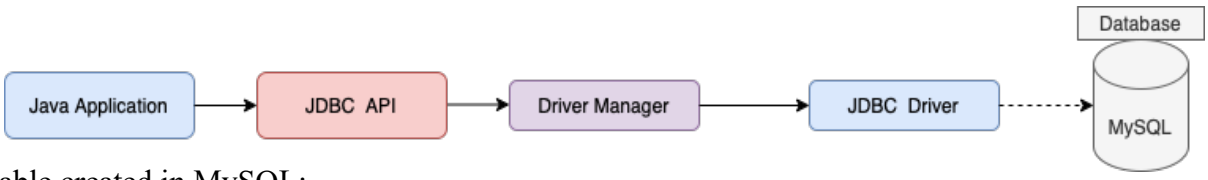
Criterion C: Development

Table of Contents

1. JDBC Connectivity, MySQL

JDBC (Java Database Connectivity) is an API (Application Programming Interface) that enables Java programs to interact with a database. It provides a standard interface for Java applications to connect with various databases, in our case (MySQL). The program permits the user to input access requests in Structured Query Language (SQL). The user can create SQL statements and submit them to the program essentially, users can perform different operations on the data contained in the database. Below is a diagram showing the connectivity from Java Application to MySQL database.

The diagram below exhibits the process of connecting Java Application to the MySQL Database:



Sample table created in MySQL:

ID	First	Last	Role	Age	Email	Password
1	Allan	Smith	Customer	24	A@gmail.com	123456
2	Sally	Karen	Employee	19	S@gmail.com	Sally123

This Java code establishes a connection to a MySQL database using the JDBC driver and the specified connection string. It also creates a statement object that can be used to execute SQL queries on the connected database.

Code to establish a connection from java NetBeans IDE to MySQL Data Base:

```
public class connectionz {
public static Connection con;
public static Statement stmt;
public static ResultSet rs;
public static void connect()
    {


        try {
            Class.forName("com.mysql.cj.jdbc.Driver");
            con= DriverManager.getConnection("jdbc:mysql://localhost:3306/mysql?zeroDateTimeBehavior=CONVERT_TO_NULL","root","Sami9870");
            stmt=con.createStatement();



        } catch (Exception e)
        {
         JOptionPane.showMessageDialog(null, e);
        }




    }
```

I created a MySQL Database table called ‘Employee1’ where all users who create or have an account has their details stored in the Employee1 table. The login code/ query below outlines the connection between java NetBeans IDE and MySQL database ‘Employee1’ table as well as identifies whether the Email, Password and Role inputted is correspondent to the values of the ‘Employee1’ table.

```
try {
    Class.forName("com.mysql.cj.jdbc.Driver");
   Connection con= DriverManager.getConnection("jdbc:mysql://localhost:3306/mysql?zeroDateTimeBehavior=CONVERT_TO_NULL","root","Sami9870");
   pst= con.prepareStatement("select * from Employee1 where Email=? and Password=? ");
   pst.setString (1, uname);
   pst.setString (2, pword);
   rs= pst.executeQuery();

   if (rs.next()) {
       String s1 = rs.getString("Role");
       String un= rs.getString("Email");
       if(Role.equalsIgnoreCase ("Admin") && s1.equalsIgnoreCase ("admin")) {
           AdminPage ad = new AdminPage(un);
           ad.setVisible(true);
           setVisible (false);

       }
         if(Role.equalsIgnoreCase ("Employee") && s1.equalsIgnoreCase ("employee")) {
           Menu me = new Menu(un);
           me.setVisible(true);
           setVisible (false);

         }

          if(Role.equalsIgnoreCase ("Customer") && s1.equalsIgnoreCase ("customer")) {
           CustomerPage cp = new CustomerPage(un);
           cp.setVisible(true);
           setVisible (false);

          }

   }
   else {
       JOptionPane.showMessageDialog(rootPane, "Email or Password Incorrect");
   }

}
catch (Exception ex) {
    System.out.print(""+ ex);
}
```

The ‘pst’ object is created to execute a parameterized SQL query to select records from the Employee1 table where the email and password match the values entered by the user.

rs is a ResultSet object. It is initialized with the result set of the SQL query executed in the try block using the executeQuery() method of the PreparedStatement object pst.
After the execution of the query, the rs.next() method is called to move the cursor of the ResultSet to the first row. Then, the rs.getString() method is used to retrieve the value of the specified column from the current row of the ResultSet. Finally, the retrieved value is compared with the selected value of the Role variable using the equalsIgnoreCase() method to determine the role of the user and to redirect the user to the appropriate page based on their role.

Note: The code uses encapsulation to hide the implementation details of the database connection and the PreparedStatement object. The instance variables pst and rs are declared as private, and their values are accessed using public getter methods.

The Register page, inserts new data/ accounts to the MySQL table ‘Employee1’ as the user creates an account. The data added to the database follows ID, First (name), Last (name), Role (user type), Age, Email, and Password. The source code below demonstrates this.

```
    String fname = jTextField1.getText();
    String lname = jTextField2.getText();
    String utype = jComboBox1.getSelectedItem().toString();
    String pas = String.valueOf(jPasswordField1.getPassword());
    String m= jTextField3.getText();
    String a= jTextField5.getText();
    try {
        Class.forName("com.mysql.cj.jdbc.Driver");
       con= DriverManager.getConnection("jdbc:mysql://localhost:3306/mysql?zeroDateTimeBehavior=CONVERT_TO_NULL","root","Sami9870");
       String sql= "SELECT MAX(ID) from Employee1";
       st = con.createStatement();
        rs= st.executeQuery(sql);
        if (rs.next()) {
            lastid= rs.getInt(1);
            lastid++;
        }
        if (verifyFields()) {
        String q= "INSERT INTO Employee1 (ID, First, Last, Role, Age, Email, Password) values ('"+lastid+"','" +fname+ "','"+lname+ "','"+utype+"','"+a+ "','"+m+"','"+pas+"')";
        pst = con.prepareStatement(q);
        pst.execute();
        JOptionPane.showMessageDialog(null, "Account Saved! Your ID is: "+lastid);
        Login l1= new Login();
        l1.setVisible(true);
        this.dispose();
        }
    } catch (Exception e)
    {
    JOptionPane.showMessageDialog((null), e);
}
```

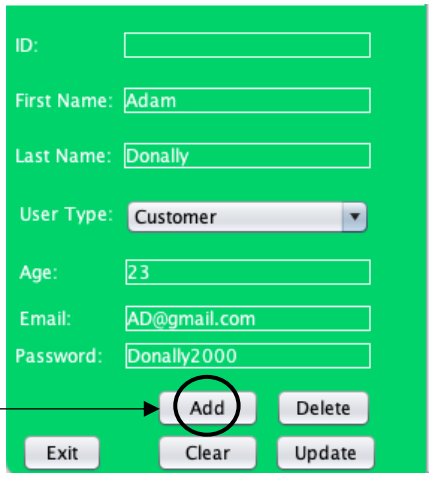
Retrieves values entered by the user for fields, executes a query to retrieve the maximum ID value from the Employee1 table. The lastid variable is set to this value plus one, which will be used to insert the new record with a unique ID.

Similarly, the admin user can also add accounts to the 'Employee1' database as well as delete and update the database.

Add:

```
    try {
        Class.forName("com.mysql.cj.jdbc.Driver");
       con= DriverManager.getConnection("jdbc:mysql://localhost:3306/mysql?zeroDateTimeBehavior=CONVERT_TO_NULL","root","Sami9870");
       String sql= "SELECT MAX(ID) from Employee1";
       st = con.createStatement();
        rs= st.executeQuery(sql);
        if (rs.next()) {
            lastid= rs.getInt(1);
            lastid++;
        }
        if (verifyFields()) {
        Statement st= con.createStatement();
        pst= con.prepareStatement("INSERT INTO Employee1"+"(ID, First, Last, Role, Age, Email, Password) values" +"(?,?,?,?,?,?,?)");
        pst.setInt(1, lastid);
        pst.setString(2, jTextField1.getText());
        pst.setString(3, jTextField5.getText());
        pst.setString(4, jComboBox1.getSelectedItem().toString());
        pst.setString(5, jTextField3.getText());
        pst.setString(6, jTextField2.getText());
        pst.setString(7, jTextField4.getText());
        pst.executeUpdate();
        JOptionPane.showMessageDialog(this, "Account Added!");
        updateDB();

        jTextField1.setText(null);
        jTextField2.setText(null);
        jTextField3.setText(null);
        jTextField4.setText(null);
        jTextField5.setText(null);
        jTextField6.setText(null);
        jComboBox1.setSelectedItem("Select");

        }
    } catch (Exception e)
    {
    JOptionPane.showMessageDialog((null), e);
}
```

Clears fields once data has been added.



Takes the text fields inputted by the user and inserts the data as a new row in the 'Employees1' table regarding the columns.

Delete:

```
    try {
    Class.forName("com.mysql.cj.jdbc.Driver");
    con = DriverManager.getConnection("jdbc:mysql://localhost:3306/mysql?zeroDateTimeBehavior=CONVERT_TO_NULL", "root", "Sami9870");
    String sql = "DELETE FROM Employee1 where ID='" + jTextField6.getText() + "'";
    PreparedStatement pst = con.prepareStatement(sql);
    pst.execute();
    JOptionPane.showMessageDialog(this, "Account Deleted!");

    // Subtract ID numbers by one
    String sql1 = "UPDATE Employee1 SET ID = ID - 1 WHERE ID > " + jTextField6.getText();
    PreparedStatement pst1 = con.prepareStatement(sql1);
    pst1.executeUpdate();

    String sql2 = "SELECT MAX(ID) from Employee1";
    st = con.createStatement();
    rs = st.executeQuery(sql2);
    if (rs.next()) {
        lastid = rs.getInt(1);
    }
    updateDB();
    jTextField1.setText(null);
    jTextField2.setText(null);
    jTextField3.setText(null);
    jTextField4.setText(null);
    jTextField5.setText(null);
    jTextField6.setText(null);
    jComboBox1.setSelectedItem("Select");
} catch (ClassNotFoundException ex) {
    JOptionPane.showMessageDialog(null, ex);
} catch (SQLException ex) {
    Logger.getLogger(editaccount.class.getName()).log(Level.SEVERE, null, ex);
}
```

Delete text fields and rows from 'Employees1' table regarding the columns.

After deleting the account, we use an SQL UPDATE statement to subtract the ID numbers by one for all records that have an ID greater than the ID of the deleted account. This ensures that there are no gaps in the ID sequence.

Update:

```
    try {
        int update= JOptionPane.showConfirmDialog(null, "Confirm if you want to update this member's information", "Warning",JOptionPane.YES_NO_OPTION);
                if (update==JOptionPane.YES_OPTION) {
                    Class.forName("com.mysql.cj.jdbc.Driver");
                    con= DriverManager.getConnection("jdbc:mysql://localhost:3306/mysql?zeroDateTimeBehavior=CONVERT_TO_NULL","root","Sami9870");
         pst= con.prepareStatement("Update Employee1 Set First=?, Last=?, " +"Role=?, Age=?, "+"Email=?, Password=?"+"where ID="+jTextField6.getText());

         pst.setString(1, jTextField1.getText());
         pst.setString(2, jTextField5.getText());
         pst.setString(3, jComboBox1.getSelectedItem().toString());
         pst.setString(4, jTextField3.getText());
         pst.setString(5, jTextField2.getText());
         pst.setString(6, jTextField4.getText());
         pst.executeUpdate();
         JOptionPane.showMessageDialog(this, "Data Updated!");
         updateDB();
         jTextField1.setText(null);
         jTextField2.setText(null);
         jTextField3.setText(null);
         jTextField4.setText(null);
         jTextField5.setText(null);
         jComboBox1.setSelectedItem("Select");
         jTextField6.setText(null);

                }




    } catch (ClassNotFoundException ex) {
        JOptionPane.showMessageDialog(null, ex);

    } catch (SQLException ex) {
     Logger.getLogger(editaccount.class.getName()).log(Level.SEVERE, null, ex);
}
```

SQL Query for updating fields/ rows in the database.

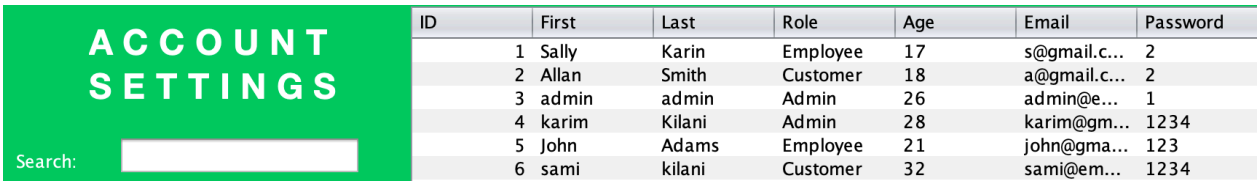
As more users create accounts and use the café software, the admin will need to be aware of all the users in the database. Using a search key could be exceptionally convenient and beneficial for the admin to search for users within the database.

Search for specified data in the database.

The code below illustrates the 'searchTemp' text field to search in the database table for the particular user using the email.

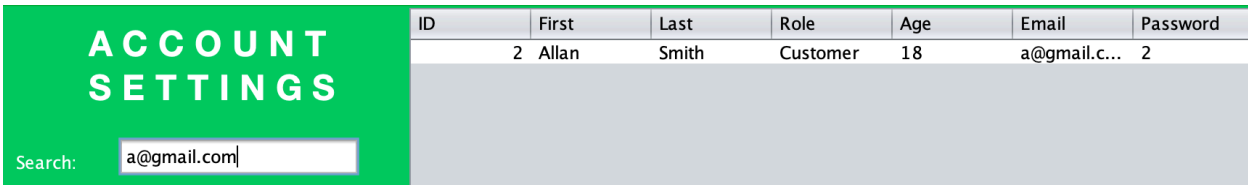
```
int q, i;
    try {
       String searchTemp = jTextField7.getText() +"%";
        Class.forName("com.mysql.cj.jdbc.Driver");
        con= DriverManager.getConnection("jdbc:mysql://localhost:3306/mysql?zeroDateTimeBehavior=CONVERT_TO_NULL","root","Sami9870");
        pst = con.prepareStatement("SELECT * from Employee1 WHERE Email LIKE '"+searchTemp+"'");

        rs= pst.executeQuery();
        ResultSetMetaData StData = rs.getMetaData();
        q= StData.getColumnCount();
        dm=(DefaultTableModel)jTable1.getModel();
        dm.setRowCount(0);
        while (rs.next()) {
            Vector columnData= new Vector();
            columnData.add(rs.getString("ID"));
            columnData.add(rs.getString("First"));
            columnData.add(rs.getString("Last"));
            columnData.add(rs.getString("Role"));
            columnData.add(rs.getString("Age"));
            columnData.add(rs.getString("Email"));
            columnData.add(rs.getString("Password"));
            dm.addRow(columnData);
        }

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

When nothing is searched the full table is displayed, showing all users/ rows:



ID	First	Last	Role	Age	Email	Password
1	Sally	Karin	Employee	17	s@gmail.c...	2
2	Allan	Smith	Customer	18	a@gmail.c...	2
3	admin	admin	Admin	26	admin@e...	1
4	karim	Kilani	Admin	28	karim@gm...	1234
5	John	Adams	Employee	21	john@gma...	123
6	sami	kilani	Customer	32	sami@em...	1234

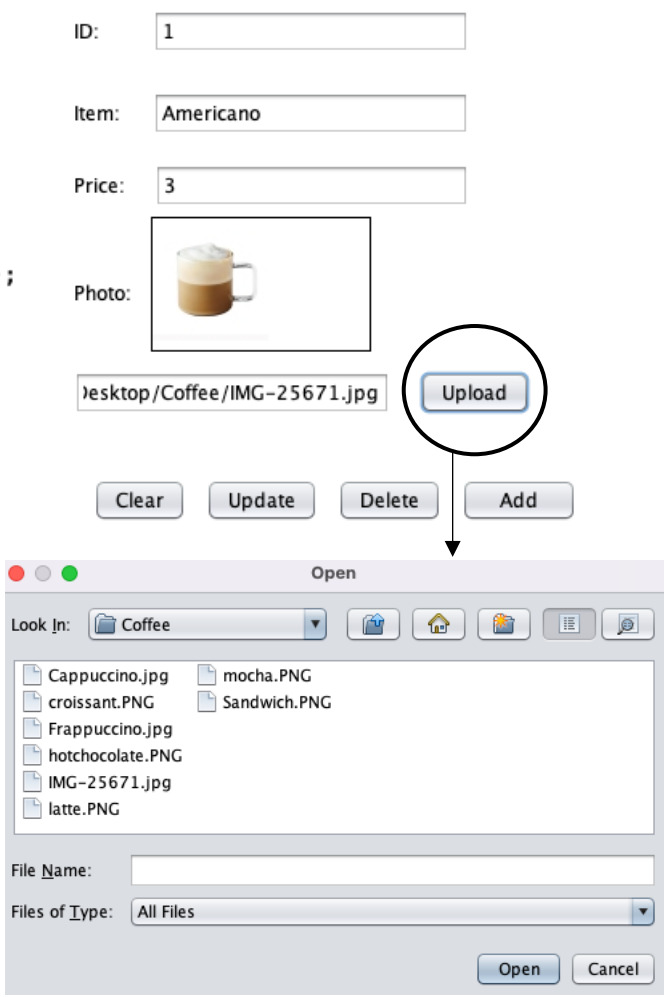
When a user's email is typed, only the row with the matching email will be displayed:



ID	First	Last	Role	Age	Email	Password
2	Allan	Smith	Customer	18	a@gmail.c...	2

As this is a Café program, a MySQL database 'Menu' is created to customize the products on the menu of the café. The employee and admin users can add, update, and delete products on their menu. An identical code to the accounts was used to do this in addition to using 'Byte' to show the picture of each item on the menu.

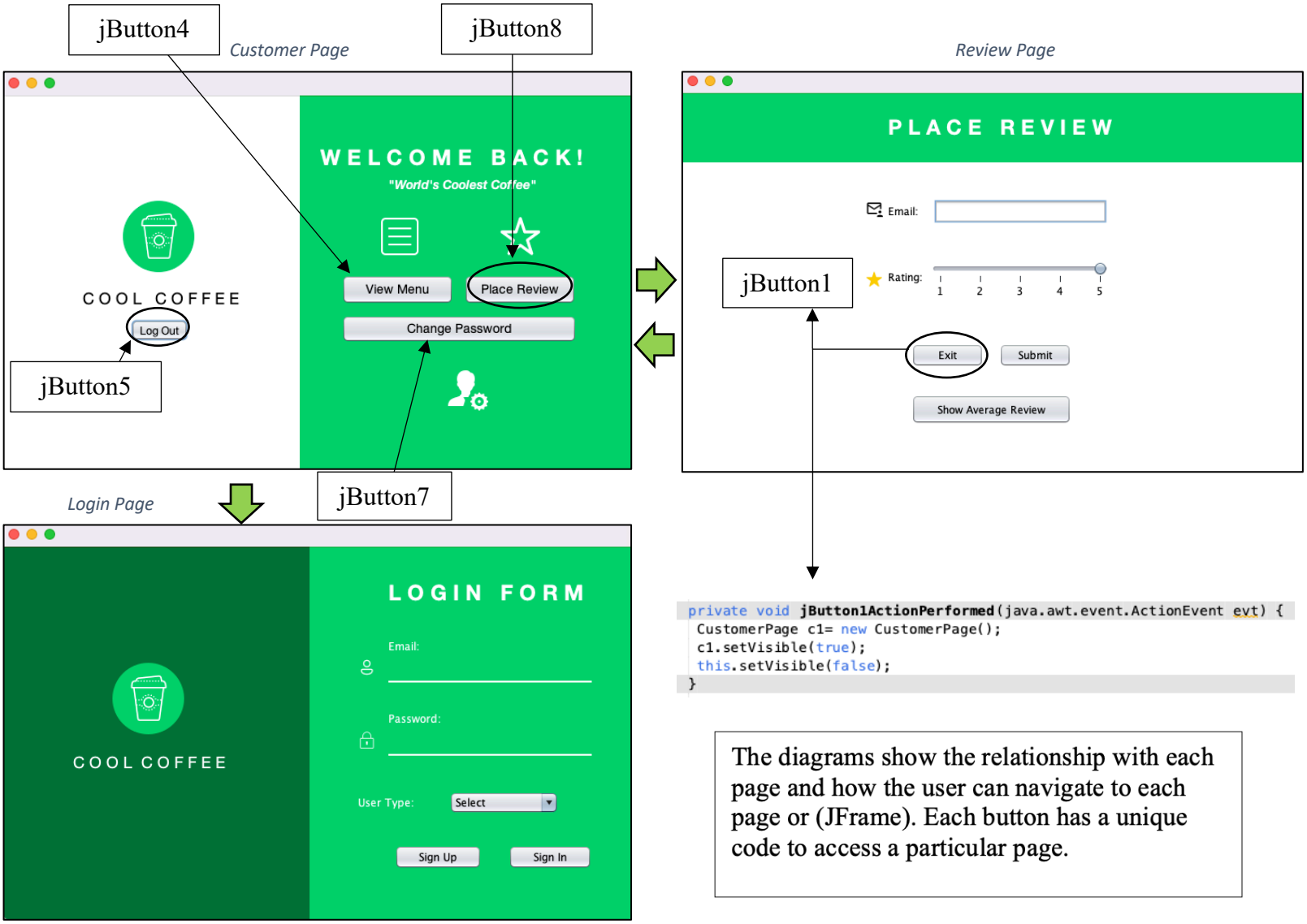
```
JFileChooser chooser = new JFileChooser();
chooser.showOpenDialog(null);
File f= chooser.getSelectedFile();
jLabel4.setIcon(new ImageIcon(f.toString()));
filename= f.getAbsolutePath();
jTextField4.setText(filename);

try {
    File image= new File (filename);
    FileInputStream fis= new FileInputStream(image);
    ByteArrayOutputStream bos= new ByteArrayOutputStream();
    byte [] buf= new byte[1024];
    for (int readNum;(readNum=fis.read(buf)) !=-1;) {
        bos.write(buf, 0, readNum);
    }
    photo=bos.toByteArray();
}
catch (Exception e) {
    JOptionPane.showMessageDialog(null, e);

}
```



The code above shows a file chooser dialog box to allow the user to select a file. Once the user selects a file, the code obtains the file's path and sets a label's icon to display the image. Additionally, the code reads the selected file and converts it to a byte array, which is stored in a variable named "photo."

2. Navigating Graphical User Interface's

```
    private void jButton5ActionPerformed(java.awt.event.ActionEvent evt) {
        Login l1= new Login();
        l1.setVisible(true);
        this.dispose();
    }

    private void jButton9ActionPerformed(java.awt.event.ActionEvent evt) {

    }

    private void jButton4ActionPerformed(java.awt.event.ActionEvent evt) {
       viewmenu v1 = new viewmenu();
       v1.setVisible(true);
       this.setVisible(false);
    }

    private void jButton8ActionPerformed(java.awt.event.ActionEvent evt) {
        review r1 = new review();
        r1.setVisible(true);
        this.setVisible(false);
    }

    private void jButton7ActionPerformed(java.awt.event.ActionEvent evt) {
setting l1= new setting();
l1.setVisible(true);
this.dispose();

    }
```




```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
 CustomerPage c1= new CustomerPage();
 c1.setVisible(true);
 this.setVisible(false);
}
```

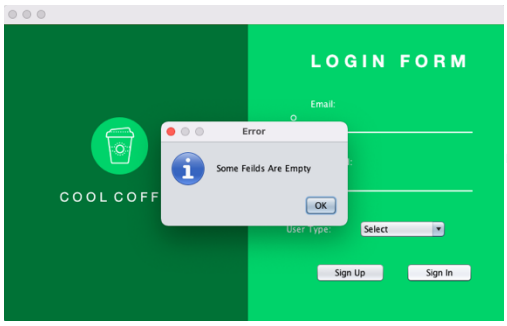
The diagrams show the relationship with each page and how the user can navigate to each page or (JFrame). Each button has a unique code to access a particular page.

3. Data Validation rules:

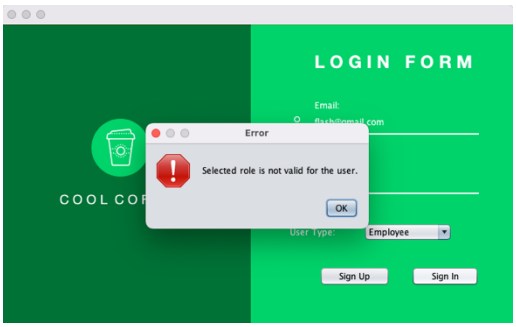
Throughout the program, a multitude of validation rules have been applied to ensure the data inputted by the user is appropriate and correct to the context of the program. The rules are adopted often when there is a change in the database.

- Login Validation:

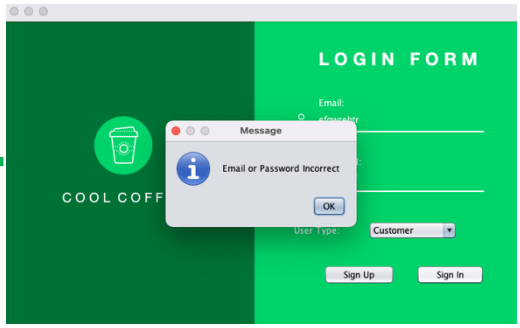
```java
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {

    String uname=jTextField1.getText();
    String pword=jPasswordField1.getText();
    String Role= jComboBox1.getSelectedItem().toString();

  if (uname.equals("") || pword.equals("") || Role.equals("Select")) {
      JOptionPane.showMessageDialog(rootPane, "Some Feilds Are Empty", "Error", 1);

  }
  else {
      try {
          Class.forName("com.mysql.cj.jdbc.Driver");
        Connection con= DriverManager.getConnection("jdbc:mysql://localhost:3306/mysql?zeroDateTimeBehavior=CONVERT_TO_NULL","root","Sami9870");
        pst= con.prepareStatement("select * from Employee1 where Email=? and Password=? ");
        pst.setString (1, uname);
        pst.setString (2, pword);
        rs= pst.executeQuery();

        if (rs.next()) {
String s1 = rs.getString("Role");
String un= rs.getString("Email");
if (Role.equalsIgnoreCase("Admin") && s1.equalsIgnoreCase("admin")) {
    AdminPage ad = new AdminPage(un);
    ad.setVisible(true);
    setVisible(false);
} else if (Role.equalsIgnoreCase("Employee") && s1.equalsIgnoreCase("employee")) {
    Menu me = new Menu(un);
    me.setVisible(true);
    setVisible(false);
} else if (Role.equalsIgnoreCase("Customer") && s1.equalsIgnoreCase("customer")) {
    CustomerPage cp = new CustomerPage(un);
    cp.setVisible(true);
    setVisible(false);
} else {
    JOptionPane.showMessageDialog(rootPane, "Selected role is not valid for the user.", "Error", JOptionPane.ERROR_MESSAGE);
}
}
        else {
            JOptionPane.showMessageDialog(rootPane, "Email or Password Incorrect");
        }

      }
      catch (Exception ex) {
          System.out.print(""+ ex);
      }
  }
```



Presence Check



Role Based Access Check (RBAC)



Authentication Check

- A presence check is used to determine whether the user has inputted values or not. An example is seen below:

```
public boolean verifyFields() {

        String item= jTextField2.getText();
        String price= jTextField3.getText();
        if ( item.trim().equals("") || price.trim().equals("")) {
            JOptionPane.showMessageDialog(null, "One or More Fields are Empty");
            return false;
        }

        else {
            return true;
        }
    }
```

If statement used to declare whether the boxes have been filled or not.

An appropriate message will be shown boxes are left empty.

- A range check is used to determine if the numbers/ letters are with in the data range.

Example 1:

```
int age = Integer.parseInt(a);
if (age < 1 || age > 100) {
    JOptionPane.showMessageDialog(null, "Invalid Age. Please enter a value between 1 and 100.");
    return false;
}
return true;
```

Example 2:

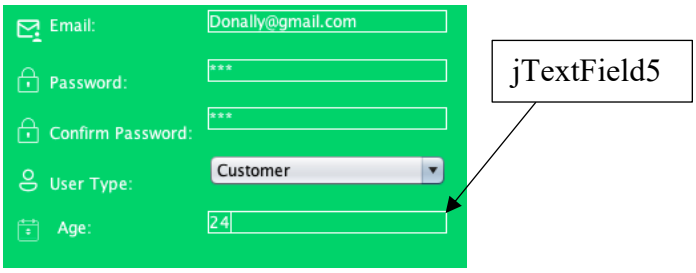
```
try {
    // Check if the rating is within the valid range of 1-5

    if (rating < 1 || rating > 5) {
        JOptionPane.showMessageDialog(null, "Rating should be between 1 and 5");
        return;
    }

    Class.forName("com.mysql.cj.jdbc.Driver");
    conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/mysql?zeroDateTimeBehavior=CONVERT_TO_NULL", "root", "Sami9870");
    pst = conn.prepareStatement(query);
    pst.setString(1, email);
    pst.setInt(2, rating);
    pst.setString(3, email);
    int result = pst.executeUpdate();
    if (result > 0) {
        JOptionPane.showMessageDialog(null, "Review added successfully");
    } else {
        JOptionPane.showMessageDialog(null, "Error adding review/ Email does not exist");
    }
} catch (SQLException ex) {
    JOptionPane.showMessageDialog(null, ex.getMessage());
} catch (ClassNotFoundException ex) {
    Logger.getLogger(review.class.getName()).log(Level.SEVERE, null, ex);
} catch (NumberFormatException ex) {
    JOptionPane.showMessageDialog(null, "Rating should be a number between 1 and 5");
}
jTextField1.setText(null);
```

- A uniqueness check is used to ensure there are no email duplicates.

```
try {
    PreparedStatement pst = con.prepareStatement("SELECT COUNT(*) FROM Employee1 WHERE Email = ?");
    pst.setString(1, email);
    ResultSet rs = pst.executeQuery();
    if (rs.next()) {
        int count = rs.getInt(1);
        if (count > 0) {
            JOptionPane.showMessageDialog(null, "Email already exists in the database.");
            return false;
        }
    }
} catch (HeadlessException | SQLException e) {
    JOptionPane.showMessageDialog(null, "Error while checking for duplicate email: " + e.getMessage());
    return false;
}
```

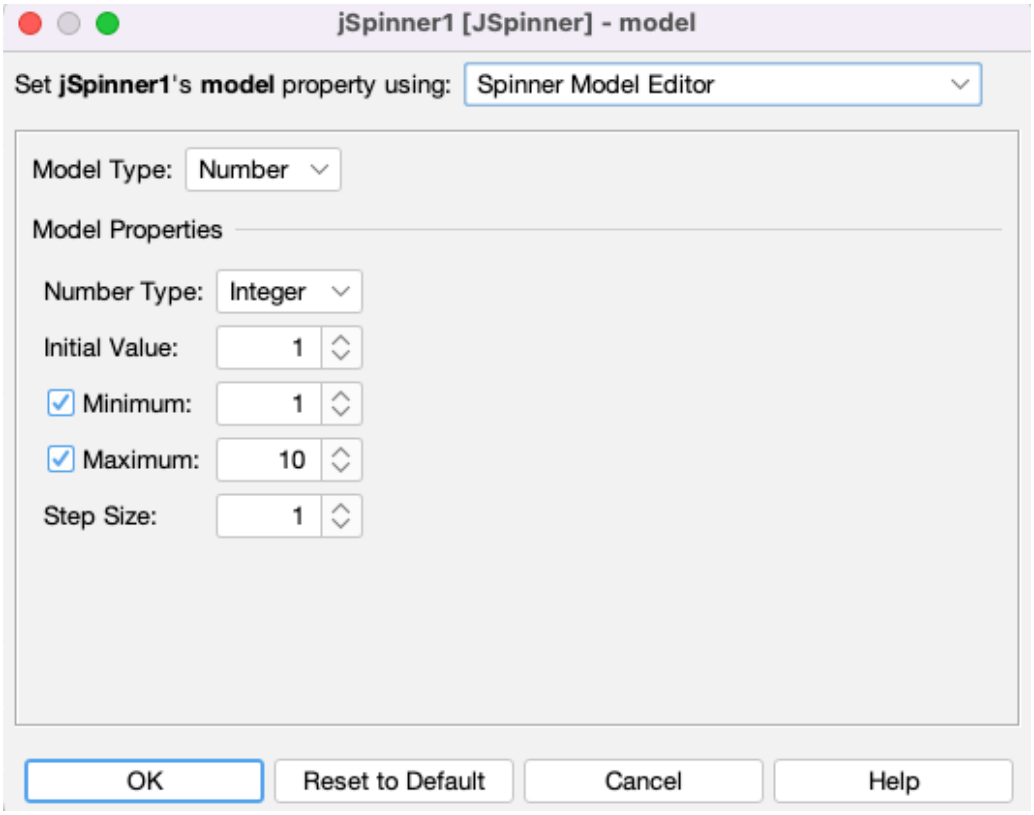
- Number validation only numbers can be inputted. In our case, age can only be a number.

```
private void jTextField5KeyTyped(java.awt.event.KeyEvent evt) {
    char c = evt.getKeyChar();
    if (!Character.isDigit(c)) {
        evt.consume();
    }
}
```



- JSpinner Type and Size Validation

The JSpinner's data has been validated; the box only accepts integers that are equal to zero or greater than 0 thus, the maximum quantity value that can be ordered per item is 10.



- A Data Format Check to ensure the data is consistent with what is expected. In our case, when the user inputs their email it should follow the email structure using '@' and '.com' at the end.

```
try {
    PreparedStatement pst = con.prepareStatement("SELECT COUNT(*) FROM Employee1 WHERE Email = ?");
    pst.setString(1, email);
    ResultSet rs = pst.executeQuery();
    if (rs.next()) {
        int count = rs.getInt(1);
        if (count > 0) {
            JOptionPane.showMessageDialog(null, "Email already exists in the database.");
            return false;
        }
    }
} catch (HeadlessException | SQLException e) {
    JOptionPane.showMessageDialog(null, "Error while checking for duplicate email: " + e.getMessage());
    return false;
}

// Validate email format
String emailPattern = "^[\\w-\\.]+@([\\w-]+\\.)+[\\w-]{2,4}$";
if (!email.matches(emailPattern)) {
    JOptionPane.showMessageDialog(null, "Invalid email format. Please enter a valid email address.");
    return false;
}
```

Email format check.

4. Joining SQL Databases

As for the checkout system, the employee inserts the ID of the user who is wishing to buy the coffee. The ID (Customer_ID), Sales_ID, ProductName, price, quantity, total (price * quantity), OrderDate data are all included in the 'Sales_product' table.

Another table 'Sales' includes the total of all the products the user has selected, pay and balance.

```
String query = "SELECT MAX(ID) FROM Sales";
pst = con.prepareStatement(query);
ResultSet rs = pst.executeQuery();

if (rs.next()) {
lastid = rs.getInt(1) + 1;
 }

query = "insert into Sales (ID, Subtotal, Pay, Balance) values (?,?,?,?)";

 pst = con.prepareStatement(query);
 pst.setInt(1, lastid);
 pst.setString(2, total);
 pst.setString(3, pay);
 pst.setString(4, balance);
 pst.executeUpdate();

 int row= jTable1.getRowCount();
```

The method retrieves the subtotal, balance, and pay amount from text fields and executes a SELECT query to retrieve the maximum ID from the "Sales" table. If the query returns a result, the method sets the "lastid" variable to the maximum ID plus one.

the method inserts the transaction data into the "Sales" table by executing an INSERT query..

Code for inserting customer's order in 'Sales_product'.

```
        String query1 = "SELECT MAX(ID) FROM Sales_product";
pst = con.prepareStatement(query1);
ResultSet rs1 = pst.executeQuery();

query1 = "insert into Sales_product (ID, Sales_ID, ProductName, Price, Quantity, Total, OrderDate) values (?,?,?,?,?,?,?)";
pst1 = con.prepareStatement(query1, Statement.RETURN_GENERATED_KEYS);

int ID = Integer.parseInt(jTextField4.getText());
String Productname= "";
int sales_id;
int price;
int qty;
int tot=0;
String OrderDate = ""+now();
OrderDate = OrderDate.substring(0,10);
for (int i=0; i<jTable1.getRowCount();i++) {
    sales_id= (int)jTable1.getValueAt(i, 0);
    Productname= (String)jTable1.getValueAt(i, 1);
    price= (int)jTable1.getValueAt(i, 2);
    qty= (int)jTable1.getValueAt(i, 3);
    tot= (int)jTable1.getValueAt(i, 4);

    pst1.setInt(1, ID);
    pst1.setInt(2, lastid);
    pst1.setString(3, Productname);
    pst1.setInt(4, price);
    pst1.setInt(5, qty);
    pst1.setInt(6, tot);
    pst1.setString(7, OrderDate);
    pst1.executeUpdate();
    ResultSet rs2 = pst1.getGeneratedKeys();
}
```

```
JOptionPane.showMessageDialog(this, "Sales Complete");
HashMap <String , Object> parameters = new HashMap<>();
parameters.put("Parameters1", lastid);
```

HashMap is used to pass a key-value pair as a parameter to a method.

A HashMap is a collection that stores data in key-value pairs, where each key is unique. In this case, the key is a string "Parameters1" and the value is the variable "lastid".

Code for calculating sum of all products added to cart by the customer, the amount the customer is paying, and the subtraction of the sum with the pay and storing this in the 'Sales' table.

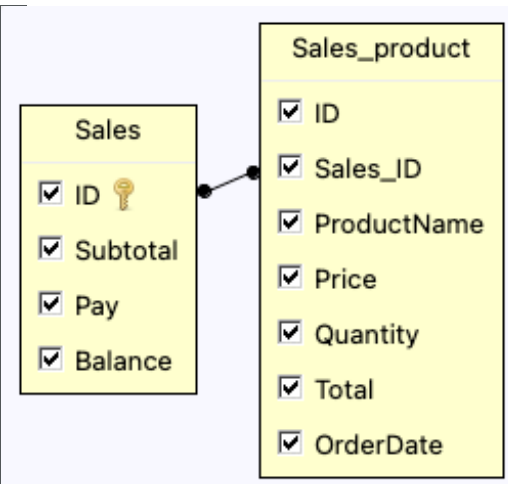
```
int tot= Integer.parseInt(jTextField3.getText());
int pay= Integer.parseInt(jTextField1.getText());

int bal= pay-tot;
if (bal >= 0) {
jTextField2.setText(String.valueOf(bal));

Sales();
}
else {
    JOptionPane.showMessageDialog(null, "Payment unsuccessful.");
}
```

Payment validation, if the payment the user inputted is less than the total; the payment will not go through.

Using TIBCO Jasper soft Studio and adding Jasper Reports library, I was able to create a receipt that was printable for every time a customer bought coffee. I used an SQL query to merge the Sales_ID column from the 'Sales_Product' table and the ID column from the 'Sales' table.

The main purpose of this query is to retrieve the details of the latest sales transaction from the database.

```sql
SELECT mysql.`Sales_product`.`ID`,
  mysql.`Sales`.`ID`,
  mysql.`Sales`.`Subtotal`,
  mysql.`Sales`.`Pay`,
  mysql.`Sales`.`Balance`,
  mysql.`Sales_product`.`Sales_ID`,
  mysql.`Sales_product`.`ProductName`,
  mysql.`Sales_product`.`Price`,
  mysql.`Sales_product`.`Quantity`,
  mysql.`Sales_product`.`Total`,
  mysql.`Sales_product`.`OrderDate`
FROM mysql.`Sales_product`
  INNER JOIN mysql.`Sales` ON
   mysql.`Sales`.`ID` = mysql.`Sales_product`.`Sales_ID`
WHERE
   mysql.`Sales_product`.`Sales_ID` = (SELECT MAX(Sales_ID) from Sales_product)
```



The query uses the INNER JOIN keyword to join the Sales_product and Sales tables based on the condition that the ID column of the Sales table is equal to the Sales_ID column of the Sales_product table. This will combine the rows from both tables that have matching ID and Sales_ID values.

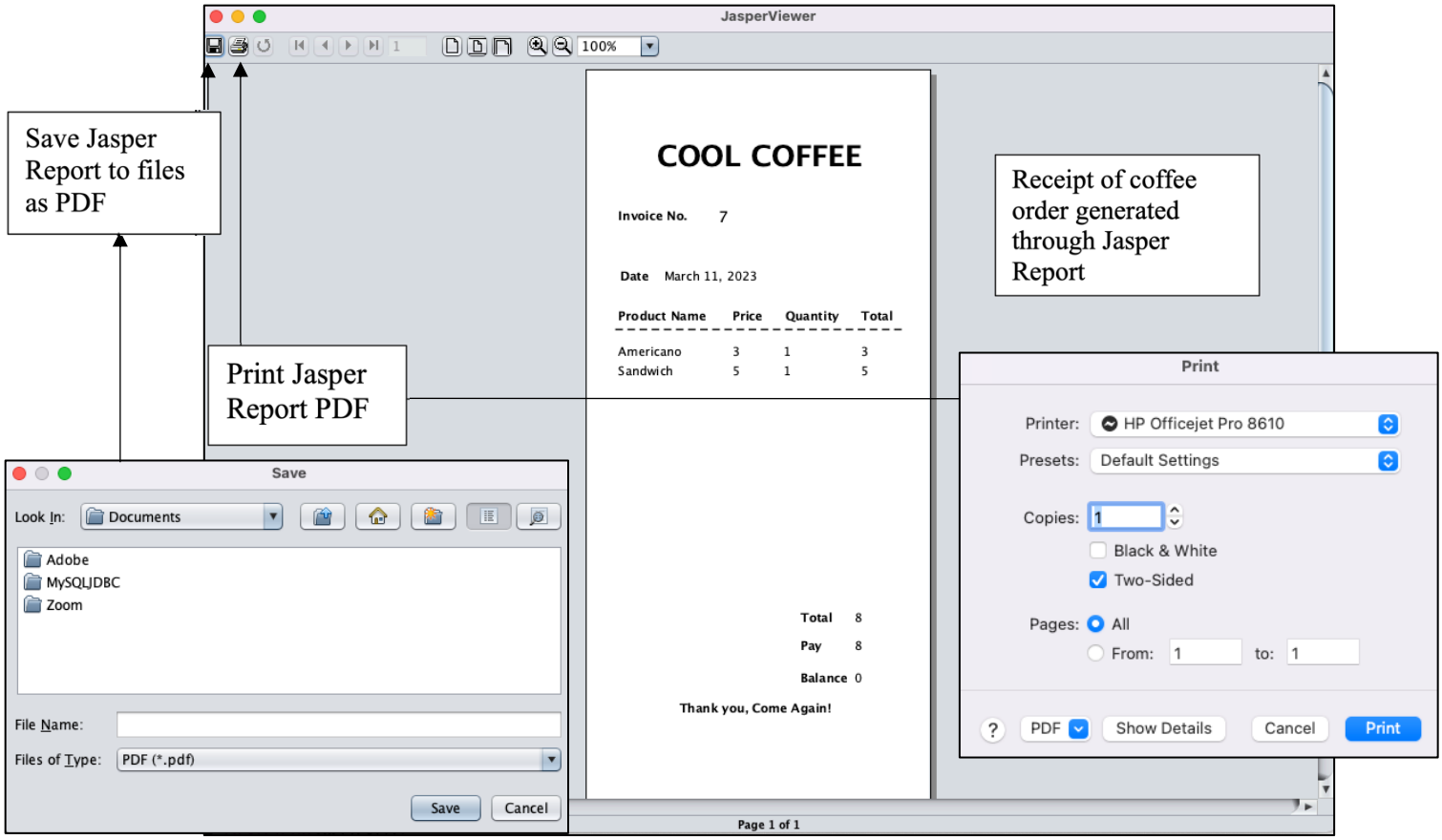
The query starts by selecting specific columns from both tables using the SELECT statement. The columns selected include the ID columns from both tables, the Subtotal, Pay, and Balance columns from the Sales table, and the ProductName, Price, Quantity, Total, and OrderDate columns from the Sales_product table.

The query applies a filter to only return the rows where the Sales_ID column of the Sales_product table is equal to the maximum Sales_ID value in the Sales_product table. This is achieved using a subquery that selects the maximum Sales_ID value from the Sales_product table.

The code below uses the JasperReports library to generate and display a report for an invoice regarding the two tables.

```java
        try {

    Class.forName("com.mysql.cj.jdbc.Driver");
    Connection con =     DriverManager.getConnection("jdbc:mysql://localhost:3306/mysql?zeroDateTimeBehavior=CONVERT_TO_NULL","root","Sami9870");
    String reportPath = "invoice.jrxml";
    JasperReport jr = JasperCompileManager.compileReport(reportPath);
    JasperPrint jp = JasperFillManager.fillReport(jr,null, con);
    JasperViewer.viewReport(jp);
    con.close();
        } catch (Exception ex) {
            Logger.getLogger(ordercoffee.class.getName()).log(Level.SEVERE, null, ex);
        }



    } catch (ClassNotFoundException ex) {
        Logger.getLogger(ordercoffee.class.getName()).log(Level.SEVERE, null, ex);
    }
    catch (SQLException ex) {
        Logger.getLogger(ordercoffee.class.getName()).log(Level.SEVERE, null, ex);
    }
}
```

Jasper report additional library



5. Try & Catch Blocks (Exception Handling)

try and catch Blocks have been used often throughout this program to avoid crashing during any process (retrieving data from the MySQL Database, adding data etc...)

```
public void updateDB() {
 int q, i;
 try {
     Class.forName("com.mysql.cj.jdbc.Driver");
     con= DriverManager.getConnection("jdbc:mysql://localhost:3306/mysql?zeroDateTimeBehavior=CONVERT_TO_NULL","root","Sami9870");
     pst = con.prepareStatement("SELECT * from Employee1");

     rs= pst.executeQuery();
     ResultSetMetaData StData = rs.getMetaData();
     q= StData.getColumnCount();
     dm=(DefaultTableModel)jTable1.getModel();
     dm.setRowCount(0);
     while (rs.next()) {
         Vector columnData= new Vector();
         columnData.add(rs.getInt("ID"));
         columnData.add(rs.getString("First"));
         columnData.add(rs.getString("Last"));
         columnData.add(rs.getString("Role"));
         columnData.add(rs.getString("Age"));
         columnData.add(rs.getString("Email"));
         columnData.add(rs.getString("Password"));
         dm.addRow(columnData);
     }

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

Retrieving data from MySQL table and inserting in a table.

6. Creating Bar Charts using JFreeChart

I designed a graph using NetBeans showing the Sales of the Café for each month. The code retrieves sales data from a MySQL database and calculates the total revenue for each date. It then creates a bar graph using JFreeChart to display the total sales by date. The graph is displayed in a chart panel using a JFrame.

```java
public void generateSalesBarGraph() {
    try {
        Class.forName("com.mysql.cj.jdbc.Driver");
        Connection con = DriverManager.getConnection("jdbc:mysql://localhost:3306/mysql?zeroDateTimeBehavior=CONVERT_TO_NULL","root","Sami9870");

        // Retrieve the sales data from the database, grouped by month
        String query = "SELECT MONTH(OrderDate) AS Month, SUM(Total) AS TotalSales FROM Sales_product GROUP BY MONTH(OrderDate)";
        PreparedStatement pst = con.prepareStatement(query);
        ResultSet rs = pst.executeQuery();

        // Calculate the total sales/revenue for each month
        DefaultCategoryDataset dataset = new DefaultCategoryDataset();
        while (rs.next()) {
            int month = rs.getInt("Month");
            int totalSales = rs.getInt("TotalSales");
            dataset.setValue(totalSales, "Sales", getMonthName(month));
        }

        // Create a bar graph using JFreeChart
        JFreeChart chart = ChartFactory.createBarChart("Total Sales by Month", "Month", "Sales in Dinar", dataset, PlotOrientation.VERTICAL, false, true, false);

        // Display the bar graph in a chart panel
        ChartPanel chartPanel = new ChartPanel(chart);
        JFrame frame = new JFrame();
        frame.setContentPane(chartPanel);
        frame.setSize(600, 400);
        frame.setVisible(true);
    } catch (ClassNotFoundException ex) {
        Logger.getLogger(ordercoffee.class.getName()).log(Level.SEVERE, null, ex);
    } catch (SQLException ex) {
        Logger.getLogger(ordercoffee.class.getName()).log(Level.SEVERE, null, ex);
    }
}

// Helper method to get the name of a month from its number (1 = January, 2 = February, etc.)
private String getMonthName(int month) {
    String[] monthNames = {"January", "February", "March", "April", "May", "June", "July", "August", "September", "October", "November", "December"};
    return monthNames[month - 1];
}
```

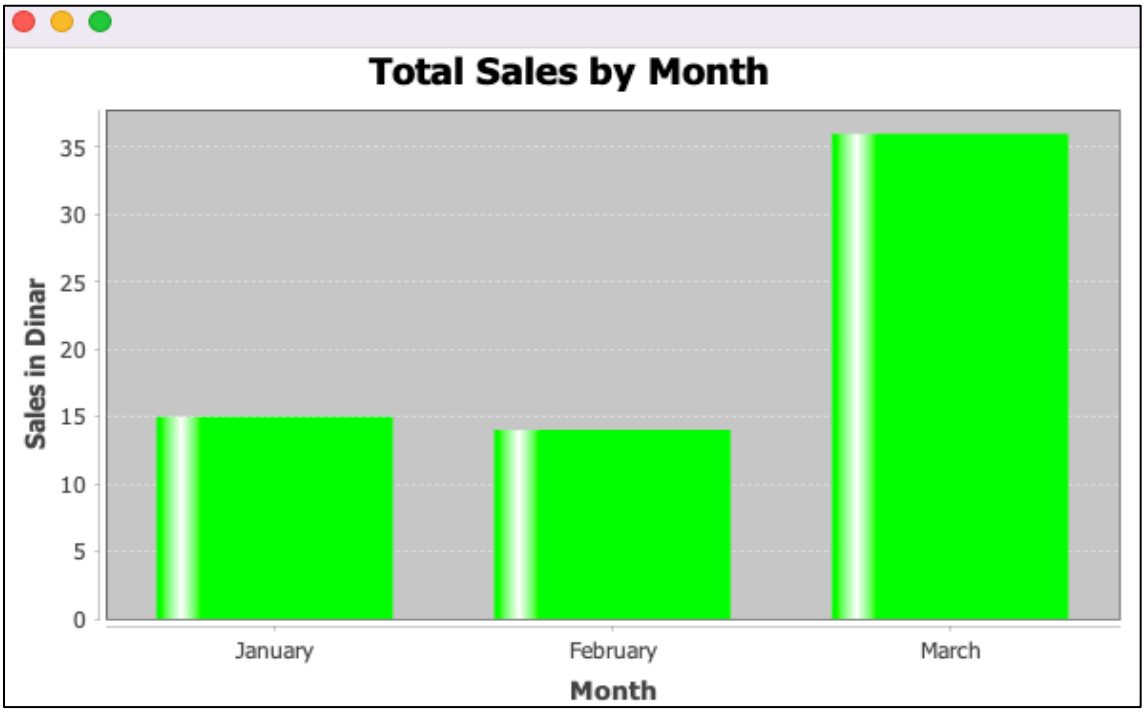
A while loop is used to iterate through the result set and calculate the total sales revenue for each month. The month name is obtained using a helper method called "getMonthName"

Creates a new bar chart using JFrameChart (Java Library)

Sets graph title, bars color, axis titles, size and background color

One dimensional array to store months for each sale and used a method.

Bar Chart created using sample data:



Word Count: 951

7. Bibliography:

I. Oracle. (2021). NetBeans IDE. Retrieved March 11, 2023, from https://netbeans.org/

II. MySQL. (2021). MySQL. Retrieved March 11, 2023, from https://www.mysql.com/

III. JasperReports. (2021). JasperReports Library. Retrieved March 11, 2023, from https://community.jaspersoft.com/project/jasperreports-library

IV. Jaspersoft. (2021). JasperReports Library. TIBCO Software Inc. https://community.jaspersoft.com/project/jasperreports-library

V. BoostMyTool (2021). *Connect to MySQL Database from NetBeans 12.5 (2021) and Run SQL Queries*. [online] www.youtube.com. Available at: https://www.youtube.com/watch?v=VNoU590W750&t=11s [Accessed 11 Oct. 2022].

VI. JRS-SE (2018). *Build Jasper Report using Jasper Library using Net Beans IDE:* [online] Jaspersoft Community. Available at: https://community.jaspersoft.com/wiki/build-jasper-report-using-jasper-library-using-net-beans-ide [Accessed 12 Oct. 2022].

VII. netbeans.apache.org. (n.d.). *Connecting to a MySQL Database*. [online] Available at: https://netbeans.apache.org/kb/docs/ide/mysql.html.

VIII. vakamble083 (2016). *How to use Java Code in Tibco Jaspersoft Studio*. [online] Jaspersoft Community. Available at: https://community.jaspersoft.com/questions/988646/how-use-java-code-tibco-jaspersoft-studio [Accessed 16 Nov. 2022].

IX. Wikipedia Contributors (2019). *Java Database Connectivity*. [online] Wikipedia. Available at: https://en.wikipedia.org/wiki/Java_Database_Connectivity.

X. www.javatpoint.com. (n.d.). *JDBC Tutorial | What is Java Database Connectivity(JDBC) - javatpoint*. [online] Available at: https://www.javatpoint.com/java-jdbc.

XI. www.tutorialspoint.com. (n.d.). *JDBC - Introduction - Tutorialspoint*. [online] Available at: https://www.tutorialspoint.com/jdbc/jdbc-introduction.htm.