COS 106 – Application Programming with Java

Name : Chaw Thiri Win

Table of Contents

System Overview

For TV Repair Private Company, this system will be a Java Swing application that manages client records using a MySQL database (RepairDB). Users will be able to access, manage, and change user logins, TV product data, and customer information through the application. JDBC will be used to connect to the database and conduct activities such as retrieving customer data, altering product information, and searching for customer records. The application will have an easy user interface that will allow users to do a variety of operations such as adding new clients, changing existing ones, deleting entries, and retrieving data based on specified criteria. By connecting to the database, the system will effectively administer and save client data, therefore helping the company's growth ambitions.

ER Diagrams

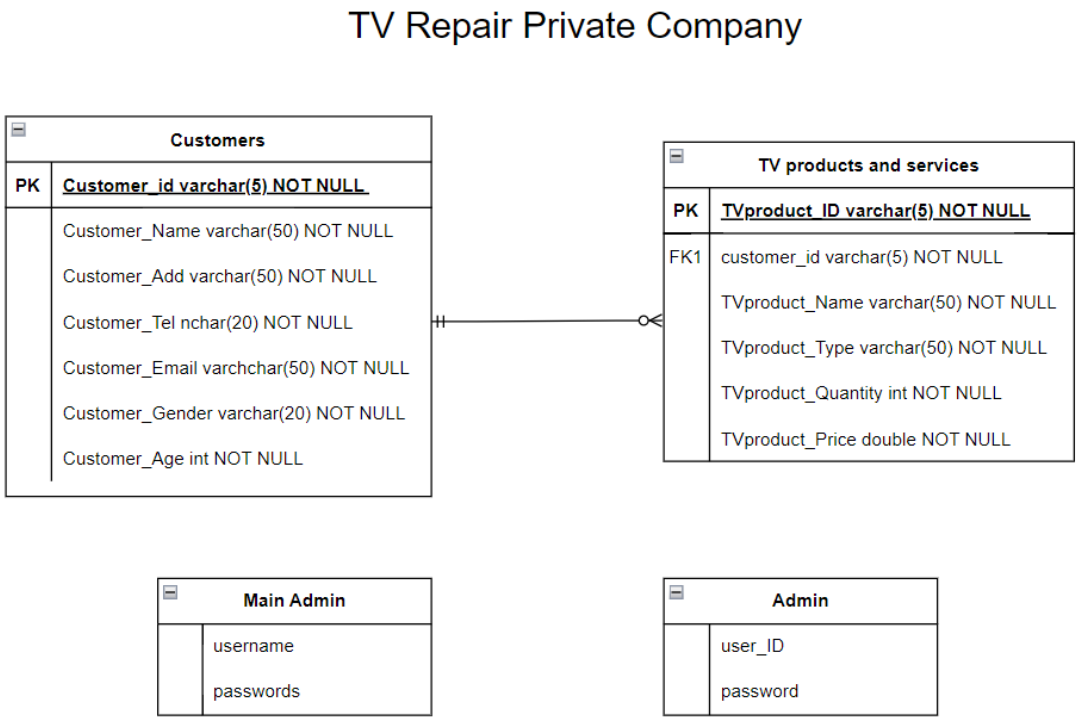

Figure: Entity Relationship of TV Repair Private Company

Output of Frames

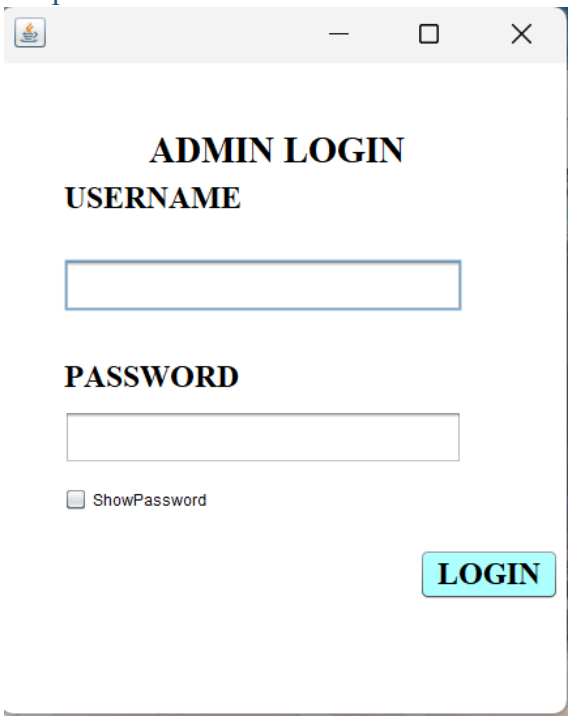

Figure 1 – Admin Login Form

In this form user(admin) have to add username and password to log in.

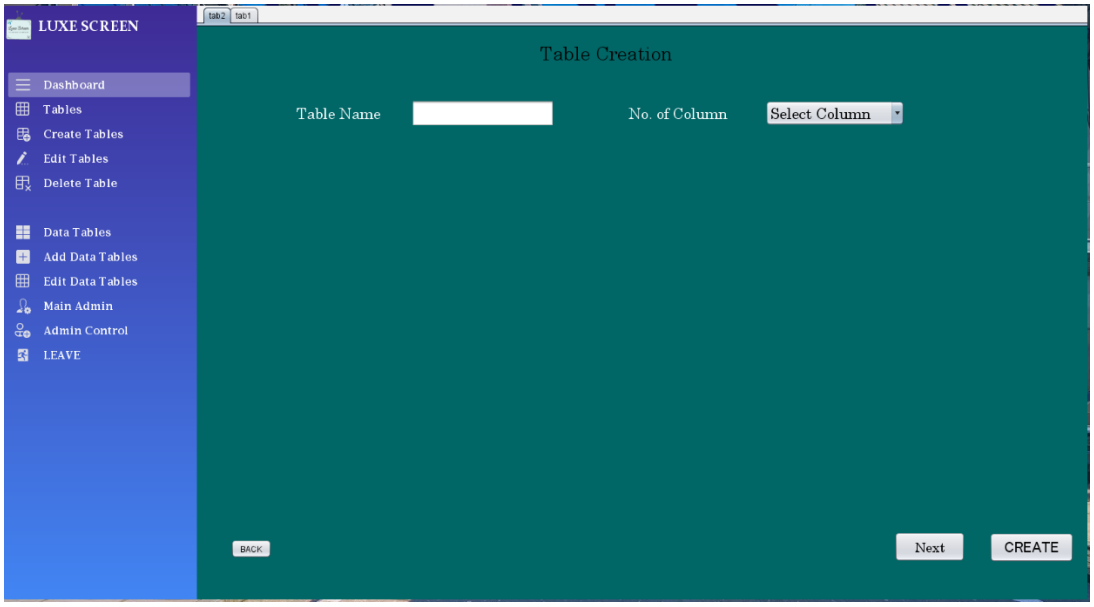

Figure 2 – Dashboard Form

After log in the user would reach through this main menu form. When the user write the table name and select the columns the different types of column will appear.

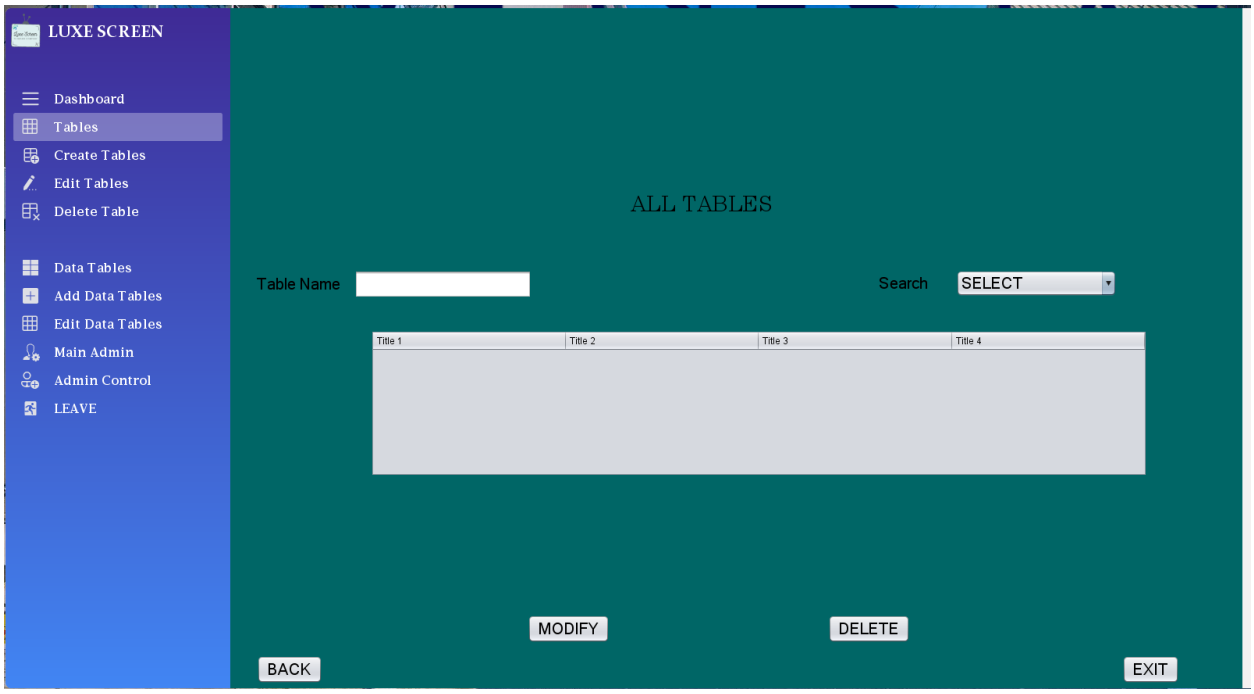

Figure 3 – All Tables Form

This is all table form and here user can search the table name and also select it from combo box that modify and delete button will lead to other modification form and deletion form.

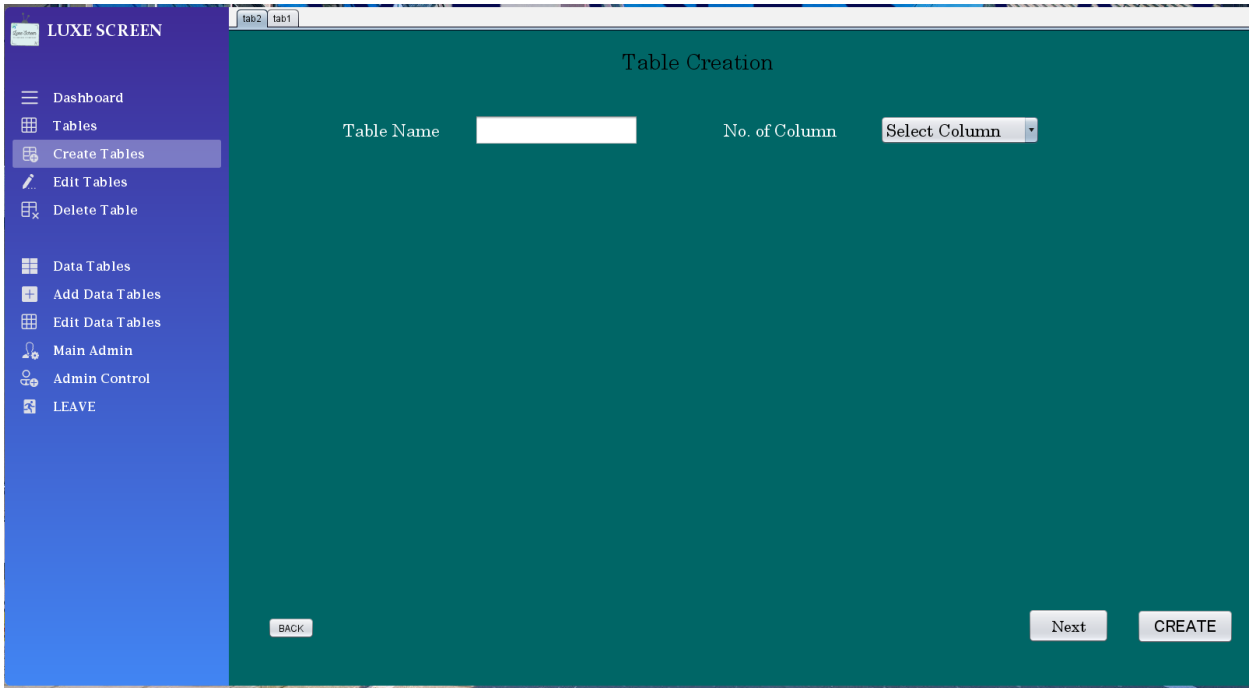

Figure 4 – Table Creation Form

In this form user will be able to create the form according to the selection of the columns there are four different categories to choose 5,10,15 and 20.

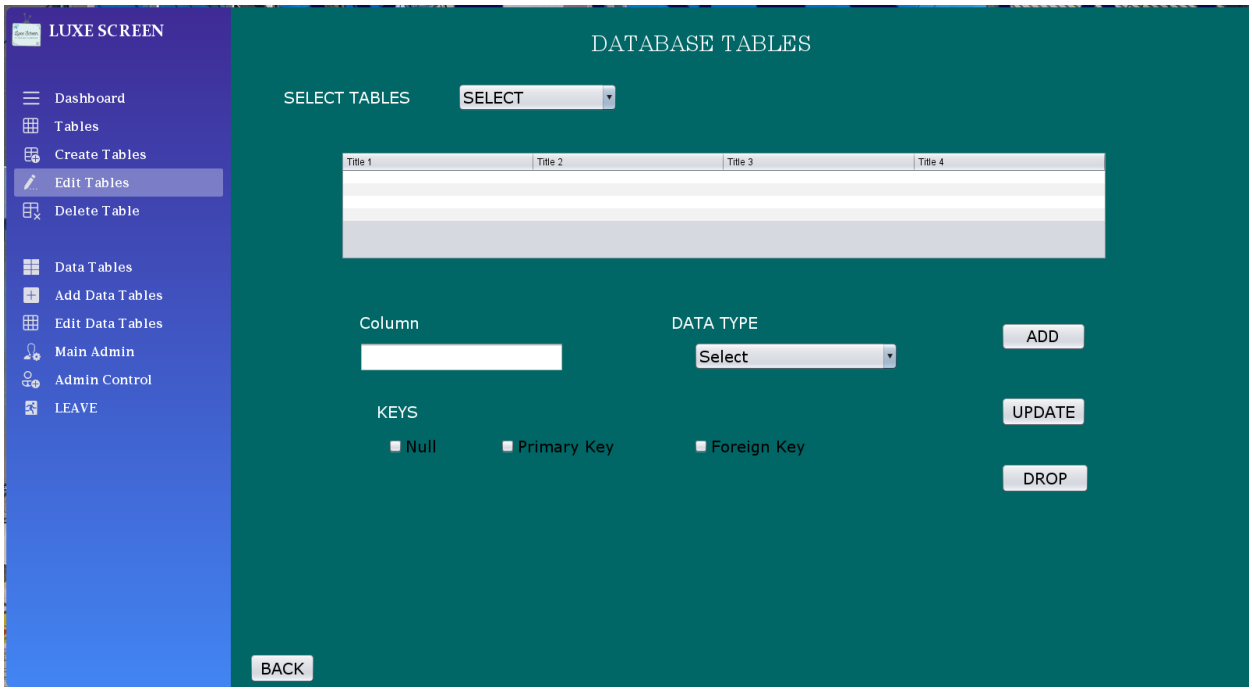

Figure 5 – Database Form

This is database edit form where User would be able to add,update or drop the columns.

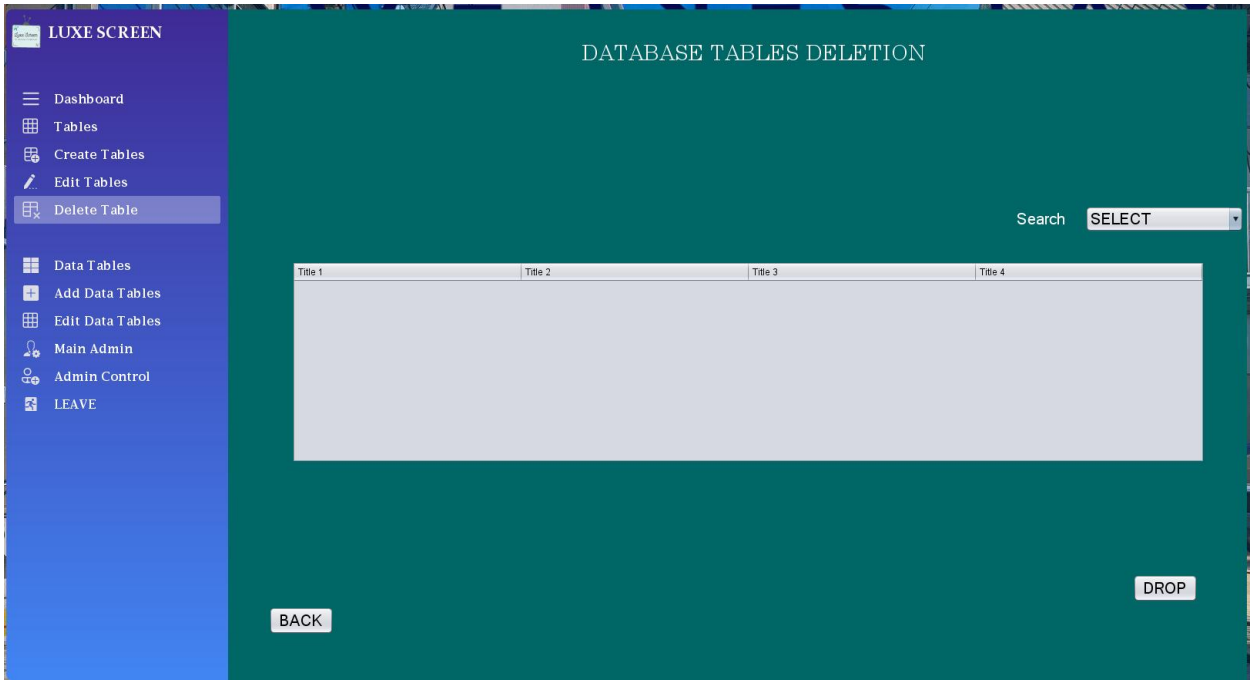

Figure 6 – Database Form

This is data deletion form where user would be able to drop that whole table. If it is built in table user wouldn't be able to drop it.

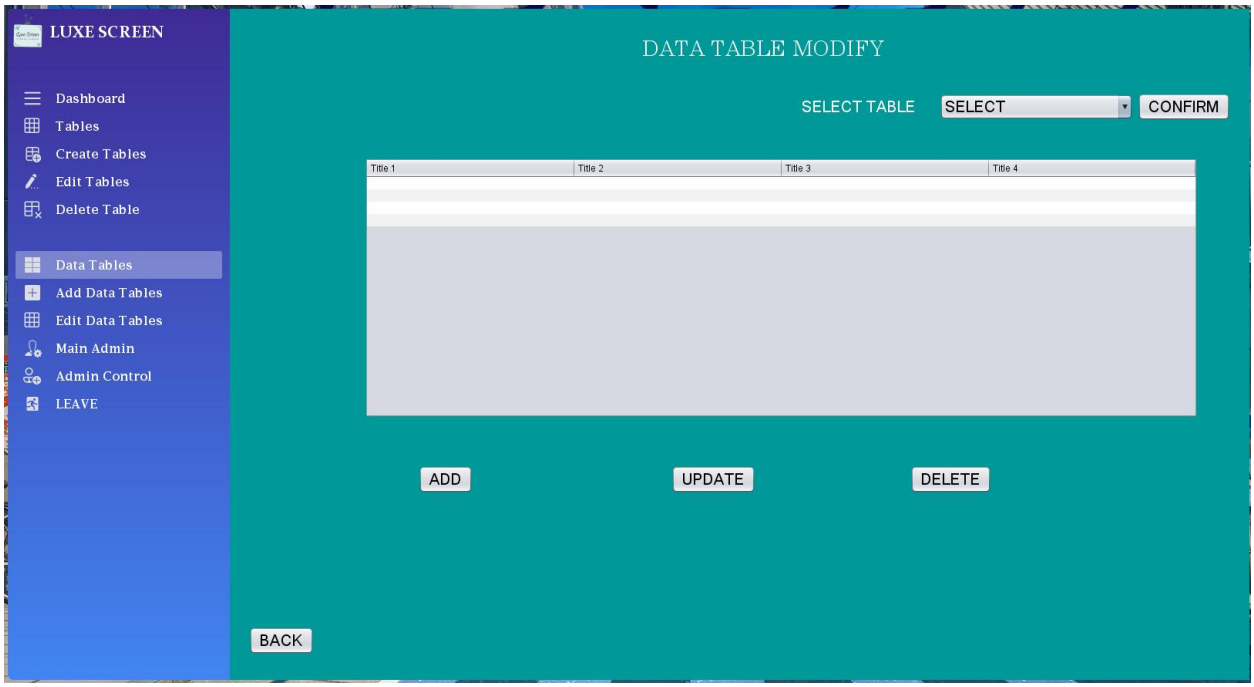

Figure 7 – Database Table Modify Form

This is data table modify main form where each button lead to different panels. The add button will lead to add data table for the update and delete buttons will lead to edit data form.

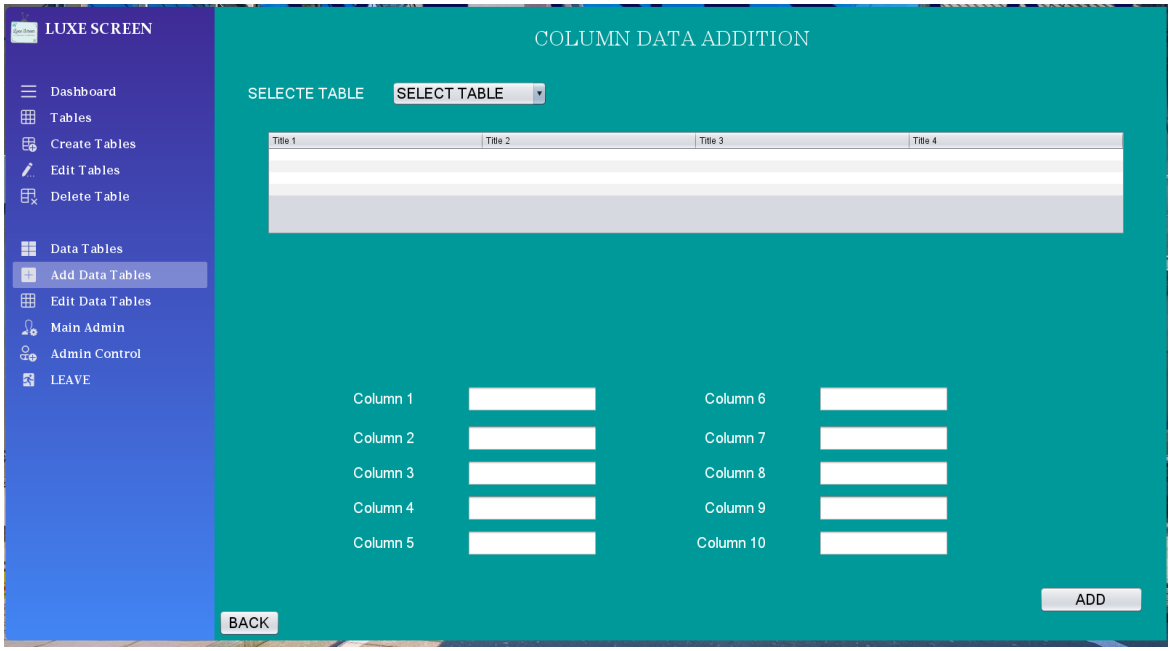

Figure 8 – Database Table Modify Form

This is data column addition form where user will be able to add data here. According to user chosen table name the column name would be change according to that table

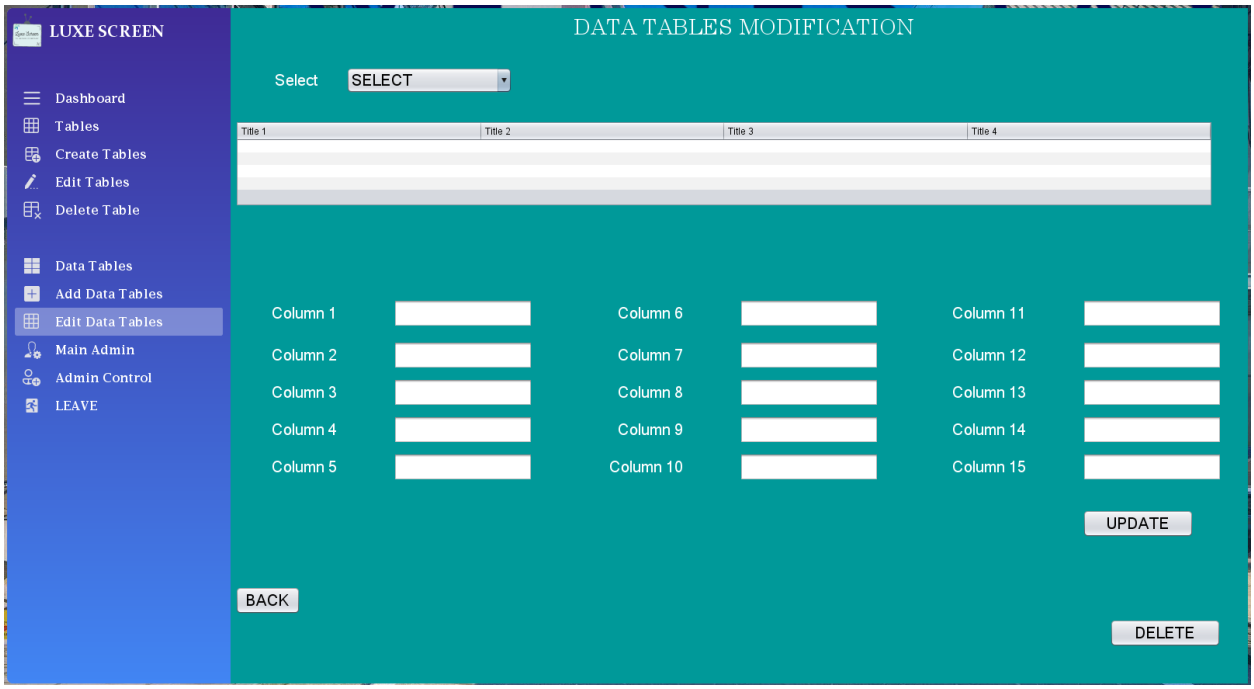

Figure 9 – Data Table Modification Form

This is data table modification for where user would be able to update or delete the data. And also here by selecting from the combobox, the column name will change according to that table name.

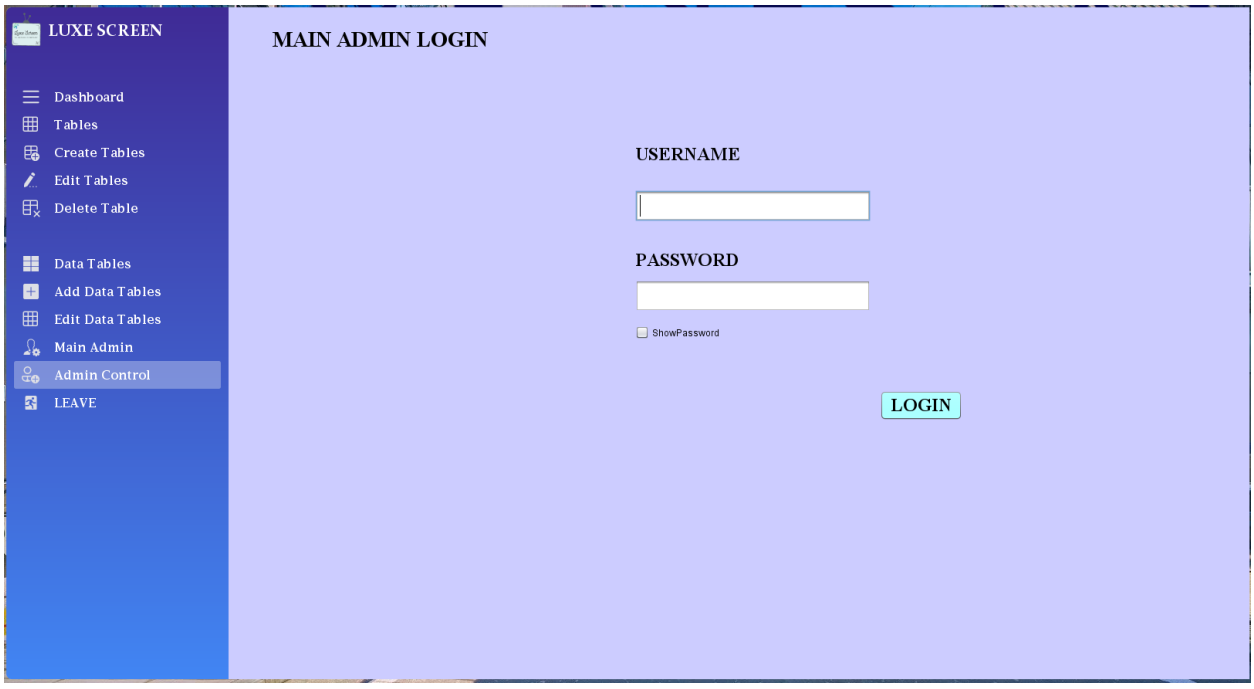

Figure 10 – Main Admin Login Form

This is main admin login form where main admin control admins.

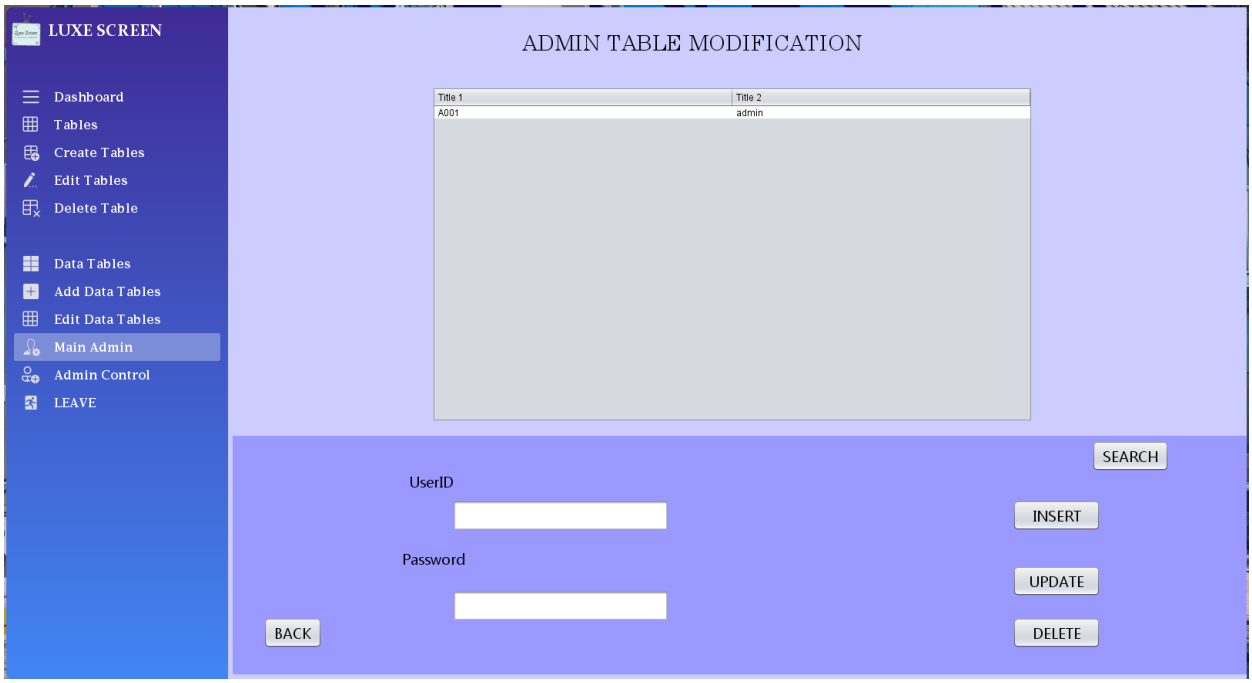

Figure 11 – Admin Table Modification Form

This is admin table modification form. This form can be only reach by main admin after they are login. Here main admin would be able to insert new admins or update data existing data or delete the data.

Coding

```
private void cboColumnItemStateChanged(ItemEvent evt) {
  if (evt.getStateChange() == ItemEvent.SELECTED) {
   // Get the selected number of columns
   String selectedItem = (String) cboColumn.getSelectedItem();
   int numberOfColumns;

   try {
       numberOfColumns = Integer.parseInt(selectedItem);
   } catch (NumberFormatException e) {
       JOptionPane.showMessageDialog(this, "Please Select the No. of Column");
       return;
   }
   // Loop through each array and set visibility based on the selected number of columns
   for (int i = 0; i < labels.length; i++) {
       boolean isVisible = i < numberOfColumns;
       labels[i].setVisible(isVisible);
       textFields[i].setVisible(isVisible);
       comboBoxes[i].setVisible(isVisible);
       chkN[i].setVisible(isVisible);
       chkP[i].setVisible(isVisible);
       chkF[i].setVisible(isVisible);
   }

   if (numberOfColumns > 10) {
       btnNext.setVisible(true);
       btnCreate.setVisible(false);
   } else {
       btnNext.setVisible(false);
       btnCreate.setVisible(true);
   }
}
}
```

Figure 1 - Combo Columns Item State Change

This function manages the dynamic display of UI components when the user specifies the number of columns for table construction. When a column count is selected, it runs through preconfigured arrays of labels, text fields, combo boxes, and checkboxes, displaying or concealing them based on the number of columns chosen. If more than ten columns are selected, it displays a "Next" button to move to the next tab; otherwise, it displays the "Create" button. This solution offers a flexible, user-friendly interface for building database tables with a variable number of columns, with the UI dynamically adapting to meet the user's column count.

```
// Construct SQL CREATE TABLE statement
StringBuilder sqlBuilder = new StringBuilder("CREATE TABLE " + tName + " (");

for (int i = 0; i < numColumns; i++) {
    sqlBuilder.append(columnData[i][0]).append(" ").append(columnData[i][1]).append(" ")
              .append(columnData[i][2]).append(" ").append(columnData[i][3]);
    if (i < numColumns - 1) {
        sqlBuilder.append(", "); // Add comma if not the last column
    }
}

sqlBuilder.append(")");
```

Figure 2 – Construct SQL Statement

This code snippet creates a StringBuilder to dynamically produce a SQL CREATE TABLE query. It iterates over the user-specified number of columns, attaching each column's information (name, data type, null constraint, and primary key status) to the SQL statement. The approach separates columns properly using commas and adds parenthesis at the beginning and conclusion of the statement. This solution enables the flexible and automated development of SQL table formation syntax depending on user input, allowing for tables with various column counts.

```
if(name.equals("") && pword.equals(""))
{
    JOptionPane.showMessageDialog(this,"Enter UserName and Password","Error",JOptionPane.ERROR_MESSAGE);
}
else if(count == 1)
{
    JOptionPane.showMessageDialog(this,"Login Success");
    //Form Change
    Main mm = new Main();
    this.setVisible(false);
    mm.setVisible(true);
    mm.setTitle("Main Menu");
}
else
{
    JOptionPane.showMessageDialog(this, "Incorrect UserName or Password");
    txtUsername.setText("");
    txtPassword.setText("");
}
```

Figure 3 – Login Validation

This code manages user login validation in a Java Swing application. It initially determines if the username (name) and password (pword) boxes are empty; if so, it shows an error message urging the user to input both credentials. If the login attempt is successful (as shown by count == 1), it displays a success message and takes the user to the main menu by creating a new instance of the Main class and concealing the existing login form. If the credentials are wrong, it notifies the user of the failure and clears the username and password input boxes, enabling them to try again. This solution offers a user-friendly experience by giving specific feedback based on the outcome of the login attempt.

```
// List of tables that cannot be deleted
String[] protectedTables = {"Customer", "admin", "mainadmin", "tv_products_and_services"};

// Check if the selected table is in the protected list
boolean isProtected = false;
for (String table : protectedTables) {
    if (table.equalsIgnoreCase(selectedTable)) {
        isProtected = true;
        break;
    }
}

if (isProtected) {
    JOptionPane.showMessageDialog(this, "You cannot delete the table: " + selectedTable);
    return; // Exit the method if the table is protected
}
```

Figure 4 – Protected Tables

This code snippet creates an array called protectedTables to prevent the deletion of essential tables in a database. It iterates over the array to check if the user-selected table is included, and if so, warns the user via JOptionPane, ensuring data integrity.

```
// Update Text Field Labels Method
private void updateTextFieldLabels(ResultSetMetaData metaData) throws SQLException {
    int columnCount = metaData.getColumnCount();

    // Reset labels and visibility
    JLabel[] labels = {
        jLabel3, jLabel4, jLabel5, jLabel6, jLabel7,
        jLabel8, jLabel9, jLabel10, jLabel11, jLabel12,
        jLabel13, jLabel14, jLabel15, jLabel16, jLabel17
    };

    // Hide all labels and text fields initially
    for (int i = 0; i < labels.length; i++) {
        textFields[i].setVisible(false);
        labels[i].setVisible(false);
    }

    // Show only existing columns
    for (int i = 0; i < Math.min(columnCount, labels.length); i++) {
        String columnName = metaData.getColumnName(i + 1);
        labels[i].setText(columnName);
        textFields[i].setVisible(true);
        labels[i].setVisible(true);
        textFields[i].setText(""); // Clear text fields
    }
}
```

Figure 5 – Update Text Labels

The above given code show that labels will change according to the data table. The updateTextFieldLabels function uses SQL result set metadata to change the visibility and labels of text fields; it loops over columns, displays only those in the result set, retrieves column numbers, initializes an array of JLabel objects, hides text fields and labels, labels column names, reveals related text fields, and clears empty text boxes.

```
// Check for duplicate ID
if (!primaryKeyValue.isEmpty()) {
    String duplicateCheckQuery = "SELECT COUNT(*) FROM " + selectedTable +
                                 " WHERE " + primaryKeyColumn + " = ?";
    PreparedStatement checkStmt = con.prepareStatement(duplicateCheckQuery);
    checkStmt.setString(1, primaryKeyValue);

    ResultSet rs = checkStmt.executeQuery();
    if (rs.next() && rs.getInt(1) > 0) {
        JOptionPane.showMessageDialog(this,
            "Error: " + primaryKeyColumn + " already exists",
            "Duplicate ID Entry",
            JOptionPane.ERROR_MESSAGE
        );
        return;
    }
}
```

Figure 6 – Duplicate ID checking

This code looks for duplication in a primary key value before adding a new record to a database table. It initially checks to see if a value for the main key has been entered, and if so, it generates a SQL query to find out how frequently that value occurs in the specified table. It uses a PreparedStatement to execute the query and sets the value of the primary key as a parameter. If the result shows that the primary key value already exists (count > 0), it exits from the process and uses JOptionPane to generate an error message. This ensures data integrity by removing duplicate primary key items from the data.

Error Validation

Figure – 1

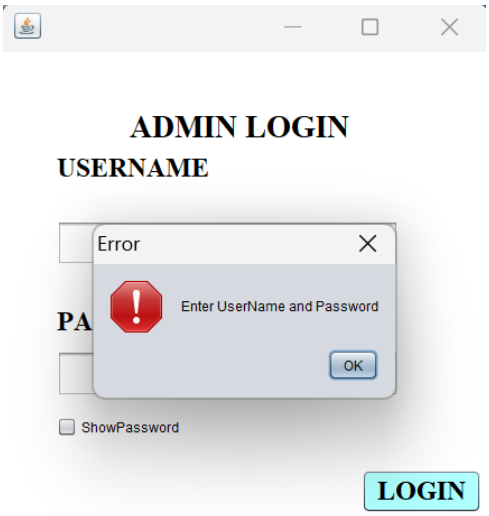

Figure – 2

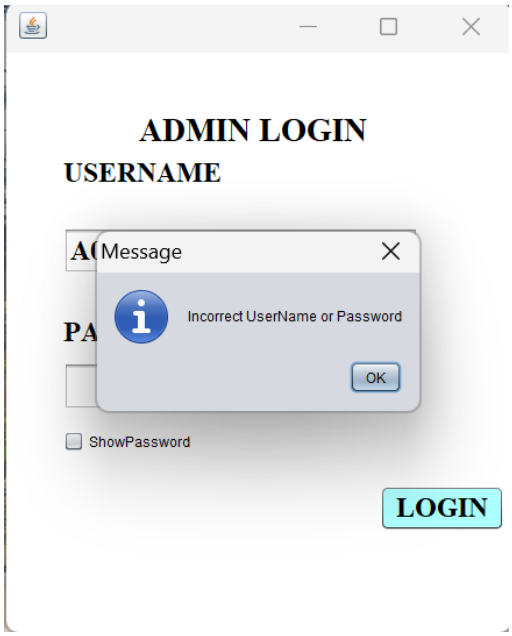

Figure – 3

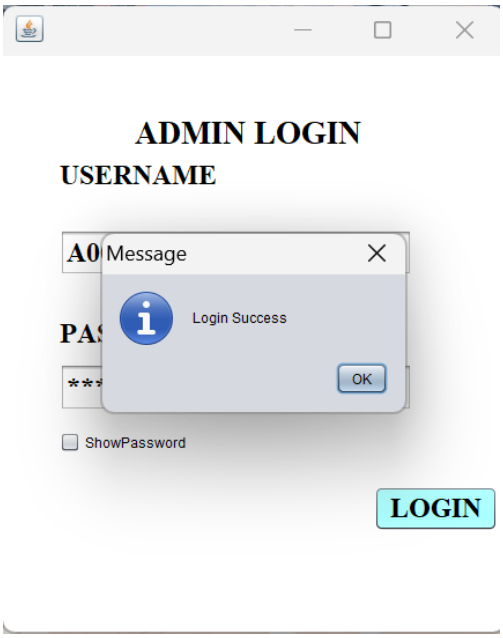

Figure – 4

According to this above pictures, user login validation in a Java Swing application. It checks if username and password boxes are empty, displays an error message, and prompts the user to input both credentials.

If successful, it creates a new instance of the Main class and hides the existing login form. If incorrect credentials are entered, it clears the input boxes, allowing the user to try again.