

ASSIGNMENT-02

NAME : JATHURSHAN P.
INDEX NO : 19/ENG/041
REGISTRATION NO : EN93827
SUBMISSION DATE : 20/06/2022

First a database 'ABCD' is created after logging into MySQL. Then by selecting the database, the tables DEPARTMENT and DEPT_LOCATIONS were created, and the values were added into it. For the DEPARTMENT table the primary key is Dnumber and for the DEPT_LOCATIONS table the primary keys are Dlocation, Dnumber and the foreign key is Dnumber.

```

assignment2 - Notepad
File Edit View

mysql> CREATE DATABASE ABCD;
Query OK, 1 row affected (0.33 sec)

mysql> USE ABCD;
Database changed
mysql> 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 (0.48 sec)

mysql> 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 (0.45 sec)

mysql> DESCRIBE DEPARTMENT;
+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| Dname | varchar(20) | NO | | NULL | |
| Dnumber | int(5) | NO | PRI | NULL | |
| Mgr_ssn | int(50) | NO | | NULL | |
| Mgr_start_date | date | NO | | NULL | |
+-----+-----+-----+-----+-----+-----+
4 rows in set (0.00 sec)

mysql> DESCRIBE DEPT_LOCATIONS;
+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| Dnumber | int(5) | NO | PRI | NULL | |
| Dlocation | varchar(20) | NO | PRI | NULL | |
+-----+-----+-----+-----+-----+-----+
2 rows in set (0.00 sec)

mysql> INSERT INTO DEPARTMENT VALUES
-> ('Research',5,333445555,'1998-05-22'),
-> ('Administration',4,987654321,'1995-01-01'),
-> ('Headquarters',1,888665555,'1981-06-19');
Query OK, 3 rows affected (0.00 sec)
Records: 3 Duplicates: 0 Warnings: 0

mysql> INSERT INTO DEPT_LOCATIONS VALUES
-> (1,'Houston'),
-> (4,'Stafford'),
-> (5,'Bellaire'),
-> (5,'Houston'),
-> (5,'Sugarland');
Query OK, 5 rows affected (0.00 sec)
Records: 5 Duplicates: 0 Warnings: 0

mysql> SELECT *FROM DEPARTMENT;
+-----+-----+-----+-----+
| Dname | Dnumber | Mgr_ssn | Mgr_start_date |
+-----+-----+-----+-----+
| Research | 5 | 333445555 | 1998-05-22 |
| Administration | 4 | 987654321 | 1995-01-01 |
| Headquarters | 1 | 888665555 | 1981-06-19 |
+-----+-----+-----+-----+
3 rows in set (0.30 sec)

mysql> SELECT *FROM DEPT_LOCATIONS;
+-----+-----+
| Dnumber | Dlocation |
+-----+-----+
| 1 | Houston |
| 4 | Stafford |
| 5 | Bellaire |
| 5 | Houston |
| 5 | Sugarland |
+-----+-----+
5 rows in set (0.09 sec)

```

Figure-01: Creating database, creating tables inside it, put values inside tables, then display the tables

Table-01: Data types of given dataset

Attribute	Data types (in SQL)	Data types (in Java)
Dname	varchar	String
Dnumber	integer	Integer
Mgr_ssn	integer	Integer
Mgr_start_date	date	String
Dlocation	varchar	String

GUI implementation was used here to get the input via the text fields (in JFrame) as String data type and Integer.parseInt(String variable) was used to convert to integer value.

Here in java, java.sql.Date datatype can be used for the Mgr_start_date, but here String is used for Mgr_start_date because I used SQL exception handling (so no need of separate date datatype).

Libraries required for implementation [1]

```
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-02: Libraries used

Sql connection : class used for connection with a particular database. Within the context of a connection, SQL statements are run and results are returned. The database of a connection object can offer information about its tables, supported SQL syntax, stored procedures, etc.

Sql DriverManager: class has fundamental service for managing a collection of JDBC drivers. Method getConnection() will try to connect to the specified database URL.

Ex: con = DriverManager.getConnection("jdbc:mysql://localhost:3306/ABCD","root", "");

Sql PreparedStatement: PreparedStatement object stores a precompiled SQL statement. This object can be used to execute this statement several times. Should use the setObject method with target SQL type, if parameter type conversions are needed.

Ex: PreparedStatement p = con.prepareStatement(" UPDTAE DEPARTMENT SET Dname = ?
WHERE Dnumber = ?");

```
p.setString(1, Dname);
p.setInt(2, Integer.parseInt(Dnumber));
```

,where con – active connection, Dname,Dnumber – String values got from input fields.

Sql ResultSet: data table that provides a database result set and is often created by executing a statement that queries the database. The ResultSet object keep a cursor that points to the current row of data, next move that cursor to next row. Method next, returns false when there are no rows in object. So, can iterate through result set using while loop.

Sql Exception: Exception that gives informations about database access error or other issues. We can catch these exceptions when calling other class methods and can display them through dialog box.

We can use import java.sql.*; instead of the above 5 libraries (import java.sql.*; will import all the libraries related to sql).

JOptionPane: with this class it is easy to pop up a dialog box that asks users for a value or informs them of something like information messages or error messages. In the java implementation JOptionPane is used to pop up error messages and insert, update, delete success messages.

DefaultTableModel: TableModel implementation (uses vector of vectors) that stores the cell value objects. In the java implementation it is used to show and clear the table data.

Connector required

mysql-connector-java-8.0.29 is used - database driver help to access the MySQL databases from the applications created with Java. For getting information from database tables and performing other transactions, the connector use a JDBC driver. Cannot work with a MySQL database inside a java app without using this connector.

Here, Netbeans IDE was used and the project developed as a Java with Maven and the connector java-8.0.29 is added under dependencies section. Also GUI implementation is made here.

The image displays two screenshots of a Java GUI application. The top screenshot shows a window titled 'Table 1' with a 'Table 2' button in the top right. It contains four text input fields labeled 'Dname', 'Dnumber', 'Mgr_ssn', and 'Mgr_start_date'. Below these fields are four buttons: 'Insert', 'Update', 'Delete', and 'Search by Dnumber'. A table is displayed on the right side of the window with the following data:

Dname	Dnumber	Mgr_ssn	Mgr_start_date
Research	5	333445555	1998-05-22
Administration	4	987654321	1995-01-01
Headquarters	1	888665555	1981-06-19

The bottom screenshot shows a window titled 'Table 2' with a 'Table 1' button in the top right. It contains four text input fields: 'Dnumber', 'New Dnumber', 'Dlocation', and 'New Dlocation'. Below these fields are five buttons: 'Insert', 'Search by Dlocation', 'Update', 'Delete', and 'Search by Dnumber'. A note '(use above fields for updation only)' is present. A table is displayed on the right side of the window with the following data:

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

Figure-03: Output of the java implelmentation (GUI implemenation is made with Java JFrame form)

According to the above output, user can choose a particular table by clicking the button at right side up corner (ex: when clicking Table 2 button current tab closes and Table 2 tab will open). Before the insert, search, update, delete operations first let's see how the database is connected using JDBC driver, because that is needed (used) under every operation (insert, search, update, delete).

```
// For Database connection - using JDBC driver (included in the dependencies)
private Connection DBconnection() {

    try {
        con = DriverManager.getConnection("jdbc:mysql://localhost:3306/ABCD","root", "");

        if(con != null){
            System.out.println("Connected to DB successfully");
            return con;
        }
        else{
            System.out.println("Not Connected to DB");
            return null;
        }
    }
    catch (SQLException e) {
        JOptionPane.showMessageDialog(null, e, "SQL connection error", JOptionPane.ERROR_MESSAGE);
        return null;
    }
}
```

Figure-04: Method DBconnection() – DB connection using JDBC driver

JDBC driver enables a java app to interact with a database and enables transaction between front and backend. Using this we can connect to the localhost TCP port that contains the specific database. The method getConnection() connects to the database taking 3 parameters database URL ('jdbc:mysql://localhost:3306/ABCD'), username ('root'), and password(''). If there any connection errors to the database, it will be shown in the dialog box (JOptionPane) using SQLException class.

```
*/
public class Table1JFrame extends javax.swing.JFrame {

    Connection con = null;
    PreparedStatement p = null;
    ResultSet rs = null;
    String sql;
```

Figure-05: Declaration of common variables (to be used in the operations)

There are 2 Java JFrame forms for each table.

Implementation of Search operation

Here in this method for DEPARTMENT table, the details of department will be searched and shown in appropriate text fields according to the given 'Dnumber'. So user need to enter the Dnumber then need to press 'Search by Dnumber' button. Similarly for DEPT_LOCATIONS table there are 2 options available as 'Search by Dlocation' and 'Search by Dnumber' buttons.

Ex: If user click 'Search by Dlocation', he need to enter the Dlocation to get the details. Similarly for the 'Search by Dnumber' button.

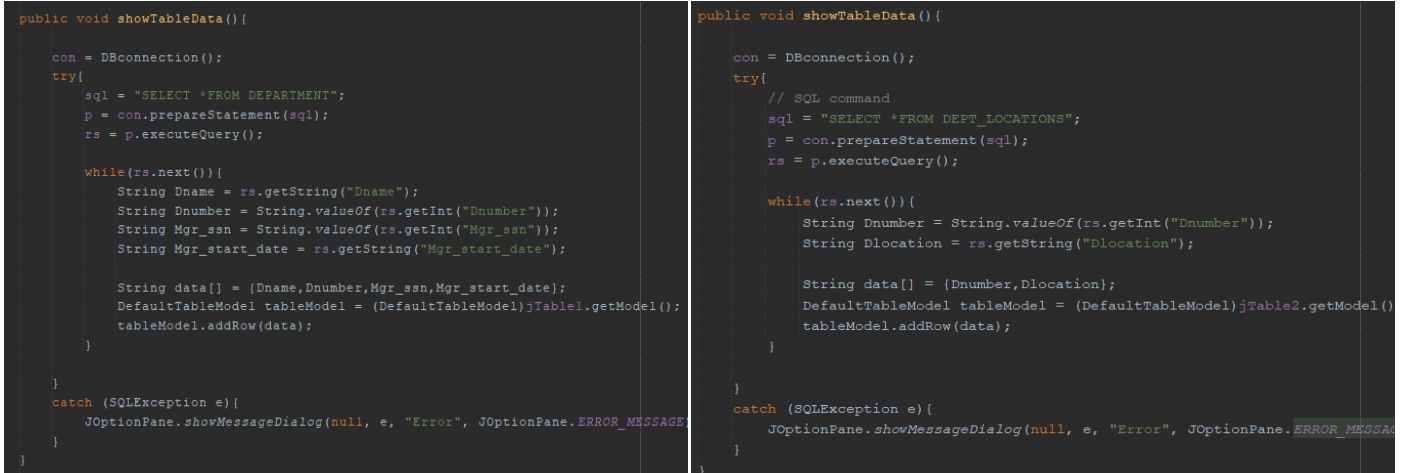
```
private void jButtonSearchActionPerformed(java.awt.event.ActionEvent evt) {  
    // TODO add your handling code here:  
    String Dnumber = jTextFieldDnumber.getText();  
    //check whether the text field is empty or not  
    if(Dnumber.equals("")){  
        JOptionPane.showMessageDialog(null, "Dnumber required to search the data",  
            "Error",JOptionPane.ERROR_MESSAGE);  
    }  
    else{  
        con = DBconnection();  
  
        try{  
            //SQL command  
            sql = "SELECT *FROM DEPARTMENT WHERE Dnumber = ?";  
            p = con.prepareStatement(sql);  
            p.setString(1,Dnumber);  
            rs = p.executeQuery();  
  
            String Dname = ""; // To check a mismatching Dnumber  
            while(rs.next()){  
                // get data from the result object  
                Dname = rs.getString("Dname");  
                String Mgr_ssn = rs.getString("Mgr_ssn");  
                String Mgr_start_date = rs.getString("Mgr_start_date");  
  
                // then set the data to appropriate text fields  
                jTextFieldDname.setText(Dname);  
                jTextFieldMgr_ssn.setText(Mgr_ssn);  
                jTextFieldMgr_start_date.setText(Mgr_start_date);  
            }  
  
            // if no matching data found  
            //- clear textfields and show error msg  
            if("").equals(Dname)){  
                jTextFieldDname.setText("");  
                jTextFieldMgr_ssn.setText("");  
                jTextFieldMgr_start_date.setText("");  
  
                JOptionPane.showMessageDialog(null, "No data found","Error",  
                    JOptionPane.ERROR_MESSAGE);  
            }  
        }catch(SQLException e){  
            JOptionPane.showMessageDialog(null, e,"Error",JOptionPane.ERROR_MESSAGE);  
        }  
    }  
}
```

Figure-06: jButtonDlocationSearchActionPerformed() method for DEPARTMENT table

'Dnumber' value should not be empty, so that is checked first and if it is empty an error dialog box is shown. If it is not empty DB connection() method is called and creates a database connection. Then the SQL command is created and converted to PreparedStatement object using Connection object. Then setString() method is used because parameter type conversions ("Dnumber = ?") are needed. Then the query result is gathered into ResultSet object and then by iterating through the result object by a while loop using next method the text fields will be filled with appropriate details. If no matching data found on the table (when iterating through) the text fields will be cleared and error dialog will pop up. The above tasks (after empty checking) are surrounded by try catch block to catch any sql database related errors or other errors using SQLException (used in all the operations -> insert,update,delete).

For the DEPT_LOCATIONS table there are two methods jButtonDlocationSearchActionPerformed() and jButtonDnumberSearchActionPerformed() for search by Dlocation and search for Dnumber. Both are similar to the above DEPARTMENT TABLE search method. Only thing change will be the SQL command and text fields.

Ex: search by Dnumber -> SELECT *FROM DEPT_LOCATIONS WHERE Dnumber = ?



```

public void showTableData() {
    con = DBconnection();
    try{
        sql = "SELECT *FROM DEPARTMENT";
        p = con.prepareStatement(sql);
        rs = p.executeQuery();

        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 e){
        JOptionPane.showMessageDialog(null, e, "Error", JOptionPane.ERROR_MESSAGE);
    }
}

public void showTableData() {
    con = DBconnection();
    try{
        // SQL command
        sql = "SELECT *FROM DEPT_LOCATIONS";
        p = con.prepareStatement(sql);
        rs = p.executeQuery();

        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 e){
        JOptionPane.showMessageDialog(null, e, "Error", JOptionPane.ERROR_MESSAGE);
    }
}

```

Figure-07:showTableData() method for DEPARTMENT and DEPT_LOCATIONS table

Here according to the figure-07 the sql command SELECT *FROM (table name) is used then it is converted to PreparedStatement object using Connection object. The query result is gathered into ResultSet object and then by iterating through the result object by a while loop using next method each time the row data is gathered into strings variables, then tableModel is created and the gathered row data will be added to the table.

Implementation of insert operation

Here in this method user has to enter valid details to the DEPARTMENT and DEPT_LOCATIONS tables and then click the insert button. All details need to be inserted (text fields cannot be null).

Sql command : INSERT INTO DEPARTMENT (attributs) VALUES (values)

See Figure-08,

Text fields here should not be empty (all the fields should be filled), so that is checked first and if it is empty an error dialog box is shown. If it is not empty DB connection() method is called and creates a database connection. Then clearing the data on table because after insertion there is a show table method which shows the data on table. If we don't clear the data after insertion of data, old data will combine with the data after insertion. Then the sql command is given. In the sql command the integers such as Dnumber, Mgr_ssn are need to converted to integers by the Integer.parseInt() method (because what we get from text field is string, ex: Dnumber as String). Then the SQL command with the relevant values is created as a string and then it is converted to a PreparedStatement object using the Connection object.

Next, insertion process will be executed using executeUpdate() method of PreparedStatement object. Above method of p will return the no of rows inserted. So, if no of inserted rows is greater than 0, show a success dialog message, table will be shown with inserted data using showTableData() method and inserted text fields will be cleared.

```
private void jButtonInsertActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    // Get the data from text fields
    String Dname = jTextFieldDname.getText();
    String Dnumber = jTextFieldDnumber.getText();
    String Mgr_ssn = jTextFieldMgr_ssn.getText();
    String Mgr_start_date = jTextFieldMgr_start_date.getText();

    //check whether the text fields are empty or not
    if(Dname.equals("") || Dnumber.equals("") || Mgr_ssn.equals("") || Mgr_start_date.equals("")){
        JOptionPane.showMessageDialog(null, "Fields cannot be null","Error",JOptionPane.ERROR_MESSAGE);
    }
    else{
        //Database connection
        con = DBConnection();
        //clear data on table
        jTable1.setModel(new DefaultTableModel(null, new String[]{"Dname","Dnumber","Mgr_ssn","Mgr_start_date"}));

        try{
            //SQL command
            sql = "INSERT INTO DEPARTMENT (Dname,Dnumber,Mgr_ssn,Mgr_start_date) VALUES ('"
                + Dname + "','" + Integer.parseInt(Dnumber) + "','" + Integer.parseInt(Mgr_ssn) + "','"
                + Mgr_start_date + "')";
            p = con.prepareStatement(sql);

            int insertedRows = p.executeUpdate();
            if(insertedRows > 0){
                JOptionPane.showMessageDialog(null, "Insertion successfull","success",JOptionPane.INFORMATION_MESSAGE);
                showTableData();
                //clearing the text fields after inserting
                jTextFieldDname.setText("");
                jTextFieldDnumber.setText("");
                jTextFieldMgr_ssn.setText("");
                jTextFieldMgr_start_date.setText("");
            }
        }catch(SQLException e){
            JOptionPane.showMessageDialog(null, e,"Error",JOptionPane.ERROR_MESSAGE);
        }
    }
}
```

Figure-08: jButtonInsertActionPerformed() method for DEPARTMENT table

Similarly for the DEPT_LOCATIONS table there is jButtonInsertActionPerformed() method which is similar to the above, other than the sql command and text fields (Figure-09).

```
private void jButtonInsertActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    // Get the data from text fields
    String Dnumber = jTextFieldDnumber.getText();
    String Dlocation = jTextFieldDlocation.getText();

    //check whether the text fields are empty or not
    if(Dnumber.equals("") || Dlocation.equals("")){
        JOptionPane.showMessageDialog(null, "Fields cannot be null","Error",JOptionPane.ERROR_MESSAGE);
    }
    else{
        //Database connection
        con = DBConnection();
        //clear data on table
        jTable2.setModel(new DefaultTableModel(null, new String[]{"Dnumber","Dlocation"}));

        try{
            //SQL command
            sql = "INSERT INTO DEPT_LOCATIONS (Dnumber,Dlocation) VALUES ('"
                + Integer.parseInt(Dnumber) + "','" + Dlocation + "')";
            p = con.prepareStatement(sql);

            int insertedRows = p.executeUpdate();
            if(insertedRows > 0){
                JOptionPane.showMessageDialog(null, "Insertion successfull","success",JOptionPane.INFORMATION_MESSAGE);
                showTableData(); // show table data
                //clearing the text fields after inserting
                jTextFieldDnumber.setText("");
                jTextFieldDlocation.setText("");
            }
        }catch(SQLException e){
            JOptionPane.showMessageDialog(null, e,"Error",JOptionPane.ERROR_MESSAGE);
        }
    }
}
```

Figure-09:
jButtonInsertActionPerformed()
method for DEPT_LOCATIONS
table

Implementation of update operation

```
private void jButtonUpdateActionPerformed(java.awt.event.ActionEvent evt) {  
    // TODO add your handling code here:  
    String Dname = jTextFieldDname.getText();  
    String Dnumber = jTextFieldDnumber.getText();  
    String Mgr_ssn = jTextFieldMgr_ssn.getText();  
    String Mgr_start_date = jTextFieldMgr_start_date.getText();  
  
    if(Dname.equals("") || Dnumber.equals("") || Mgr_ssn.equals("") || Mgr_start_date.equals("")){  
        JOptionPane.showMessageDialog(null, "Fields cannot be null", "Error", JOptionPane.ERROR_MESSAGE);  
    }  
    else{  
        con = DBConnection();  
        //clear data on table  
        jTable1.setModel(new DefaultTableModel(null, new String[]{"Dname", "Dnumber", "Mgr_ssn", "Mgr_start_date"}));  
  
        try{  
            //SQL command  
            sql = "UPDATE DEPARTMENT SET Dname = ?, Mgr_ssn = ?, Mgr_start_date = ? WHERE Dnumber = ?";  
            p = con.prepareStatement(sql);  
  
            p.setString(1, Dname);  
            p.setInt(2, Integer.parseInt(Mgr_ssn));  
            p.setString(3, Mgr_start_date);  
            p.setInt(4, Integer.parseInt(Dnumber));  
  
            int UpdatedRows = p.executeUpdate();  
            if(UpdatedRows > 0){  
                JOptionPane.showMessageDialog(null, "Update successfull", "success", JOptionPane.INFORMATION_MESSAGE);  
                showTableData();  
                //clearing the text fields after updating  
                jTextFieldDname.setText("");  
                jTextFieldDnumber.setText("");  
                jTextFieldMgr_ssn.setText("");  
                jTextFieldMgr_start_date.setText("");  
            }  
        } catch (SQLException e) {  
            JOptionPane.showMessageDialog(null, e, "Error", JOptionPane.ERROR_MESSAGE);  
        }  
    }  
}
```

Figure-10: jButtonUpdateActionPerformed() method for DEPARTMENT table

```
private void jButtonUpdateActionPerformed(java.awt.event.ActionEvent evt) {  
    // TODO add your handling code here:  
    String Dnumber = jTextFieldDnumber.getText();  
    String Dlocation = jTextFieldDlocation.getText();  
    String NewDnumber = jTextFieldNewDnumber.getText();  
    String NewDlocation = jTextFieldNewDlocation.getText();  
  
    if(Dlocation.equals("") || Dnumber.equals("") || NewDlocation.equals("") || NewDnumber.equals("")){  
        JOptionPane.showMessageDialog(null, "All four fields are required for updation", "Error", JOptionPane.ERROR_MESSAGE);  
    }  
    else{  
        con = DBConnection();  
        //clear data on table  
        jTable2.setModel(new DefaultTableModel(null, new String[]{"Dnumber", "Dlocation"}));  
  
        try{  
            //SQL command  
            sql = "UPDATE DEPT_LOCATIONS SET Dnumber = ? , Dlocation = ? WHERE Dnumber = ? AND Dlocation = ?";  
            p = con.prepareStatement(sql);  
  
            p.setInt(1, Integer.parseInt(NewDnumber));  
            p.setString(2, NewDlocation);  
            p.setInt(3, Integer.parseInt(Dnumber));  
            p.setString(4, Dlocation);  
  
            int UpdatedRows = p.executeUpdate();  
            if(UpdatedRows > 0){  
                JOptionPane.showMessageDialog(null, "Update successfull", "success", JOptionPane.INFORMATION_MESSAGE);  
                showTableData(); // show table data  
                //clearing the text fields after updating  
                jTextFieldDnumber.setText("");  
                jTextFieldDlocation.setText("");  
                jTextFieldNewDnumber.setText("");  
                jTextFieldNewDlocation.setText("");  
            }  
        }  
    }  
} catch (SQLException e) {  
}
```

Figure-10: jButtonUpdateActionPerformed() method for DEPT_LOCATIONS table

Sql command = UPDATE (table name) (variable = ?) WHERE (variable = ?)

All the field (inserted) values will be required, first that is checked. If all fields are filled then create data base connection. Then clear the data on table. After that inside the try catch block the following were done,

Created sql command, created sql string is converted to PreparedStatement object, then parameters of that object (p) is passed using setString() and setInt() methods appropriately. After that, updation is done using executeUpdate() method of PreparedStatement object p. It returns to an integer, the no of rows updated. If it is greater than the 0, inserted data will be updated based on the Dnumber for DEPARTMENT table. But in the DEPT_LOCATIONS table the inserted data will be updated based on the Dnumber and Dlocation. Here in the DEPT_LOCATIONS table there is small difference when doing updation process. That is, in DEPT_LOCATIONS table there are two text fields which are New Dnumber and New Dlocation, these values will be updated over the Dnumber and Dlocation (because there are two primary keys in the table– Dlocation,Dnumber)

```
sql = "UPDATE DEPT_LOCATIONS SET Dnumber = ? , Dlocation = ? WHERE Dnumber = ? AND Dlocation = ?";
```

```
p = con.prepareStatement(sql);
```

```
p.setInt(1, Integer.parseInt(NewDnumber)); //New Dnumber
```

```
p.setString(2,NewDlocation); //New Dlocation
```

```
p.setInt(3, Integer.parseInt(Dnumber));
```

```
p.setString(4,Dlocation);
```

Implementation of delete operation

```
private void jButtonDeleteActionPerformed(java.awt.event.ActionEvent evt) {  
    // TODO add your handling code here:  
    String Dnumber = jTextFieldDnumber.getText();  
    String Dlocation = jTextFieldDlocation.getText();  
  
    if(Dnumber.equals("") || Dlocation.equals("")){  
        JOptionPane.showMessageDialog(null, "Dlocation, Dnumber required","Error",JOptionPane.ERROR_MESSAGE);  
    }  
    else{  
        con = DBconnection();  
        //clear data on table  
        jTable2.setModel(new DefaultTableModel(null, new String[]{"Dnumber","Dlocation"}));  
  
        try{  
            //SQL command  
            sql = "DELETE FROM DEPT_LOCATIONS WHERE Dnumber = ? AND Dlocation = ?";  
            p = con.prepareStatement(sql);  
  
            p.setInt(1, Integer.parseInt(Dnumber));  
            p.setString(2,Dlocation);  
  
            int deletedRows = p.executeUpdate();  
            if(deletedRows > 0){  
                JOptionPane.showMessageDialog(null, "Deletion successful","success",JOptionPane.INFORMATION_MESSAGE);  
                showTableData(); // show table data  
                //clearing the text fields after inserting  
                jTextFieldDnumber.setText("");  
                jTextFieldDlocation.setText("");  
            }  
        }  
        catch(SQLException e){  
            JOptionPane.showMessageDialog(null, e,"Error",JOptionPane.ERROR_MESSAGE);  
        }  
    }  
}
```

Figure-11: jButtonDeleteActionPerformed() method for DEPT_LOCATIONS table

Here in this delete method for DEPT_LOCATIONS, because of two primary keys are there, such as Dnumber, Dlocation. The delete operation will take place when both text fields for Dnumber and Dlocation are filled. So, check the above condition and if both fields are not empty, then create the database connection using DBconnection() method, clear the table data. After that inside the try, catch block following were done,

Creation of the sql command which is DELETE FROM DEPT_LOCATIONS WHERE Dnumber = ? AND Dlocation = ?,

created sql string is converted to PreparedStatement object, then parameters of that object (p) is passed using setString() and setInt() methods appropriately. After that, deletion is done using executeUpdate() method of PreparedStatement object p. It returns to an integer, the no of rows deleted. If it is greater than the 0, data will be deleted based on the Dnumber and Dlocation. Also show the data by showTableData() and clear the text fields.

Similarly for the DEPARTMENT table inside the method jButtonDeleteActionPerformed(), based only on Dnumber the data are deleted. So here the only field needed is Dnumber to delete the data associated with it.(sql command: DELETE FROM DEPARTMENT WHERE Dnumber = ?)

How to avoid mismatch impedance

Mismatch impedance 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 database. For that most appropriate data types were used in java implementation. Ex: java – String, sql – varchar, also used Integer.parseInt(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");  
    .....  
}
```

References

[1]"Java Platform SE 7", *Docs.oracle.com*, 2022. [Online]. Available: <https://docs.oracle.com/javase/7/docs/api/index.html>. [Accessed: 19- Jun- 2022].