());
        String Mgr_start_date=tMgr_start_date.getText();
        System.out.println (x: Dnumber) ;

        pst=c.prepareStatement (string: "update department set Dname= ?, Mgr_ssn= ?, Mgr_start_date= ? where Dnumber= ?");
        pst.setString(i: 1, string: Dname) ;
        pst.setInt(i: 2, i1: Mgr_ssn) ;
        pst.setString(i: 3, string: Mgr_start_date);
        pst.setInt (i: 4, i1: Dnumber);
        pst.executeUpdate();
        JOptionPane. showMessageDialog(parentComponent: this, message: "Record Updated");
        tDname. setText (t: "");
        tDnumber.setText (t: "");
        tMgr_ssn.setText (t: "");
        tMgr_start_date.setText(t: "");
        tDname.requestFocus () ;
        table_update();
    }
    catch (SQLException ex) {
        Logger. getLogger(name: JavaApplication1. class.getName ()).log (level: Level.SEVERE, msg: null, thrown: ex) ;
    }
}
```

Figure 11: UPDATE function for Department Table



```
private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
Integer Dnumber=Integer.parseInt(s: tDnum.getText());
String Dlocation=tDlocation.getText();
Connection c = db.getConnection();
try {
    pst=c.prepareStatement (string: "update dept_locations set Dlocation= ? where Dnumber= ?");
    pst.setString(i: 1, string: Dlocation);
    pst.setInt (i: 2, i1: Dnumber);
    pst.executeUpdate();
    JOptionPane.showMessageDialog(parentComponent: this, message: "Record Updated");
    tDnum.setText (t: "");
    tDlocation.setText (t: "");
    tDnum. requestFocus ();
    table2_update();
    }

catch (SQLException ex) {
   Logger. getLogger(name: JavaApplication1. class.getName()) .log (level: Level. SEVERE, msg: null, thrown: ex) ;
}
    }
```

Figure 12: UPDATE function for Dept_location Table



DEPARTMENT

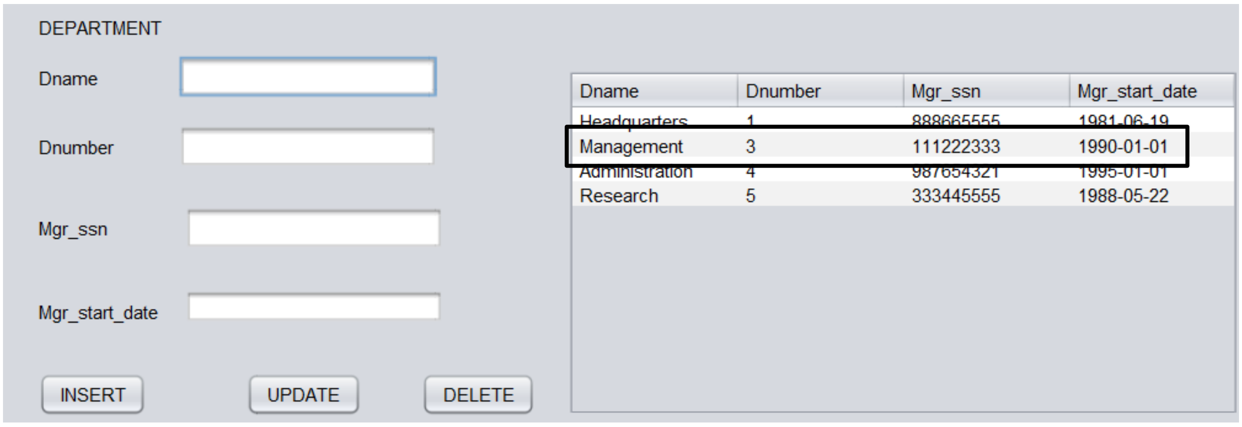
| | |
|---|---|
| Dname | Management |
| Dnumber | 3 |
| Mgr_ssn | 111222333 |
| Mgr_start_date | 1990-01-01 |

| Dname | Dnumber | Mgr_ssn | Mgr_start_date |
|---|---|---|---|
| Headquarters | 1 | 888665555 | 1981-06-19 |
| HR | 3 | 111222333 | 1990-01-01 |
| Administration | 4 | 987654321 | 1995-01-01 |
| Research | 5 | 333445555 | 1988-05-22 |

Message

Record Updated

Figure 13: Update Dname of the selected record Dnumber = 3 in Department Table



DEPARTMENT

| | |
|---|---|
| Dname | |
| Dnumber | |
| Mgr_ssn | |
| Mgr_start_date | |

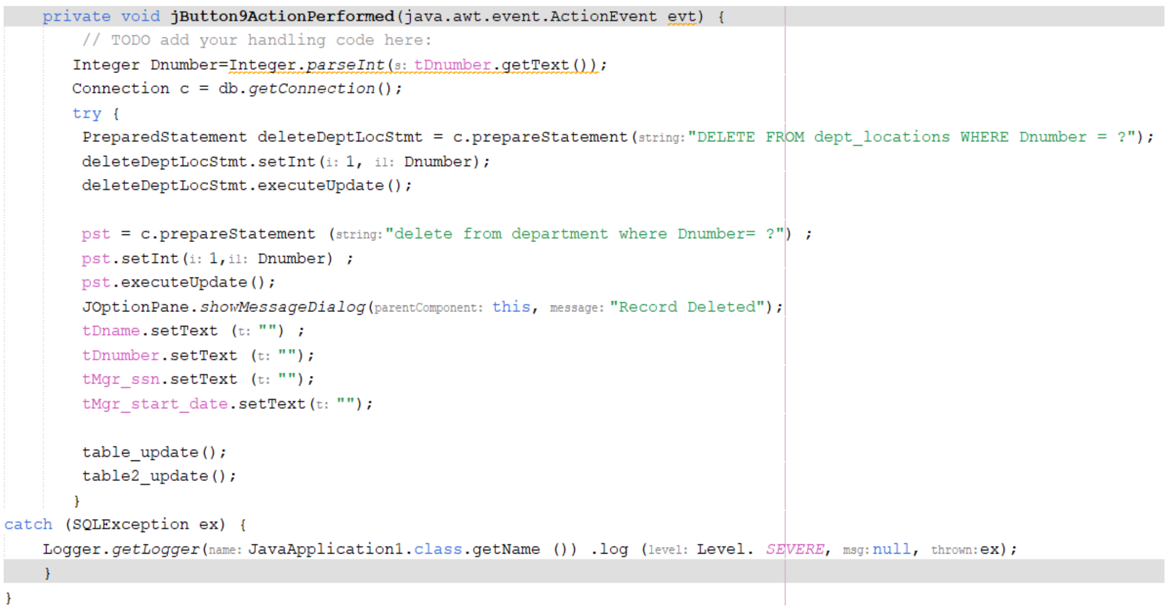
| Dname | Dnumber | Mgr_ssn | Mgr_start_date |
|---|---|---|---|
| Headquarters | 1 | 888665555 | 1981-06-19 |
| Management | 3 | 111222333 | 1990-01-01 |
| Administration | 4 | 987654321 | 1995-01-01 |
| Research | 5 | 333445555 | 1988-05-22 |

Figure 14: The record was updated successfully

The same process can be done to the Dept_location Table and update a relevant record as required.

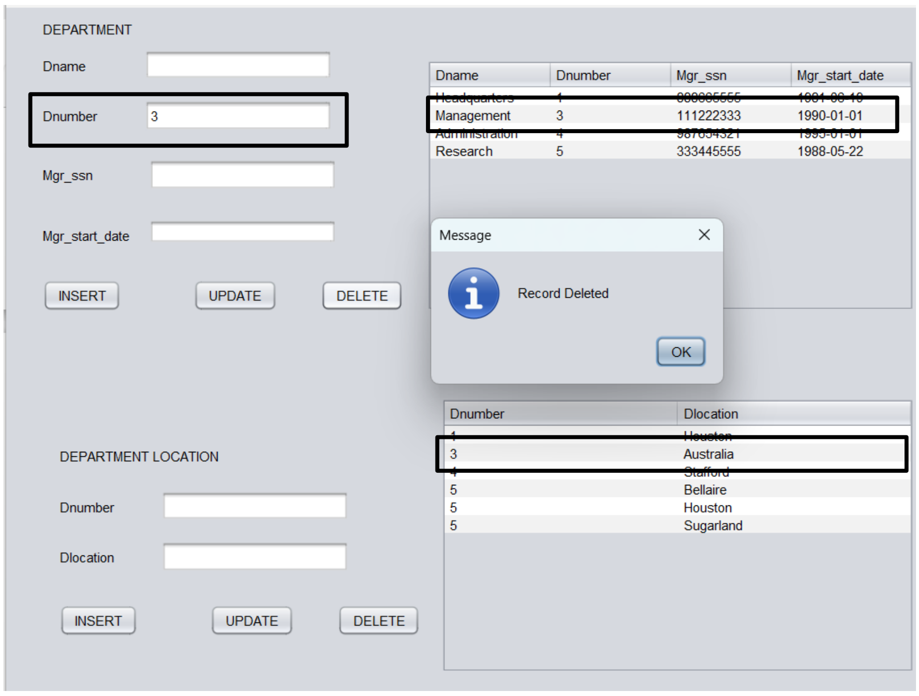
**DELETE**

As shown in Figure 15, the record to be deleted is identical by its Primary Key which is the ‘Dnumber’. Then in the UI a confirmation message will be displayed. By clicking Ok the relevant can be deleted successfully. If the record I related to the Dept_location table the relevant records in the Dept_location table which has the same Dnumber will also be deleted along with it. But when deleting a record from the Dept_location table it won’t delete the relevant records in the Department table as Department is the parent Table. Except that the Deletion function is similar.



```
    private void jButton9ActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
       Integer Dnumber=Integer.parseInt(s: tDnumber.getText());
       Connection c = db.getConnection();
       try {
        PreparedStatement deleteDeptLocStmt = c.prepareStatement(string:"DELETE FROM dept_locations WHERE Dnumber = ?");
        deleteDeptLocStmt.setInt(i: 1, i1: Dnumber);
        deleteDeptLocStmt.executeUpdate();

        pst = c.prepareStatement (string:"delete from department where Dnumber= ?") ;
        pst.setInt(i: 1,i1: Dnumber) ;
        pst.executeUpdate();
        JOptionPane.showMessageDialog(parentComponent: this, message: "Record Deleted");
        tDname.setText (t: "") ;
        tDnumber.setText (t: "");
        tMgr_ssn.setText (t: "");
        tMgr_start_date.setText(t: "");

        table_update();
        table2_update();
       }
catch (SQLException ex) {
    Logger.getLogger(name: JavaApplication1.class.getName ()) .log (level: Level. SEVERE, msg:null, thrown:ex);
    }
}
```

Figure 15: DELETE function for Department Table



DEPARTMENT

| | |
|---|---|
| Dname | |
| Dnumber | 3 |
| Mgr_ssn | |
| Mgr_start_date | |

| Dname | Dnumber | Mgr_ssn | Mgr_start_date |
|---|---|---|---|
| Headquarters | 1 | 888665555 | 1981-06-19 |
| Management | 3 | 111222333 | 1990-01-01 |
| Administration | 4 | 987654321 | 1995-01-01 |
| Research | 5 | 333445555 | 1988-05-22 |

Message

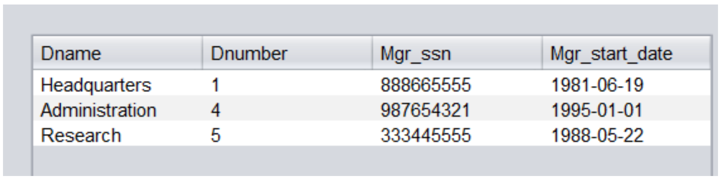
Record Deleted

DEPARTMENT LOCATION

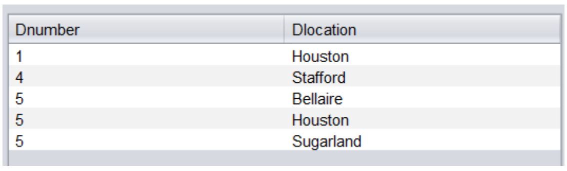
| | |
|---|---|
| Dnumber | |
| Dlocation | |

| Dnumber | Dlocation |
|---|---|
| 1 | Houston |
| 3 | Australia |
| 4 | Stafford |
| 5 | Bellaire |
| 5 | Houston |
| 5 | Sugarland |

Figure 16: Enter the Dnum to delete the relevant record in Department Table



| Dname | Dnumber | Mgr_ssn | Mgr_start_date |
|---|---|---|---|
| Headquarters | 1 | 888665555 | 1981-06-19 |
| Administration | 4 | 987654321 | 1995-01-01 |
| Research | 5 | 333445555 | 1988-05-22 |



| Dnumber | Dlocation |
|---|---|
| 1 | Houston |
| 4 | Stafford |
| 5 | Bellaire |
| 5 | Houston |
| 5 | Sugarland |

Figure 17: Department Table after Delete the record Figure 18: Dept_location after Delete

The record has been removed from both tables as they are related.

**Avoid Impedance Mismatch during the implementation**

Impedance mismatch occurs due to the difference between the database model and the programming language model. Mainly there are two cases that results an impedance mismatch.

1. When the data types of the programming language differ from the attribute data types available in the data model.

   This can be prevented by having a binding for each host programming language that specifies the data types for each attribute type in data model that is compatible with the programming language types. The compatible data types for Java used in this process are shown in Tables 01 and 02.

2. In most of the queries, the outcomes consists of sets of tuples, and each tuple contains a series of attributes.

   To prevent this mismatches during the process, a while loop was employed to iterate through the tuples in the query result which allows to access a single tuple at a time and a for loop was used to extract individual attribute values from the tuple..