BALIGE
PUBLISHING

DATA SCIENCE USING JDBC AND MYSQL

WITH OBJECT-ORIENTED APPROACH AND APACHE NETBEANS IDE

BALIGE CITY
NORTH SUMATERA

VIVIAN SIAHAAN
RISMON HASIHOLAN SIANIPAR

DATA SCIENCE USING JDBC AND MYSQL WITH OBJECT-ORIENTED APPROACH AND APACHE NETBEANS IDE

DATA SCIENCE USING JDBC AND MYSQL WITH OBJECT-ORIENTED APPROACH AND APACHE NETBEANS IDE

First Edition

VIVIAN SIAHAAN
RISMON HASIHOLAN SIANIPAR

Published: MAY 2023
Production reference: 01050223
Published by BALIGE Publishing Ltd.
BALIGE, North Sumatera

ABOUT THE AUTHOR
ABOUT THE AUTHOR

Vivian Siahaan is a fast-learner who likes to do new things. She was born, raised in Hinalang Bagasan, Balige, on the banks of Lake Toba, and completed high school education from SMAN 1 Balige. She started herself

 learning Java, Android, JavaScript, CSS, C ++, Python, R, Visual Basic, Visual C #, MATLAB, Mathematica, PHP, JSP, MySQL, SQL Server, Oracle, Access, and other programming languages. She studied programming from scratch, starting with the most basic syntax and logic, by building several simple and applicable GUI applications. Animation and games are fields of programming that are interests that she always wants to develop. Besides studying mathematical logic and programming, the author also has the pleasure of reading novels. Vivian Siahaan has written dozens of ebooks that have been published on Sparta Publisher: Data Structure with Java; Java Programming: Cookbook; C ++ Programming: Cookbook; C Programming For High Schools / Vocational Schools and Students; Java Programming for SMA / SMK; Java Tutorial: GUI, Graphics and Animation; Visual Basic Programming: From A to Z; Java Programming for Animation and Games; C # Programming for SMA / SMK and Students; MATLAB For Students and Researchers; Graphics in JavaScript: Quick Learning Series; JavaScript Image Processing Methods: From A to Z; Java GUI Case Study: AWT & Swing; Basic CSS and JavaScript; PHP / MySQL Programming: Cookbook; Visual Basic: Cookbook; C ++ Programming for High Schools / Vocational Schools and Students; Concepts and Practices of C ++; PHP / MySQL For Students; C # Programming: From A to Z; Visual Basic for SMA / SMK and Students; C # .NET and SQL Server for High School / Vocational School and Students. At the ANDI Yogyakarta publisher, Vivian Siahaan also wrote a number of books including: Python Programming Theory and Practice; Python GUI Programming; Python GUI and Database; Build From Zero School Database Management System In Python / MySQL; Database Management System in Python / MySQL; Python / MySQL For Management Systems of Criminal Track Record Database; Java / MySQL For Management Systems of Criminal Track Records Database; Database and Cryptography Using Java / MySQL; Build From Zero School Database Management System With Java / MySQL.

 **Rismon Hasiholan Sianipar** was born in Pematang Siantar, in 1994. After graduating from SMAN 3 Pematang Siantar 3, the writer traveled to the city of Jogjakarta. In 1998 and 2001 the author completed his Bachelor of Engineering (S.T) and Master of Engineering (M.T) education in the Electrical Engineering of Gadjah Mada University, under the guidance of Prof. Dr. Adhi Soesanto and Prof. Dr. Thomas Sri Widodo, focusing on research on non-stationary signals by analyzing their energy using time-frequency maps. Because of its non-stationary nature, the distribution of signal energy becomes very dynamic on a time-frequency map. By mapping the distribution of energy in the time-frequency field using discrete wavelet transformations, one can design non-linear filters so that they can analyze the pattern of the data contained in it. In 2003, the author received a Monbukagakusho scholarship from the Japanese Government. In 2005 and 2008, he completed his Master of Engineering (M.Eng) and Doctor of Engineering (Dr.Eng) education at Yamaguchi University, under the guidance of Prof. Dr. Hidetoshi Miike. Both the master's thesis and his doctoral thesis, R.H. Sianipar combines SR-FHN (Stochastic Resonance Fitzhugh-Nagumo) filter strength with cryptosystem ECC (elliptic curve cryptography) 4096-bit both to suppress noise in digital images and digital video and maintain its authenticity. The results of this study have been documented in international scientific journals and officially patented in Japan. One of the patents was published in Japan with a registration number 2008-009549.

He is active in collaborating with several universities and research institutions in Japan, particularly in the fields of cryptography, cryptanalysis and audio / image / video digital forensics. R.H. Sianipar also has experience in conducting code-breaking methods (cryptanalysis) on a number of intelligence data that are the object of research studies in Japan. R.H. Sianipar has a number of Japanese patents, and has written a number of national / international scientific articles, and dozens of national books. R.H. Sianipar has also participated in a number of workshops related to cryptography, cryptanalysis, digital watermarking, and digital forensics. In a number of workshops, R.H. Sianipar helps Prof. Hidetoshi Miike to create applications related to digital image / video processing, steganography, cryptography, watermarking, non-linear screening, intelligent descriptor-based computer vision, and others, which are used as training materials. Field of interest in the study of R.H. Sianipar is multimedia security, signal processing / digital image / video, cryptography, digital communication, digital forensics, and data compression / coding. Until now, R.H. Sianipar continues to develop applications related to analysis of signal, image, and digital video, both for research purposes and for commercial purposes based on the Python programming language, MATLAB, C ++, C, VB.NET, C # .NET, R, and Java.

ABOUT THE BOOK

This book uses the Sakila sample database which is a fictitious database designed to represent a DVD rental store. The 15 tables of the database include film, film_category, actor, customer, rental, payment and inventory among others. The Sakila sample database is intended to provide a standard schema that can be used for examples in books, tutorials, articles, samples, and so

forth. You can download the sample database from http://viviansiahaan.blogspot.com/2023/04/data-science-using-jdbc-and-mysql-with.html.

In this project, you will design the form for every table and you will plot: top 10 film distribution by release year; top 10 film distribution by rating; top 10 film distribution by rental duration; top 10 film distribution by language; film distribution by categorized rental rate; film distribution by categorized length; film distribution by categorized replacement cost; top 10 film distribution by actor name; top 10 actor name distribution by average rental rate; top 10 actor name distribution by average replacement cost; film distribution by rating; rating distribution by average rental rate; rating distribution by average replacement cost; top 10 film distribution by category name, category distribution by average replacement cost; category distribution by average rental rate; category distribution by length; top 10 city distribution by by country; top 10 address distribution by district, top 10 address distribution by country; top 10 address distribution by city; top 10 address distribution by district; top 10 address distribution by country; top 10 address distribution by city; top 10 inventory distribution by release year; top 10 inventory distribution by film rating; top 10 inventory distribution by film language; top 10 inventory distribution by film rental duration; top 10 inventory distribution by city; top 10 inventory distribution by country; top 10 customer distribution by country; top 10 customer distribution by city; top 10 customer distribution by district; top 10 customer distribution by store country; top 10 customer distribution by store city; top 10 customer distribution by store district; top 10 staff distribution by country; top 10 staff distribution by city; rental distribution by year of rental date; rental distribution by month of rental date; 10 rental distribution by week of rental date; rental distribution by day of rental date; rental distribution by quarter of rental date; rental distribution by film release year; rental

distribution by film duration; rental distribution by film rating; top 10 rental distribution by staff name; rental distribution by film language; top 10 rental distribution by film title; rental distribution by customer active; top 10 rental distribution by film category; top 10 rental distribution by actor name; top 10 rental distribution by customer name; top 10 rental distribution by customer city; top 10 rental distribution by customer country, top 10 rental distribution by customer district; payment distribution by year of payment date; payment distribution by month of payment date; top 10 payment distribution by week of payment date; payment distribution by day of payment date; payment distribution by quarter of payment date; payment distribution by film release year; payment distribution by film duration; payment distribution by film rating; top 10 payment distribution by staff name; payment distribution by film language; top 10 payment distribution by film title; payment distribution by customer active; top 10 payment distribution by film category; top 10 payment distribution by actor name; top 10 payment distribution by customer name; top 10 payment distribution by customer city; top 10 payment distribution by customer country; and top 10 payment distribution by customer district.

CONTENT
CONTENT

DESIGNING GUI
POPULATING TABLE AND COMBOBOXES
DISPLAYING AND NAVIGATING DATA ROW BY ROW
UPDATING RECORD
INSERTING NEW RECORD
DELETING RECORD
PLOTTING CHART
SOURCE CODE

PAYMENT FORM

CREATING AND POPULATING PAYMENT TABLE
DESIGNING GUI
POPULATING TABLE AND COMBOBOXES
DISPLAYING AND NAVIGATING DATA ROW BY ROW
UPDATING RECORD
INSERTING NEW RECORD
DELETING RECORD
PLOTTING CHART
SOURCE CODE

MAIN FORM

DESIGNING GUI

PROJECT AND UTILITY CLASS

PROJECT AND UTILITY CLASS

DESCRIPTION
DESCRIPTION

This book uses the Sakila sample database which is a fictitious database designed to represent a DVD rental store. The tables of the database include film, film_category, actor, customer, rental, payment and inventory among others. The Sakila sample database is intended to provide a standard schema that can be used for examples in books, tutorials, articles, samples, and so forth. You can download the sample database from http://viviansiahaan.blogspot.com/2023/04/data-science-using-jdbc-and-mysql-with.html.

In this project, you will design the form for every table and you will plot: top 10 film distribution by release year; top 10 film distribution by rating; top 10 film distribution by rental duration; top 10 film distribution by language; film distribution by categorized rental rate; film distribution by categorized length; film distribution by categorized replacement cost; top 10 film distribution by actor name; top 10 actor name distribution by average rental rate; top 10 actor name distribution by average replacement cost; film distribution by rating; rating

distribution by average rental rate; rating distribution by average replacement cost; top 10 film distribution by category name, category distribution by average replacement cost; category distribution by average rental rate; category distribution by length; top 10 city distribution by by country; top 10 address distribution by district, top 10 address distribution by country; top 10 address distribution by city; top 10 address distribution by district; top 10 address distribution by country; top 10 address distribution by city; top 10 inventory distribution by release year; top 10 inventory distribution by film rating; top 10 inventory distribution by film language; top 10 inventory distribution by film rental duration; top 10 inventory distribution by city; top 10 inventory distribution by country; top 10 customer distribution by country; top 10 customer distribution by city; top 10 customer distribution by district; top 10 customer distribution by store country; top 10 customer distribution by store city; top 10 customer distribution by store district; top 10 staff distribution by country; top 10 staff distribution by city; rental distribution by year of rental date; rental distribution by month of rental date; 10 rental distribution by week of rental date; rental distribution by day of rental date; rental distribution by quarter of rental date; rental distribution by film release year; rental distribution by film duration; rental distribution by film rating; top 10 rental distribution by staff name; rental distribution by film language; top 10 rental distribution by film title; rental distribution by customer active; top 10 rental distribution by film category; top 10 rental distribution by actor name; top 10 rental distribution by customer name; top 10 rental distribution by customer city; top 10 rental distribution by customer country, top 10 rental distribution by customer district; payment distribution by year of payment date; payment distribution by month of payment date; top 10 payment distribution by week of payment date; payment distribution by day of payment date; payment distribution by quarter of payment date; payment distribution by film release year;

payment distribution by film duration; payment distribution by film rating; top 10 payment distribution by staff name; payment distribution by film language; top 10 payment distribution by film title; payment distribution by customer active; top 10 payment distribution by film category; top 10 payment distribution by actor name; top 10 payment distribution by customer name; top 10 payment distribution by customer city; top 10 payment distribution by customer country; and top 10 payment distribution by customer district.

The Sakila sample database includes 15 tables and the table relationships are showcased in the following entity relationship diagram.

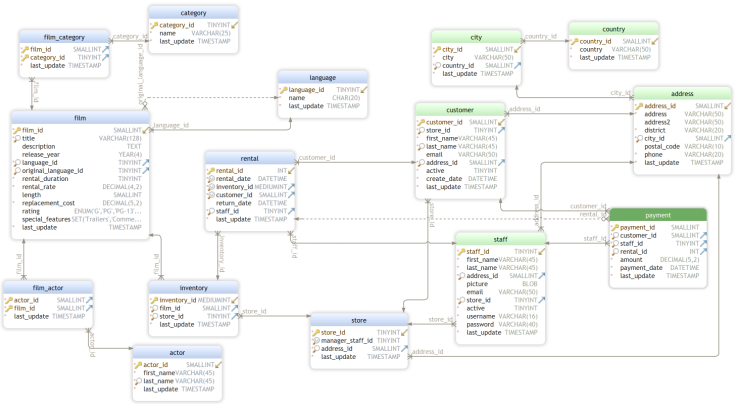


Figure 1 The ERD of Sakila database

CREATING PROJECT AND UTILITY CLASS
CREATING PROJECT AND UTILITY CLASS

Step 1	Create a new MySQL database named **Sakila** using **phpMyAdmin**.
Step 2	In NetBeans, create a new project named **Sakila**. Add **jcom jfreechart.jar**, **jcalendar.jar**, and **mysql-connector-java.jar** files libraries in your project.
Step 3	Add a new properties file into the project. Right click on the project, choos **Other > Other > Properties File**. Click Next, and set its file name as **d** Set the url string, user, and password of yours as follows:

```
1  url =
```

```
2 jdbc:mysql://localhost:3306/sakila
3 user = vivian
  password = vivian
```

Step 4

Then, create a public class named **Utility**. Define two new **getConnection()** and **testConnection()**.

The **getConnection()** method is responsible for creating a connectio database using the properties specified in the database.properties file. these properties from the file using a **FileInputStream** and a **Propertie** The method then uses these properties to create a **Connection** object **DriverManager** class. If there is an error reading the file or crea connection, the method logs a **SEVERE** level message with the **Logger** o returns **null**.

The **testConnection()** method is responsible for testing the database co It does this by calling **getConnection()** and attempting to create a conn the database. If the connection is successful, it logs an **INFO** level mess the **Logger** object indicating that the connection was successful. If th error creating the connection, the method logs a **SEVERE** level message **Logger** object.

```
1  package Sakila;
2  import java.io.FileInputStream;
3  import java.sql.Connection;
4  import java.sql.DriverManager;
5  import java.sql.SQLException;
6  import java.util.Properties;
7  import java.util.logging.Level;
8  import java.util.logging.Logger;
9  import java.sql.*;
10 import java.awt.BorderLayout;
11 import java.awt.Color;
12 import java.awt.Component;
13 import java.awt.RenderingHints;
14 import java.awt.image.BufferedImage;
15 import java.io.File;
16 import java.io.IOException;
17 import javax.swing.JOptionPane;
18 import java.text.DecimalFormat;
19 import java.text.NumberFormat;
20 import java.util.ArrayList;
21 import java.util.HashMap;
22 import java.util.List;
23 import java.util.Map;
24 import java.util.Objects;
25 import javax.imageio.ImageIO;
26 import javax.swing.ImageIcon;
```

```
27 import javax.swing.JComboBox;
28 import javax.swing.JFileChooser;
29 import javax.swing.JFrame;
30 import javax.swing.JLabel;
31 import javax.swing.JPanel;
32 import javax.swing.JTable;
33 import javax.swing.JTextField;
34 import javax.swing.SwingUtilities;
35 import javax.swing.UIManager;
36 import javax.swing.UnsupportedLookAndFeelException;
37 import javax.swing.plaf.nimbus.NimbusLookAndFeel;
38 import javax.swing.table.DefaultTableCellRenderer;
39 import javax.swing.table.TableCellRenderer;
40 import javax.swing.table.TableColumn;
41 import org.jfree.chart.ChartFactory;
42 import org.jfree.chart.ChartPanel;
43 import org.jfree.chart.JFreeChart;
44 import org.jfree.chart.labels.ItemLabelAnchor;
45 import org.jfree.chart.labels.ItemLabelPosition;
46 import
47 org.jfree.chart.labels.StandardCategoryItemLabelGenerator;
48 import
49 org.jfree.chart.labels.StandardPieSectionLabelGenerator;
50 import org.jfree.chart.plot.CategoryPlot;
51 import org.jfree.chart.plot.PiePlot;
52 import org.jfree.chart.plot.PlotOrientation;
53 import
54 org.jfree.chart.renderer.category.CategoryItemRenderer;
55 import org.jfree.chart.ui.TextAnchor;
56 import org.jfree.data.category.DefaultCategoryDataset;
57 import org.jfree.data.general.DefaultPieDataset;
58
59 public class Utility {
60     private static final Logger LOGGER =
61 Logger.getLogger(Utility.class.getName());
62
63     public static Connection getConnection() throws
64 SQLException {
65         Connection conn = null;
66         Properties props = new Properties();
67         try (FileInputStream fis = new
68 FileInputStream("database.properties")) {
69             props.load(fis);
70             String url = props.getProperty("url");
71             String user = props.getProperty("user");
72             String password =
73 props.getProperty("password");
74             conn = DriverManager.getConnection(url, user,
75 password);
76         } catch (IOException ex) {
77             LOGGER.log(Level.SEVERE, null, ex);
78         }
79         return conn;
80
```

```
    }

    public static void testConnection() {
        try (Connection conn = getConnection()) {
            String message = String.format("Successfully
connected to %s database", conn.getCatalog());
            LOGGER.log(Level.INFO, message);
        } catch (SQLException ex) {
            LOGGER.log(Level.SEVERE, null, ex);
        }
    }
}
```

Step 5

In **Utility** class, define a new method named **populate_combobox()**. It ta
parameters:

- **sql**: the SQL statement to execute in order to retrieve data for p
  the combo box.
- **jcb**: the **JComboBox** object that will be populated with the retriev
- **obj_form**: the parent **JFrame** object.

The method first removes all items from the combo box u
jcb.removeAllItems() method. Then it creates a **ResultSet** object by
the SQL query with **getConnection()** method.

Next, the method loops through the **ResultSet** and retrieves the first
value for each row, casts it to a generic type **T**, and adds it to the combo
the **jcb.addItem()** method. Finally, the method closes the **ResultSet**.

If any SQL exception is thrown, the method displays an error message d
with the exception message.

```
    public static <T> void
populate_combobox(String sql,
JComboBox<T> jcb, JFrame obj_form) {
        ResultSet rs = null;
        jcb.removeAllItems();
        try (Connection conn =
getConnection()) {
            Statement st =
conn.createStatement();
            rs = st.executeQuery(sql);

            while (rs.next()) {
                T value = (T)
rs.getObject(1);
                jcb.addItem(value);
            }

        } catch (SQLException ex) {
```

```
19
20  JOptionPane.showMessageDialog(obj_form,
21  ex.getMessage(),
22                  "ERROR",
23  JOptionPane.ERROR_MESSAGE);
24          } finally {
25              try {
                    if (rs != null)
    rs.close();
                } catch (SQLException ex) {

    JOptionPane.showMessageDialog(obj_form,
    ex.getMessage(),
                        "ERROR",
    JOptionPane.ERROR_MESSAGE);
                }
            }
        }
```

Step 6

Then, add a new method named **table_renderer()**. This method is customize the rendering of a **JTable** object. The method takes a **JTable** obj input parameter, and sets the default cell renderer of the **JTable** object to renderer. The custom renderer overrides **getTableCellRendererComponent()** method of the **TableCellR** interface, and sets the background color of alternate rows to white and li to make it easier to read the data in the table.

The method achieves this by creating a new **TableCellRenderer** ob overriding its **getTableCellRendererComponent()** method. The method the row number is even or odd, and sets the background color of accordingly. If the row number is even, the background color is set to whi the row number is odd, the background color is set to light gray. The returns the component with the updated background color.

This method can be called after populating a **JTable** object to apply the cu rendering.

```
1       public static void table_renderer(JTable jtable){
2           jtable.setDefaultRenderer(Object.class,
3               new TableCellRenderer(){
4               private DefaultTableCellRenderer
5   DEFAULT_RENDERER = new DefaultTableCellRenderer();
6               @Override
7               public Component
8   getTableCellRendererComponent(JTable table,
9               Object value, boolean isSelected, boolean
10  hasFocus, int row, int column) {
11                  Component c =
12  DEFAULT_RENDERER.getTableCellRendererComponent(table,
```

```
                value, isSelected, hasFocus, row,
column);
                if (row%2 == 0){
                    c.setBackground(Color.WHITE);
                }
                else {

c.setBackground(Color.LIGHT_GRAY);
                }
                return c;
            }
        });
    }
```

Step 7

Then, add a new method named **set_column_header()**. This method column header for a **JTable**. The method takes two arguments: the **JTab** and a **String** array containing the column headers.
The method loops through the columns of the table using a for loop and column model of the table header for each column using the **getColumn** method. It then sets the header value of each column using the **setHeade** method and the corresponding value from the header array.

Finally, the method returns the same header array that was passed argument.

```
    //Sets columns header
    public static String[] set_column_header(JTable
table, String header[]) {
        for(int i=0;i<table.getColumnCount();i++)
        {
            TableColumn cols =
table.getTableHeader().getColumnModel().getColumn(i);
            cols.setHeaderValue(header[i]);
        }
        return header;
    }
```

Step 8

Still in **Utility** class, add a new method named **find_combo_value_selecte** method is used to find the selected item in a **JComboBox** and set it to the value if it exists. It takes two parameters, the **JComboBox** and the value for.

It first sets a boolean variable found to false to indicate that the specified not yet been found. Then it iterates over each item in the **JComboBox** us loop.

Inside the loop, it retrieves the item at the current index and compares specified value using the **Objects.equals()** method. If the item mat specified value, it sets the selected index of the **JComboBox** to the curr and sets found to true to indicate that the value was found. It then brea the loop since there is no need to continue iterating.

If the specified value is not found in the **JComboBox**, it sets the selected -1 to indicate that no item is selected.

```
    //Finds selected item in combobox
    public static <T> void
find_combo_value_selected(JComboBox<T>
comboBox, T value) {
        boolean found = false;
        for (int i = 0; i <
comboBox.getItemCount(); i++) {
            T item =
comboBox.getItemAt(i);
            if (Objects.equals(item,
value)) {

comboBox.setSelectedIndex(i);
                found = true;
                break;
            }
        }
        if (!found) {
        comboBox.setSelectedIndex(-1);
        }
    }
```

Step 9

Then, add a new method named **item_to_display()**. This method is used the pagination of data displayed in a form. It takes in the SQL stater getting the minimum and maximum values of the data, a text field w current page number is displayed, and a string indicating the dir pagination (first, previous, next, last).

The method first checks if the text field is empty or null. If it is, it re default page number of 1. Otherwise, it tries to parse the integer value o field and sets that as the current page number. If the parsing fails due to a integer format, an error message is displayed and the default page n returned.

Depending on the direction of pagination, the method executes the corre SQL statement to get the minimum and maximum values of the data, updates the current page number accordingly. If the direction is "first", i current page number to the minimum value. If it is "prev", it decren current page number by 1, or sets it to the maximum value if it is already

"next", it increments the current page number by 1, or sets it to 1 if it is a
the maximum value. If it is "last", it sets the current page number to the r
value.

If any errors occur during the SQL execution, an error message is displa
method then returns the updated current page number.

```
1                public    static    int
2  item_to_display(JFrame   frm,   String
3  sql_min,  String  sql_max,  JTextField
4  jtf, String direction) {
5          int currentValue = 1;
6
7       String text = jtf.getText();
8                  if  (text  ==  null  ||
9  text.trim().isEmpty()) {
10             return currentValue;
11         }
12
13         try {
14                           currentValue  =
15 Integer.parseInt(text);
16         } catch (NumberFormatException
17 ex) {
18
19 JOptionPane.showMessageDialog(frm,
20 ex.getMessage(),              "ERROR",
21 JOptionPane.ERROR_MESSAGE);
22             return currentValue;
23         }
24
25         switch (direction) {
26             case "first":
27                 try (Connection conn =
28 getConnection();
29                      PreparedStatement
30 ps = conn.prepareStatement(sql_min)) {
31                        ResultSet rs =
32 ps.executeQuery();
33                     if (rs.next()) {
34                         currentValue =
35 rs.getInt(1);
36                     }
37                 } catch (SQLException
38 ex) {
39
40 JOptionPane.showMessageDialog(frm,
41 ex.getMessage(),
42                                "ERROR",
43 JOptionPane.ERROR_MESSAGE);
44
45 }catch(java.lang.NumberFormatException
```

```
ex){

JOptionPane.showMessageDialog(frm,
ex.getMessage(),

 "ERROR",JOptionPane.ERROR_MESSAGE);
        }
                break;

            case "prev":
                try (Connection conn =
getConnection();
                        PreparedStatement
ps = conn.prepareStatement(sql_max)) {
                            ResultSet rs =
ps.executeQuery();
                    if (rs.next()) {
                                int max =
rs.getInt(1);
                                        if
(currentValue > 1) {

currentValue--;
                            } else {

currentValue = max;
                            }
                        }
                 } catch (SQLException
ex) {

JOptionPane.showMessageDialog(frm,
ex.getMessage(),
                                    "ERROR",
JOptionPane.ERROR_MESSAGE);

}catch(java.lang.NumberFormatException
ex){

JOptionPane.showMessageDialog(frm,
ex.getMessage(),

 "ERROR",JOptionPane.ERROR_MESSAGE);
        }
                break;

            case "next":
                try (Connection conn =
getConnection();
                        PreparedStatement
ps = conn.prepareStatement(sql_max)) {
                            ResultSet rs =
ps.executeQuery();
                    if (rs.next()) {
```

```
                                int max =
rs.getInt(1);
                                        if
(currentValue < max) {

currentValue++;
                            } else {

currentValue = 1;
                            }
                        }
                    } catch (SQLException
ex) {

JOptionPane.showMessageDialog(frm,
ex.getMessage(),
                                    "ERROR",
JOptionPane.ERROR_MESSAGE);

}catch(java.lang.NumberFormatException
ex){

JOptionPane.showMessageDialog(frm,
ex.getMessage(),

 "ERROR",JOptionPane.ERROR_MESSAGE);
        }
                break;

            case "last":
                try (Connection conn =
getConnection();
                        PreparedStatement
ps = conn.prepareStatement(sql_max)) {
                           ResultSet rs =
ps.executeQuery();
                    if (rs.next()) {
                        currentValue =
rs.getInt(1);
                    }
                } catch (SQLException
ex) {

JOptionPane.showMessageDialog(frm,
ex.getMessage(),
                                    "ERROR",
JOptionPane.ERROR_MESSAGE);

}catch(java.lang.NumberFormatException
ex){

JOptionPane.showMessageDialog(frm,
ex.getMessage(),
```

```
 "ERROR",JOptionPane.ERROR_MESSAGE);
        }
                break;
            default:
                break;
        }
        return currentValue;
    }
```

Step 10

Then, add a new method named **setLookAndFeel()**.This method sets the feel of a **JFrame** to the Nimbus look and feel, and then customizes the various UI elements. The Nimbus look and feel is a cross-platform look that was introduced in Java 6.

The method first sets the Nimbus look and feel **UIManager.setLookAndFeel()**. It then sets various UI color **UIManager.put()**. Finally, it updates the UI for the **JFram SwingUtilities.updateComponentTreeUI()**.

```
1   public static void setLookAndFeel(JFrame
2  frame) {
3           try {
4               // Set the Nimbus look and feel
5               UIManager.setLookAndFeel(new
6  NimbusLookAndFeel());
7
8               // Set the UI colors
9               UIManager.put("control", new
10 Color(240, 240, 240));
11              UIManager.put("info", new
12 Color(128, 128, 128));
13              UIManager.put("text", new
14 Color(50, 50, 50));
15              UIManager.put("nimbusBase", new
16 Color(250, 30, 49));
17
18 UIManager.put("nimbusAlertYellow", new
19 Color(248, 187, 0));
20
21 UIManager.put("nimbusDisabledText", new
22 Color(128, 128, 128));
23              UIManager.put("nimbusFocus", new
24 Color(115, 164, 209));
25              UIManager.put("nimbusGreen", new
26 Color(176, 179, 50));
27              UIManager.put("nimbusInfoBlue",
   new Color(66, 139, 221));

   UIManager.put("nimbusLightBackground", new
```

```
Color(220, 220, 220));
            UIManager.put("nimbusOrange",
new Color(191, 98, 4));
            UIManager.put("nimbusRed", new
Color(169, 46, 34));

UIManager.put("nimbusSelectedText", new
Color(255, 255, 255));

UIManager.put("nimbusSelectionBackground",
new Color(104, 93, 156));

            // Update the UI for the JFrame

SwingUtilities.updateComponentTreeUI(frame);
        } catch
(UnsupportedLookAndFeelException e) {
            System.err.println("Nimbus:
Unsupported Look and feel!");
        }
    }
```

Step 11

Then, add two new methods named **create_pie_dataset_from_categoriz** and **create_bar_dataset_from_categorized_vals()**.

These are two utility methods for creating datasets for pie charts and ba respectively. Both methods take a **Map<String, Double>** object, where represents a category and its associated value is the count for that categor

The **create_pie_dataset_from_categorized_vals()** creates a **DefaultPi** and populates it with the category names and counts from the input map.

The **create_bar_dataset_from_categorized_vals()** creates **DefaultCategoryDataset** and populates it with the category names ar from the input map, with "Count" as the series name for all categories.

```
1     public static DefaultPieDataset
2  create_pie_dataset_from_categorized_vals(Map<String,
3  Double> count_map) {
4         DefaultPieDataset dataset = new
5  DefaultPieDataset();
6         count_map.forEach((key, value) ->
7  dataset.setValue(key, value));
8         return dataset;
9     }
10
11    public static DefaultCategoryDataset
12 create_bar_dataset_from_categorized_vals(Map<String,
13 Double> count_map) {
```

```
        DefaultCategoryDataset dataset = new
DefaultCategoryDataset();
        for (Map.Entry<String, Double> entry :
count_map.entrySet()) {
            dataset.addValue(entry.getValue(),
"Count", entry.getKey());
        }
        return dataset;
    }
```

Step 12

Define two new methods named **draw_barchart_with_dataset** **draw_piechart_with_dataset()**. These two methods create and display ba charts respectively, using the **JFreeChart** library.

The **draw_barchart_with_dataset()** method takes in a **JFrame** object, object, a **DefaultCategoryDataset** object, chart title, x-axis label and y- as parameters. It removes all components from the **JPanel**, creates a **JF** object with the given parameters, sets its background color and c **ChartPanel** object with the chart. Finally, it adds the chart to the **JP** validates it.

The **draw_piechart_with_dataset()** method is similar, but it tak **DefaultPieDataset** object and a chart title as parameters. It creates a **JF** object with the given parameters, sets its background color and label gen show percentages, creates a **ChartPanel** object with the chart, and adds **JPanel**. Finally, it validates the **JPanel**.

Both methods catch any exceptions that may occur and display an error m a dialog box if necessary.

```
    public static void draw_barchart_with_dataset(JFrame frm, JPane
DefaultCategoryDataset dataset,
        String title, String xlabel, String ylabel){
        jp.removeAll();

        try{
            //Creates a JFreeChart object
            JFreeChart chartBar = ChartFactory.createBarChart(
                title, // chart title
                xlabel, // x-axis label
                ylabel, // y-axis label
                dataset, // dataset
                    PlotOrientation.HORIZONTAL, // orientation
            false, // include legend
            true, // tooltips
            false // urls
            );

```

```
19
20 chartBar.getRenderingHints().put(JFreeChart.KEY_SUPPRESS_SHADOW_GEN
21 Boolean.TRUE);
22
23 chartBar.getRenderingHints().put(RenderingHints.KEY_ANTIALIASING,
24 RenderingHints.VALUE_ANTIALIAS_ON);
25
26             //Sets background color 150
27             chartBar.setBackgroundPaint(new Color(200,250,200));
28             chartBar.getPlot().setBackgroundPaint(new Color(200, 21
29 250));
30
31             //Create a ChartPanel object
32             ChartPanel barchartPanel = new ChartPanel(chartBar);
33             jp.setLayout(new java.awt.BorderLayout());
34             jp.add(barchartPanel, BorderLayout.CENTER);
35
36             jp.validate();
37         }
38         catch (Exception ex){
39             JOptionPane.showMessageDialog(frm, ex.getMessage(),
40                     "ERROR",JOptionPane.ERROR_MESSAGE);
41         }
42     }
43
44     public static void draw_piechart_with_dataset(JFrame frm, JPane
45 DefaultPieDataset dataset, String title){
46         jp.removeAll();
47
48         try{
49             //Creates a JFreeChart object
50             JFreeChart chartPie = ChartFactory.createPieChart(
51                 title, // chart title
52                 dataset, // dataset
53             true, // include legend
54             true, // tooltips
55             false // urls
56             );
57
58             //Sets background color
59             chartPie.getPlot().setBackgroundPaint(new Color(150,200
60
61             // Sets the label generator to show percentages
62             PiePlot plot = (PiePlot) chartPie.getPlot();
63             plot.setLabelGenerator(new StandardPieSectionLabelGenerator("{0} (
64
65 NumberFormat.getNumberInstance(), new DecimalFormat("0.0%")));
66             //Create a ChartPanel object
67             ChartPanel piechartPanel = new ChartPanel(chartPie);
68             jp.setLayout(new java.awt.BorderLayout());
69             jp.add(piechartPanel, BorderLayout.CENTER);
70             jp.validate();
71         }
72
```

```
73          catch (Exception ex){
74              JOptionPane.showMessageDialog(frm, ex.getMessage(),
75                      "ERROR",JOptionPane.ERROR_MESSAGE);
            }
        }
```

Step 13

Define two new methods named **get_col_val_from_database** **get_val_from_database()**.
The **get_col_val_from_database()** takes two parameters, the **column** a names, and returns a list of values from the specified column in the specif It uses a SELECT statement to retrieve the data and then iterates thr **ResultSet** to add the column values to the list. It uses a generic type **T** to method to return a list of any type.

The **get_val_from_database takes()** four parameters: **tab** **columnName**, **condition**, and **item**. It retrieves a single value from a tab the specified condition matches the item value. It uses a **PreparedStatem** a SELECT statement that includes a WHERE clause to retrieve the c method sets the item value as a parameter in the prepared statement retrieves the result using a **ResultSet**. It returns the result object or matching row is found.

```
1       public static <T> List<T>
2   get_col_val_from_database(String
3   column, String table) throws
4   SQLException,
5   ClassNotFoundException {
6           List<T> values = new
7   ArrayList<>();
8           String query = "SELECT " +
9   column + " FROM " + table;
10
11          try(Connection conn =
12  getConnection();
13              Statement statement =
14  conn.createStatement();
15                  ResultSet resultSet
16  = statement.executeQuery(query)){
17
18              // Iterates through the
19  result set and add the column values to the
20  list
21              while
22  (resultSet.next()) {
23                  T value = (T)
24  resultSet.getObject(column);
25                  values.add(value);
26              }
27          }catch (SQLException ex){
```

```
28
29  JOptionPane.showMessageDialog(null,
30  ex.getMessage(),
31
32   "ERROR",JOptionPane.ERROR_MESSAGE)
33  ;
34          }
35          return values;
36      }
37
38      public static Object
39  get_val_from_database(String
40  tableName, String columnName,
41  String condition, Object item) {
42          Object result = null;
43          String query = "SELECT " +
44  columnName + " FROM " + tableName +
45  " WHERE " + condition + " = ?";
          try (Connection conn =
  getConnection();
               PreparedStatement ps =
  conn.prepareStatement(query)) {

               ps.setObject(1, item);
               try (ResultSet rs =
  ps.executeQuery()) {
                   if (rs.next()) {
                       result =
  rs.getObject(columnName);

                   } else {
                       // no row
  found, clear the form fields

  JOptionPane.showMessageDialog(null,
  "ITEM NOT FOUND");
                   }
               }
          } catch (SQLException ex) {

  JOptionPane.showMessageDialog(null,
  ex.getMessage(), "ERROR",
  JOptionPane.ERROR_MESSAGE);
          }
          return result;
      }
```

Step 14	Then, still in **Utility** class, define a new method named **categorize_valu** method takes a list of numerical values and a map that defines the ra different categories. It returns a map that contains the count of values within each category.

The method first initializes an empty map of counts for each category t
keys from the input map. Then, it loops through each value in the inpu
categorizes it into the corresponding range by comparing it to the valu
input map. If the value falls within a range, the corresponding category
that value. Finally, the count for the corresponding category is increment
counts map.

The method returns the counts map, which contains the count of values
category. The key of the map represents the category, and the value repre
count of values that fall within that category.

```
    public static Map<String, Double>
categorize_values(List<? extends
Number> list_vals, Map<String, ?
extends Number> range_vals) {
        Map<String, Double> counts =
 new HashMap<>();

        // Initialize the counts map
        for (String category :
range_vals.keySet()) {
            counts.put(category, 0.0);
        }

        // Loop through list_vals and
categorize them into their
corresponding range
        for (Number val : list_vals) {
            double dbValue =
val.doubleValue();
            String category =
range_vals.keySet().iterator().next();
// Default category

            for (Map.Entry<String, ?
extends Number> range :
range_vals.entrySet()) {
                if (dbValue <
range.getValue().doubleValue()) {
                    category =
range.getKey();
                    break;
                }
            }

            // Increment the count for
the corresponding category
            Double countObj =
counts.get(category);
```

```
            double count = countObj !=
null ? countObj.doubleValue() : 0;
            counts.put(category, count
+ 1);
        }

        return counts;
    }
```

Step 15

Define two new methods named **create_bar_dataset()** and **create_pie_d**
These are two utility methods that create a bar chart dataset and a
dataset from the results of an SQL query. Both methods take in a query
column name for the values, and a column name for the labels.

The **create_bar_dataset()** method uses a **DefaultCategoryDataset** ol
adds values to it using the **addValue()** method. The labels for the x-axis a
as the second parameter to this method, and the label for each data point
as the third parameter.

The **create_pie_dataset()** method uses a **DefaultPieDataset** object a
values to it using the **setValue()** method. The label for each data point is
the first parameter, and the value is added as the second parameter.

Both methods handle values of different types (**String** or **Number**) by che
type of the value and parsing it as a double if necessary. If the value is
String nor a **Number**, a **RuntimeException** is thrown.

Both methods also catch any exceptions that occur during the execution o
query and display an error message using a **JOptionPane**.

```
1   public static
2  DefaultCategoryDataset
3  create_bar_dataset(String query,
4  String valColumn, String
5  labelColumn){
6          // Creates a
7  DefaultCategoryDataset object
8          DefaultCategoryDataset
9  dataset = new
10 DefaultCategoryDataset();
11
12         try(Connection conn =
13 getConnection(); Statement stmt =
14 conn.createStatement();
15             ResultSet rs =
16 stmt.executeQuery(query)){
17
18             while (rs.next()) {
```

```
                Object value =
rs.getObject(valColumn);
                String label =
rs.getString(labelColumn);

                if (value
instanceof Number) {

dataset.addValue(((Number)
value).doubleValue(), labelColumn,
label);
                } else if (value
instanceof String) {
                    double
doubleValue =
Double.parseDouble((String) value);

dataset.addValue(doubleValue,
labelColumn, label);
                } else {
                    throw new
RuntimeException("Unsupported value
type: " +
value.getClass().getName());
                }
            }
        } catch (Exception ex){

JOptionPane.showMessageDialog(null,
ex.getMessage(),

 "ERROR",JOptionPane.ERROR_MESSAGE)
;
        }
        return dataset;
    }

    public static DefaultPieDataset
create_pie_dataset(String query,
String valColumn, String
labelColumn){
        // Creates a
DefaultPieDataset object
        DefaultPieDataset dataset =
new DefaultPieDataset();

        try(Connection conn =
getConnection(); Statement stmt =
conn.createStatement();
            ResultSet rs =
stmt.executeQuery(query)){

            while (rs.next()) {
```

```
                Object value =
rs.getObject(valColumn);
                String label =
rs.getString(labelColumn);

                if (value
instanceof Number) {

dataset.setValue(label, ((Number)
value).doubleValue());
                } else if (value
instanceof String) {
                    double
doubleValue =
Double.parseDouble((String) value);

dataset.setValue(label,
doubleValue);
                } else {
                    throw new
RuntimeException("Unsupported value
type: " +
value.getClass().getName());
                }
            }
        } catch (Exception ex){

JOptionPane.showMessageDialog(null,
ex.getMessage(),

 "ERROR",JOptionPane.ERROR_MESSAGE)
;
        }
        return dataset;
    }
```

Step 16

Then, define two new methods named **read_image()** and **show_image** two methods are related to displaying images on a Swing-based user interf

The **read_image()** method allows the user to select an image file u **JFileChooser** dialog box and then display it on a given **JLabel** control three parameters:

- **jframe**: a **JFrame** instance which is used as the parent of the file dialog box
- **jtfPhotoPath**: a **JTextField** instance that will display the absolut the selected image file
- **label**: a **JLabel** instance which is used to display the selected imag

The method first creates a **JFileChooser** object and sets the current dir the user's home directory. It then shows the dialog box and waits for th select a file. If the user chooses a valid image file that can be read, th reads the image file using the **ImageIO.read()** method and displays it on **JLabel** control. It also sets the text of the given **JTextField** control to the path of the selected image file. If the selected file is not a valid image file be read, the method shows an error message on a message dialog box a the **JLabel** and **JTextField** controls.

The **show_image()** method takes two parameters:

- **label**: a **JLabel** instance which is used to display the image
- **path**: a **String** containing the path of the image file to be displaye

This method simply loads the image from the given file path and sets it as of the given **JLabel** control using the ImageIcon class.

```
1     public void read_image(JFrame jframe, JTextField
2 jtfPhotoPath, JLabel label) {
3         BufferedImage im = null;
4         try {
5             JFileChooser fileChooser = new JFileChooser();
6             fileChooser.setCurrentDirectory(new
7 File(System.getProperty("user.home")));
8             int result = fileChooser.showOpenDialog(jframe);
9             if (result == JFileChooser.APPROVE_OPTION) {
10                File fileChosen = fileChooser.getSelectedFile();
11                if (fileChosen.isFile() && fileChosen.canRead())
12 {
13                    im = ImageIO.read(fileChosen);
14
15 jtfPhotoPath.setText(fileChosen.getAbsolutePath());
16
17                    // Display image on jLabelImage

18                    ImageIcon icon = new ImageIcon(im);
19                    label.setIcon(icon);
20                } else {
21                    throw new IOException("Selected file is not
22 a valid image file or cannot be read.");
23                }
24            }
25        } catch (IOException e) {
26            JOptionPane.showMessageDialog(jframe, "Error reading
27 image file: " + e.getMessage(),
28                "Error", JOptionPane.ERROR_MESSAGE);
29            label.setIcon(null);
30            jtfPhotoPath.setText("");
31        } catch (Exception e) {
32            ImageIcon icon = new
33 ImageIcon(getClass().getClassLoader().getResource("dummy.png"));
34            label.setIcon(icon);
35        }
```

```
    }

    public static void show_image(JLabel label, String path) {
        // Display image on JLabel control
        ImageIcon icon = new ImageIcon(path);
        label.setIcon(icon);
    }
```

Step 17

Next, define a new method named **findMaxInt()**. This method is used to maximum integer value in a specific column of a MySQL table. It takes parameters: the connection object to the database, the name of the table name of the column to search for the maximum value in.

Inside the method, an initial value of the maximum value is set to the possible integer value. The method then constructs a SQL query string to maximum value in the specified column of the specified table. The query executed using a Statement object, and the resulting **ResultSet** is proc obtain the maximum value found.

If there are no results in the **ResultSet**, the initial minimum value set at of the method is returned. Otherwise, the maximum value found in the **R** is returned.

```
    public static int
findMaxInt(Connection conn, String
tableName, String columnName)
throws SQLException {
        int max =
Integer.MIN_VALUE;
        String query = "SELECT
MAX(" + columnName + ") FROM " +
tableName;
        try (Statement stmt =
conn.createStatement();
             ResultSet rs =
stmt.executeQuery(query)) {
             if (rs.next()) {
                 max = rs.getInt(1);
             }
        }
        return max;
    }
```

Step 18

Then, define an overloaded **read_image()** method with only two paramete

```
    public void read_image(JFrame jframe, JLabel label) {
        BufferedImage im = null;
```

```
        try {
            JFileChooser fileChooser = new JFileChooser();
            fileChooser.setCurrentDirectory(new
File(System.getProperty("user.home")));
            int result = fileChooser.showOpenDialog(jframe);
            if (result == JFileChooser.APPROVE_OPTION) {
                File fileChosen = fileChooser.getSelectedFile();
                if (fileChosen.isFile() && fileChosen.canRead())
{
                    im = ImageIO.read(fileChosen);

                    // Display image on jLabelImage
                    ImageIcon icon = new ImageIcon(im);
                    label.setIcon(icon);
                } else {
                    throw new IOException("Selected file is not
a valid image file or cannot be read.");
                }
            }
        } catch (IOException e) {
            JOptionPane.showMessageDialog(jframe, "Error reading
image file: " + e.getMessage(),
                    "Error", JOptionPane.ERROR_MESSAGE);
            label.setIcon(null);
        } catch (Exception e) {
            ImageIcon icon = new
ImageIcon(getClass().getClassLoader().getResource("dummy.png"));
            label.setIcon(icon);
        }
    }
```

This is the full version of **Utility.java**:

```
package Sakila;
import java.io.FileInputStream;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.util.Properties;
import java.util.logging.Level;
import java.util.logging.Logger;
import java.sql.*;
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Component;
import java.awt.RenderingHints;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
import javax.swing.JOptionPane;
import java.text.DecimalFormat;
```

```
import java.text.NumberFormat;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Objects;
import javax.imageio.ImageIO;
import javax.swing.ImageIcon;
import javax.swing.JComboBox;
import javax.swing.JFileChooser;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JTable;
import javax.swing.JTextField;
import javax.swing.SwingUtilities;
import javax.swing.UIManager;
import javax.swing.UnsupportedLookAndFeelException;
import javax.swing.plaf.nimbus.NimbusLookAndFeel;
import javax.swing.table.DefaultTableCellRenderer;
import javax.swing.table.TableCellRenderer;
import javax.swing.table.TableColumn;
import org.jfree.chart.ChartFactory;
import org.jfree.chart.ChartPanel;
import org.jfree.chart.JFreeChart;
import org.jfree.chart.labels.ItemLabelAnchor;
import org.jfree.chart.labels.ItemLabelPosition;
import
org.jfree.chart.labels.StandardCategoryItemLabelGenerator;
import
org.jfree.chart.labels.StandardPieSectionLabelGenerator;
import org.jfree.chart.plot.CategoryPlot;
import org.jfree.chart.plot.PiePlot;
import org.jfree.chart.plot.PlotOrientation;
import
org.jfree.chart.renderer.category.CategoryItemRenderer;
import org.jfree.chart.ui.TextAnchor;
import org.jfree.data.category.DefaultCategoryDataset;
import org.jfree.data.general.DefaultPieDataset;

public class Utility {
    private static final Logger LOGGER =
Logger.getLogger(Utility.class.getName());

    public static Connection getConnection() throws
SQLException {
        Connection conn = null;
        Properties props = new Properties();
        try (FileInputStream fis = new
FileInputStream("database.properties")) {
            props.load(fis);
            String url = props.getProperty("url");
            String user = props.getProperty("user");
```

```
            String password =
props.getProperty("password");
            conn = DriverManager.getConnection(url, user,
password);
        } catch (IOException ex) {
            LOGGER.log(Level.SEVERE, null, ex);
        }
        return conn;
    }

    public static void testConnection() {
        try (Connection conn = getConnection()) {
            String message = String.format("Successfully
connected to %s database", conn.getCatalog());
            LOGGER.log(Level.INFO, message);
        } catch (SQLException ex) {
            LOGGER.log(Level.SEVERE, null, ex);
        }
    }

    public static <T> void populate_combobox(String sql,
JComboBox<T> jcb, JFrame obj_form) {
        ResultSet rs = null;
        jcb.removeAllItems();
        try (Connection conn = getConnection()) {
            Statement st = conn.createStatement();
            rs = st.executeQuery(sql);

            while (rs.next()) {
                T value = (T) rs.getObject(1);
                jcb.addItem(value);
            }

        } catch (SQLException ex) {
            JOptionPane.showMessageDialog(obj_form,
ex.getMessage(),
                "ERROR", JOptionPane.ERROR_MESSAGE);
        } finally {
            try {
                if (rs != null) rs.close();
            } catch (SQLException ex) {
                JOptionPane.showMessageDialog(obj_form,
ex.getMessage(),
                    "ERROR", JOptionPane.ERROR_MESSAGE);
            }
        }
    }

    public static void table_renderer(JTable jtable){
        jtable.setDefaultRenderer(Object.class,
            new TableCellRenderer(){
            private DefaultTableCellRenderer
DEFAULT_RENDERER = new DefaultTableCellRenderer();
```

```
            @Override
            public Component
getTableCellRendererComponent(JTable table,
            Object value, boolean isSelected, boolean
hasFocus, int row, int column) {
                Component c =
DEFAULT_RENDERER.getTableCellRendererComponent(table,
                value, isSelected, hasFocus, row, column);
                if (row%2 == 0){
                    c.setBackground(Color.WHITE);
                }
                else {
                    c.setBackground(Color.LIGHT_GRAY);
                }
                return c;
            }
        });
    }

    //Sets columns header
    public static String[] set_column_header(JTable table,
String header[]) {
        for(int i=0;i<table.getColumnCount();i++)
        {
            TableColumn cols =
table.getTableHeader().getColumnModel().getColumn(i);
            cols.setHeaderValue(header[i]);
        }
        return header;
    }

    //Finds selected item in combobox
    public static <T> void
find_combo_value_selected(JComboBox<T> comboBox, T value)
{
        boolean found = false;
        for (int i = 0; i < comboBox.getItemCount(); i++)
{
            T item = comboBox.getItemAt(i);
            if (Objects.equals(item, value)) {
                comboBox.setSelectedIndex(i);
                found = true;
                break;
            }
        }
        if (!found) {
        comboBox.setSelectedIndex(-1);
        }
    }

    public static int item_to_display(JFrame frm, String
sql_min, String sql_max, JTextField jtf, String direction)
{
```

```
        int currentValue = 1;

    String text = jtf.getText();
        if (text == null || text.trim().isEmpty()) {
            return currentValue;
        }

        try {
            currentValue = Integer.parseInt(text);
        } catch (NumberFormatException ex) {
            JOptionPane.showMessageDialog(frm,
ex.getMessage(), "ERROR", JOptionPane.ERROR_MESSAGE);
            return currentValue;
        }

        switch (direction) {
            case "first":
                try (Connection conn = getConnection();
                    PreparedStatement ps =
conn.prepareStatement(sql_min)) {
                    ResultSet rs = ps.executeQuery();
                    if (rs.next()) {
                        currentValue = rs.getInt(1);
                    }
                } catch (SQLException ex) {
                    JOptionPane.showMessageDialog(frm,
ex.getMessage(),
                    "ERROR", JOptionPane.ERROR_MESSAGE);
```

```
                }catch(java.lang.NumberFormatException ex){
                    JOptionPane.showMessageDialog(frm, ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
        }
                break;

            case "prev":
                try (Connection conn = getConnection();
                    PreparedStatement ps = conn.prepareStatement(sql_max))
{
                    ResultSet rs = ps.executeQuery();
                    if (rs.next()) {
                        int max = rs.getInt(1);
                        if (currentValue > 1) {
                            currentValue--;
                        } else {
                            currentValue = max;
                        }
                    }
                } catch (SQLException ex) {
                    JOptionPane.showMessageDialog(frm, ex.getMessage(),
                    "ERROR", JOptionPane.ERROR_MESSAGE);
                }catch(java.lang.NumberFormatException ex){
                    JOptionPane.showMessageDialog(frm, ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
        }
                break;

            case "next":
                try (Connection conn = getConnection();
                    PreparedStatement ps = conn.prepareStatement(sql_max))
{
                    ResultSet rs = ps.executeQuery();
                    if (rs.next()) {
                        int max = rs.getInt(1);
                        if (currentValue < max) {
                            currentValue++;
                        } else {
                            currentValue = 1;
                        }
                    }
                } catch (SQLException ex) {
                    JOptionPane.showMessageDialog(frm, ex.getMessage(),
                    "ERROR", JOptionPane.ERROR_MESSAGE);
                }catch(java.lang.NumberFormatException ex){
                    JOptionPane.showMessageDialog(frm, ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
        }
                break;

            case "last":
                try (Connection conn = getConnection();
```

```
                    PreparedStatement ps = conn.prepareStatement(sql_max))
{
                    ResultSet rs = ps.executeQuery();
                    if (rs.next()) {
                        currentValue = rs.getInt(1);
                    }
                } catch (SQLException ex) {
                    JOptionPane.showMessageDialog(frm, ex.getMessage(),
                "ERROR", JOptionPane.ERROR_MESSAGE);
                }catch(java.lang.NumberFormatException ex){
                    JOptionPane.showMessageDialog(frm, ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
        }
                break;
            default:
                break;
        }
        return currentValue;
    }

    /**
    * Sets the Nimbus look and feel for a JFrame.
    * @param frame the JFrame to set the look and feel for
    */
    public static void setLookAndFeel(JFrame frame) {
        try {
            // Set the Nimbus look and feel
            UIManager.setLookAndFeel(new NimbusLookAndFeel());

            // Set the UI colors
            UIManager.put("control", new Color(240, 240, 240));
            UIManager.put("info", new Color(128, 128, 128));
            UIManager.put("text", new Color(50, 50, 50));
            UIManager.put("nimbusBase", new Color(50, 200, 20));
            UIManager.put("nimbusAlertYellow", new Color(248, 187, 0));
            UIManager.put("nimbusDisabledText", new Color(128, 128, 128));
            UIManager.put("nimbusFocus", new Color(115, 164, 209));
            UIManager.put("nimbusGreen", new Color(176, 179, 50));
            UIManager.put("nimbusInfoBlue", new Color(66, 139, 221));
            UIManager.put("nimbusLightBackground", new Color(220, 220,
220));
            UIManager.put("nimbusOrange", new Color(191, 98, 4));
            UIManager.put("nimbusRed", new Color(169, 46, 34));
            UIManager.put("nimbusSelectedText", new Color(255, 255, 255));
            UIManager.put("nimbusSelectionBackground", new Color(104, 93,
156));

            // Update the UI for the JFrame
            SwingUtilities.updateComponentTreeUI(frame);
        } catch (UnsupportedLookAndFeelException e) {
            System.err.println("Nimbus: Unsupported Look and feel!");
        }
    }
```

```
    public static DefaultPieDataset
create_pie_dataset_from_categorized_vals(Map<String, Double> count_map) {
        DefaultPieDataset dataset = new DefaultPieDataset();
        count_map.forEach((key, value) -> dataset.setValue(key, value));
        return dataset;
    }

    public static DefaultCategoryDataset
create_bar_dataset_from_categorized_vals(Map<String, Double> count_map) {
        DefaultCategoryDataset dataset = new DefaultCategoryDataset();
        for (Map.Entry<String, Double> entry : count_map.entrySet()) {
            dataset.addValue(entry.getValue(), "Count", entry.getKey());
        }
        return dataset;
    }

    public static void draw_barchart_with_dataset(JFrame frm, JPanel jp,
DefaultCategoryDataset dataset,
        String title, String xlabel, String ylabel){
        jp.removeAll();

        try{
            //Creates a JFreeChart object
            JFreeChart chartBar = ChartFactory.createBarChart(
                title, // chart title
                xlabel, // x-axis label
                ylabel, // y-axis label
                dataset, // dataset
                    PlotOrientation.HORIZONTAL, // orientation
            false, // include legend
            true, // tooltips
            false // urls
            );


chartBar.getRenderingHints().put(JFreeChart.KEY_SUPPRESS_SHADOW_GENERATION,
Boolean.TRUE);

chartBar.getRenderingHints().put(RenderingHints.KEY_ANTIALIASING,
RenderingHints.VALUE_ANTIALIAS_ON);

            //Displays value on top of each bar
            CategoryPlot plot = (CategoryPlot) chartBar.getPlot();
            CategoryItemRenderer renderer = plot.getRenderer();
            renderer.setDefaultItemLabelGenerator(new
StandardCategoryItemLabelGenerator());
            renderer.setDefaultItemLabelsVisible(true);
            ItemLabelPosition position = new
ItemLabelPosition(ItemLabelAnchor.CENTER, TextAnchor.CENTER);
            renderer.setDefaultPositiveItemLabelPosition(position);

            //Sets background color 150
```

```
            chartBar.setBackgroundPaint(new Color(250,190,250));
            chartBar.getPlot().setBackgroundPaint(new Color(0, 216, 250));

            //Create a ChartPanel object
            ChartPanel barchartPanel = new ChartPanel(chartBar);
            jp.setLayout(new java.awt.BorderLayout());
            jp.add(barchartPanel, BorderLayout.CENTER);

            jp.validate();
        }
        catch (Exception ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    public static void draw_piechart_with_dataset(JFrame frm, JPanel jp,
DefaultPieDataset dataset, String title){
        jp.removeAll();

        try{
            //Creates a JFreeChart object
            JFreeChart chartPie = ChartFactory.createPieChart(
                title, // chart title
                dataset, // dataset
            true, // include legend
            true, // tooltips
            false // urls
            );

            //Sets background color
            chartPie.getPlot().setBackgroundPaint(new Color(150,200,250));

            // Sets the label generator to show percentages
            PiePlot plot = (PiePlot) chartPie.getPlot();
            plot.setLabelGenerator(new StandardPieSectionLabelGenerator("
{0} ({2})",

NumberFormat.getNumberInstance(),
                                                                    new
DecimalFormat("0.0%")));
            //Create a ChartPanel object
            ChartPanel piechartPanel = new ChartPanel(chartPie);
            jp.setLayout(new java.awt.BorderLayout());
            jp.add(piechartPanel, BorderLayout.CENTER);
            jp.validate();
        }
        catch (Exception ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }
```

```
    public static <T> List<T> get_col_val_from_database(String column,
String table) throws SQLException, ClassNotFoundException {
        List<T> values = new ArrayList<>();
        String query = "SELECT " + column + " FROM " + table;

        try(Connection conn = getConnection();
            Statement statement = conn.createStatement();
                ResultSet resultSet = statement.executeQuery(query)){

            // Iterates through the result set and add the column values to
the list
            while (resultSet.next()) {
                T value = (T) resultSet.getObject(column);
                values.add(value);
            }
        }catch (SQLException ex){
            JOptionPane.showMessageDialog(null, ex.getMessage(),
                "ERROR",JOptionPane.ERROR_MESSAGE);
        }
        return values;
    }

    public static Object get_val_from_database(String tableName, String
columnName, String condition, Object item) {
        Object result = null;
        String query = "SELECT " + columnName + " FROM " + tableName + "
WHERE " + condition + " = ?";
        try (Connection conn = getConnection();
            PreparedStatement ps = conn.prepareStatement(query)) {

            ps.setObject(1, item);
            try (ResultSet rs = ps.executeQuery()) {
                if (rs.next()) {
                    result = rs.getObject(columnName);

                } else {
                    // no row found, clear the form fields
                    JOptionPane.showMessageDialog(null, "ITEM NOT FOUND");
                }
            }
        } catch (SQLException ex) {
            JOptionPane.showMessageDialog(null, ex.getMessage(), "ERROR",
JOptionPane.ERROR_MESSAGE);
        }
        return result;
    }

    public static Map<String, Double> categorize_values(List<? extends
Number> list_vals, Map<String, ? extends Number> range_vals) {
        Map<String, Double> counts = new HashMap<>();

        // Initialize the counts map
        for (String category : range_vals.keySet()) {
```

```
            counts.put(category, 0.0);
        }

        // Loop through list_vals and categorize them into their
corresponding range
        for (Number val : list_vals) {
            double dbValue = val.doubleValue();
            String category = range_vals.keySet().iterator().next(); //
Default category

            for (Map.Entry<String, ? extends Number> range :
range_vals.entrySet()) {
                if (dbValue < range.getValue().doubleValue()) {
                    category = range.getKey();
                    break;
                }
            }

            // Increment the count for the corresponding category
            Double countObj = counts.get(category);
            double count = countObj != null ? countObj.doubleValue() : 0;
            counts.put(category, count + 1);
        }

        return counts;
    }

    public static DefaultCategoryDataset create_bar_dataset(String query,
String valColumn, String labelColumn){
        // Creates a DefaultCategoryDataset object
        DefaultCategoryDataset dataset = new DefaultCategoryDataset();

        try(Connection conn = getConnection(); Statement stmt =
conn.createStatement();
            ResultSet rs = stmt.executeQuery(query)){

            while (rs.next()) {
                Object value = rs.getObject(valColumn);
                String label = rs.getString(labelColumn);

                if (value instanceof Number) {
                    dataset.addValue(((Number) value).doubleValue(),
labelColumn, label);
                } else if (value instanceof String) {
                    double doubleValue = Double.parseDouble((String)
value);
                    dataset.addValue(doubleValue, labelColumn, label);
                } else {
                    throw new RuntimeException("Unsupported value type: " +
value.getClass().getName());
                }
            }
        } catch (Exception ex){
```

```
            JOptionPane.showMessageDialog(null, ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
        }
        return dataset;
    }

    public static DefaultPieDataset create_pie_dataset(String query, String
valColumn, String labelColumn){
        // Creates a DefaultPieDataset object
        DefaultPieDataset dataset = new DefaultPieDataset();

        try(Connection conn = getConnection(); Statement stmt =
conn.createStatement();
            ResultSet rs = stmt.executeQuery(query)){

            while (rs.next()) {
                Object value = rs.getObject(valColumn);
                String label = rs.getString(labelColumn);

                if (value instanceof Number) {
                    dataset.setValue(label, ((Number)
value).doubleValue());
                } else if (value instanceof String) {
                    double doubleValue = Double.parseDouble((String)
value);
                    dataset.setValue(label, doubleValue);
                } else {
                    throw new RuntimeException("Unsupported value type: " +
value.getClass().getName());
                }
            }
        } catch (Exception ex){
            JOptionPane.showMessageDialog(null, ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
        }
        return dataset;
    }

    public void read_image(JFrame jframe, JTextField jtfPhotoPath, JLabel
label) {
        BufferedImage im = null;
        try {
            JFileChooser fileChooser = new JFileChooser();
            fileChooser.setCurrentDirectory(new
File(System.getProperty("user.home")));
            int result = fileChooser.showOpenDialog(jframe);
            if (result == JFileChooser.APPROVE_OPTION) {
                File fileChosen = fileChooser.getSelectedFile();
                if (fileChosen.isFile() && fileChosen.canRead()) {
                    im = ImageIO.read(fileChosen);
                    jtfPhotoPath.setText(fileChosen.getAbsolutePath());

                    // Display image on jLabelImage
```

```
                    ImageIcon icon = new ImageIcon(im);
                    label.setIcon(icon);
                } else {
                    throw new IOException("Selected file is not a valid
image file or cannot be read.");
                }
            }
        } catch (IOException e) {
            JOptionPane.showMessageDialog(jframe, "Error reading image
file: " + e.getMessage(),
                "Error", JOptionPane.ERROR_MESSAGE);
            label.setIcon(null);
            jtfPhotoPath.setText("");
        } catch (Exception e) {
            ImageIcon icon = new
ImageIcon(getClass().getClassLoader().getResource("dummy.png"));
            label.setIcon(icon);
        }
    }

    public static void show_image(JLabel label, String path) {
        // Display image on JLabel control
        ImageIcon icon = new ImageIcon(path);
        label.setIcon(icon);
    }

    public static int findMaxInt(Connection conn, String tableName, String
columnName) throws SQLException {
        int max = Integer.MIN_VALUE;
        String query = "SELECT MAX(" + columnName + ") FROM " + tableName;
        try (Statement stmt = conn.createStatement();
            ResultSet rs = stmt.executeQuery(query)) {
            if (rs.next()) {
                max = rs.getInt(1);
            }
        }
        return max;
    }

    public void read_image(JFrame jframe, JLabel label) {
        BufferedImage im = null;
        try {
            JFileChooser fileChooser = new JFileChooser();
            fileChooser.setCurrentDirectory(new
File(System.getProperty("user.home")));
            int result = fileChooser.showOpenDialog(jframe);
            if (result == JFileChooser.APPROVE_OPTION) {
                File fileChosen = fileChooser.getSelectedFile();
                if (fileChosen.isFile() && fileChosen.canRead()) {
                    im = ImageIO.read(fileChosen);

                    // Display image on jLabelImage
                    ImageIcon icon = new ImageIcon(im);
```

```
                    label.setIcon(icon);
                } else {
                    throw new IOException("Selected file is not a valid image file or
cannot be read.");
                }
            }
        } catch (IOException e) {
            JOptionPane.showMessageDialog(jframe, "Error reading image
file: " + e.getMessage(),
                    "Error", JOptionPane.ERROR_MESSAGE);
            label.setIcon(null);
        } catch (Exception e) {
            ImageIcon icon = new
ImageIcon(getClass().getClassLoader().getResource("dummy.png"));
            label.setIcon(icon);
        }
    }
}
```

ACTOR FORM ACTOR FORM

CREATING AND POPULATING ACTOR TABLE

CREATING AND POPULATING ACTOR TABLE

Step 1	Create a new class named **Query_Actor**. It contains several constants representing SQL queries and a SQL table creation statement. The first four constants represent SQL queries for selecting actors from actor table in a database. The first query **sql_min** selects the minimum **actor_id** value from the **actor** table. The second query **sql_max** selects the maximum **actor_id** value from the **actor** table. The third **query sql_id** selects all columns from the **actor** table where the **actor_id** column matches a parameter (denoted by ?). The fourth **query sql_name** selects all columns from the **actor** table where the concatenation of the **first_name** and **last_name** columns matches a parameter (also denoted by ?).

The last constant **sql_actor** is a SQL statement to create a new **actor** table if it does not exist in the database. This statement defines the structure of the **actor** table with columns for **actor_id**, **first_name**, **last_name**, and **last_update**, and specifies that the **actor_id** column is the primary key and the **last_name** column is indexed.
The class also includes several getter methods for accessing these SQL queries as static constants.

```
1  package sakila;
2
3  public class Query_Actor {
4      private static final String
5  sql_min = "SELECT MIN(actor_id)
6  FROM actor";
7      private static final String
8  sql_max = "SELECT MAX(actor_id)
9  FROM actor";
10     private static final String sql_id =
11 "SELECT * FROM actor WHERE actor_id  = ?";
12     private static final String
13 sql_name = "SELECT * FROM actor
14 WHERE CONCAT(first_name,' ',
15 last_name) = ?";
16
17
18     private static final String
19 sql_actor = """
20         CREATE TABLE IF NOT EXISTS
21 actor(
22             actor_id SMALLINT
23 UNSIGNED NOT NULL AUTO_INCREMENT,
24             first_name VARCHAR(50)
25 NOT NULL,
26             last_name VARCHAR(50)
27 NOT NULL,
28             last_update TIMESTAMP
29 NOT NULL DEFAULT CURRENT_TIMESTAMP
30 ON UPDATE CURRENT_TIMESTAMP,
31             PRIMARY KEY(actor_id),
32             KEY
33 idx_actor_lname(last_name));""";
34
35     //Getter methods
36     public static String
37 get_sql_min() {
38         return sql_min;
39     }
40
41
```

```
    public static String
get_sql_max() {
        return sql_max;
    }

    public static String
get_sql_id() {
        return sql_id;
    }

    public static String
get_sql_name() {
        return sql_name;
    }

    public static String
get_sql_actor() {
        return sql_actor;
    }
}
```

Step 2

Then, create a public class named **Actor** in which each instance variable represents each column in **actor** table in the database. It has four instance variables: **actor_id**, **fname**, **lname**, and **last_update**, which respectively represent the actor's ID, first name, last name, and the date and time of the last update to their record.

The class has three constructors, each with different numbers of parameters. The default constructor initializes the instance variables to default values. The two other constructors take in different numbers of parameters and initialize the instance variables accordingly.

The class has getter and setter methods for each instance variable. The setter methods validate the input parameters and throw an **IllegalArgumentException** if they are invalid.

- **public void setActorID(int id)** - This method sets the value of the **actor_id** instance variable to the given id value. It first checks if the id value is greater than zero. If it is not, it throws an **IllegalArgumentException**.
- **public void setFirstName(String name)** - This method sets the value of the **fname**

instance variable to the given name value. It first checks if the name value is not null or empty, and if it is longer than 45 characters. If any of these checks fail, it throws an **IllegalArgumentException**.

- **public void setLastName(String name)** - This method sets the value of the **lname** instance variable to the given name value. It first checks if the name value is not null or empty, and if it is longer than 45 characters. If any of these checks fail, it throws an **IllegalArgumentException**.
- **public void setLastUpdate(Timestamp date)** - This method sets the value of the **last_update** instance variable to the given date value. It first checks if the date value is not null. If it is null, it throws an **IllegalArgumentException**.

The class also overrides the **hashCode()**, **equals()**, and **toString()** methods inherited from the **Object** class to provide more meaningful behavior for comparing and printing Actor objects.

```
1  package sakila;
2  import java.util.Objects;
3  import java.sql.Timestamp;
4
5  public class Actor {
6      private int actor_id;
7      private String fname;
8      private String lname;
9      private Timestamp last_update;
10
11     //Default constructor
12     Actor(){
13         this(1, "First Name", "Last
14 Name", new
15 Timestamp(System.currentTimeMillis()));
16     }
17
18     //Three-params constructor
19     Actor(String fname, String lname,
20 Timestamp lu){
21         setFirstName(fname);
22         setLastName(lname);
23         setLastUpdate(lu);
24     }
25
26     //Four-params constructor
```

```
27      Actor(int id, String fname, String
28  lname, Timestamp lu){
29          this(fname, lname, lu);
30          setActorID(id);
31      }
32
33      // Getter methods
34      public int getActorID() {return
35  actor_id;}
36      public String getFirstName()
37  {return fname;}
38      public String getLastName() {return
39  lname;}
40      public Timestamp getLastUpdate()
41  {return last_update;}
42
43      //Setter methods
44      public void setActorID(int id) {
45          if (id <= 0) {
46              throw new
47  IllegalArgumentException("Actor ID must be greater
48  than zero.");
49          }
50          this.actor_id = id;
51      }
52
53
54      public void setFirstName(String
55  name) {
56          if (name == null ||
57  name.trim().isEmpty()) {
58              throw new
59  IllegalArgumentException("First name cannot be null
60  or empty");
61          }
62          if (name.length() > 45) {
63              throw new
64  IllegalArgumentException("First name
65  cannot be longer than 45 characters");
66          }
67          this.fname = name;
68      }
69
70      public void setLastName(String
71  name) {
72          if (name == null ||
73  name.trim().isEmpty()) {
74              throw new
75  IllegalArgumentException("Last name cannot be null
76  or empty");
77          }
78          if (name.length() > 45) {
79
80
```

```
            throw new
IllegalArgumentException("Last name
cannot be longer than 45 characters");
        }
        this.lname = name;
    }

    public void setLastUpdate(Timestamp
date){
        if (date == null) {
        throw new
IllegalArgumentException("Date cannot
be null");
        }
        this.last_update = date;
    }

    // Override the hashCode() method
    @Override
    public int hashCode() {
        return Objects.hash(actor_id,
fname, lname, last_update);
    }

    // Override the equals() method
    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() !=
o.getClass()) return false;
        Actor act = (Actor) o;
        return actor_id == act.actor_id
&&
                Objects.equals(fname,
act.fname) &&
                Objects.equals(lname,
act.lname) &&

Objects.equals(last_update,
act.last_update);
    }

    @Override
    public String toString(){
        return "\nActor ID      : " +
getActorID() +
                "\nFull Name     : " +
getFirstName() + " " + getLastName() +
                "\nLast Update   : " +
getLastUpdate();
    }
}
```

Step 3	Create a new public class named **Actor_Utils**. It extends the **Utility** class. The purpose of this class is to provide utility functions for working with the **actor** table in a database. The class has several static methods that can be used to create and populate the **actor** table, as well as read its contents. The **create_actor_table()** method creates the actor table in the database by executing an SQL statement obtained from the **Query_Actor** class. If the table is successfully created, a message is displayed to the user. If an error occurs, an error message is displayed. The **populate_actor_table()** method populates the actor table with some sample data by creating **Actor** objects and inserting them into the table using prepared statements. If an error occurs, an error message is displayed. The **read_actor_table()** method reads the contents of the **actor** table by executing a select statement and iterating over the result set. For each row in the result set, an Actor object is created and printed to the console. The class also defines several constants, including FIRST_INDEX, INVALID_INDEX, and SQL_ID, which are used throughout the class. The **currentIndex** variable is used to keep track of the current index when iterating over the **actor** table.

```
1   package sakila;
2   import java.awt.Dimension;
3   import java.awt.image.BufferedImage;
4   import java.io.ByteArrayOutputStream;
5   import java.io.IOException;
6   import java.net.URL;
7   import java.util.logging.Level;
8   import java.util.logging.Logger;
9   import java.sql.*;
10  import java.sql.Date;
11  import java.text.ParseException;
12  import java.text.SimpleDateFormat;
13  import java.util.ArrayList;
14  import java.util.Arrays;
```

```
import java.util.HashMap;
import java.util.Objects;
import javax.imageio.ImageIO;
import javax.swing.ImageIcon;
import javax.swing.JComboBox;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.table.DefaultTableModel;
import
org.jfree.data.category.DefaultCategoryDataset;
import
org.jfree.data.general.DefaultPieDataset;

public class Actor_Utils extends Utility{
    public static final int FIRST_INDEX = 0;
    public static final int INVALID_INDEX = -1;

    private static int currentIndex =
FIRST_INDEX;
    private static final String SQL_ID =
Query_Actor.get_sql_id();

    //Creates actor table
    public static void create_actor_table() {
        try (Connection conn = getConnection())
{
            Statement stmt =
conn.createStatement();

stmt.addBatch(Query_Actor.get_sql_actor());
            stmt.executeBatch();

            String message =
String.format("Successfully creates actor
table");
            JOptionPane.showMessageDialog(null,
message,

 "INFORMATION",JOptionPane.INFORMATION_MESSAGE)
;

        } catch (SQLException ex) {
            JOptionPane.showMessageDialog(null,
ex.getMessage(),

 "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    //Populates actor table with some rows of
data
    public static void populate_actor_table(){
        try(Connection conn = getConnection()){

```

```
            String sql = """
                INSERT INTO actor(actor_id,
first_name, last_name, last_update)
                VALUES(?, ?, ?, ?)""";

            //Creates a new Actor class with
default constructor
            PreparedStatement ps1 =
conn.prepareStatement(sql);
            Actor obj1 = new Actor();
            ps1.setInt(1,obj1.getActorID());

ps1.setString(2,obj1.getFirstName());

ps1.setString(3,obj1.getLastName());

ps1.setTimestamp(4,obj1.getLastUpdate());

            // Creates a new Actor class with
four-params constructor
            PreparedStatement ps2 =
conn.prepareStatement(sql);
            Actor obj2 = new Actor(2, "Vivian",
"Siahaan", new
Timestamp(System.currentTimeMillis()));
            ps2.setInt(1,obj2.getActorID());

ps2.setString(2,obj2.getFirstName());

ps2.setString(3,obj2.getLastName());

ps2.setTimestamp(4,obj2.getLastUpdate());

            ps1.executeUpdate();
            ps2.executeUpdate();

        }catch(SQLException ex){
            JOptionPane.showMessageDialog(null,
ex.getMessage(),

 "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    //Reads the content of actor table
    public static void read_actor_table(){
        try(Connection conn = getConnection()){
            Statement stmt =
conn.createStatement();
            ResultSet rs =
stmt.executeQuery("SELECT * FROM actor");

            while(rs.next()){
```

```
                int act_id =
rs.getInt("actor_id");
                String fname =
rs.getString("first_name");
                String lname =
rs.getString("last_name");
                Timestamp lu =
rs.getTimestamp("last_update");

                //Creates an Actor object using
four-params constructor
                Actor obj = new Actor(act_id,
fname, lname, lu);
                System.out.println(obj);
            }
            rs.close();
            stmt.close();

        }catch(SQLException ex){
            JOptionPane.showMessageDialog(null,
ex.getMessage(),

 "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }
}
```

Step 4

In the driver class, **Sakila.java**, invoke **testConnection()**, **create_actor_table()**, **populate_actor_table()**, and **read_actor_table()**.

```
1  package sakila;
2
3  public class Sakila {
4      public static void
5  main(String[] args) {
6          Utility.testConnection();
7
8  Actor_Utils.create_actor_table();
9
10 Actor_Utils.populate_actor_table();

   Actor_Utils.read_actor_table();
    }
}
```

Run project to see the result in console:

```
Actor ID      : 1
Full Name     : First Name Last Name
```

	Last Update : 2023-04-21 18:59:11.0 Actor ID : 2 Full Name : Vivian Siahaan Last Update : 2023-04-21 18:59:11.0

DESIGNING GUI
DESIGNING GUI

Step 1	In the project, create a new **JFrame Form** and name it as **ActorForm.java**. In the Design tab, add five **JLabels** to the form and set their corresponding **text** properties as ACTOR ID, FULL NAME, FIRST NAME, LAST NAME, and LAST UPDATE.
Step 2	Then, add four **JTextField** to the form and set their corresponding **Variable Name** as **jtfActorID**, **jtfFirstName**, **jtfLastName**, and **jtfLastUpdate**.
Step 3	Then, add seven **JButton** to the form and set their corresponding **Variable Name** as **jbFirst**, **jbPrev**, **jbNext**, **jbLast**, **jbEdit**, **jbInsert**, and **jbDelete**. Set their corresponding **text** properties as FIRST, PREV, NEXT, LAST, EDIT, INSERT, and DELETE.
Step 4	Then, add two **JComboBoxes** to the form and set their corresponding **Variable Name** as **jcbActorID** and **jcbFullName**.
Step 5	Lastly, add a new **JTable** to the form set set its **Variable Name** as **jtActor**. Then, right-click on it, then choose **Table Contents...** and set the number of columns to 3 and the number of rows to 20.
Step 6	In the driver class, **Sakila.java**, create a new object of **ActorForm** class using its

default constructor as shown in 10 – 11:

```
1  package sakila;
2
3  public class Sakila {
4      public static void
5  main(String[] args) {
6  //        Utility.testConnection();
7
8  //
9  Actor_Utils.create_actor_table();
10 //
11 Actor_Utils.populate_actor_table();
12 //
13 Actor_Utils.read_actor_table();
           ActorForm frm = new
   ActorForm();
           frm.setVisible(true);
       }
   }
```

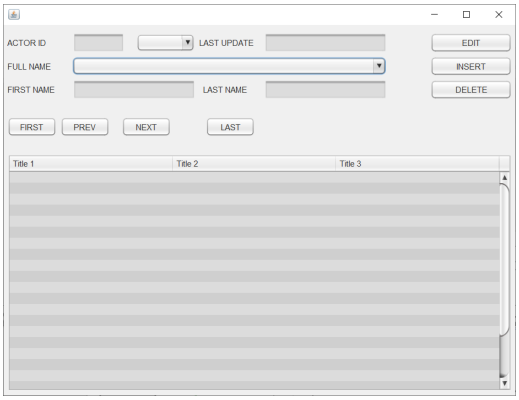


Figure 2.1 The layout of actor form

Step 8

In **RegionForm**'s constructor, invoke **setLookAndFeel()** to set the look and feel of the form as shown in line 17.

```
1  package Sakila;
2
3  import java.awt.Toolkit;
4  import java.awt.event.ActionEvent;
5  import
6  java.awt.event.ActionListener;
7  import javax.swing.JButton;
8  import javax.swing.JComboBox;
9  import javax.swing.JMenuItem;
10 import javax.swing.JOptionPane;
11 import javax.swing.JPopupMenu;
```

```
12 import javax.swing.JTable;
13 import javax.swing.JTextField;
14
15 public class ActorForm extends
16 javax.swing.JFrame {
17     public ActorForm() {
18         initComponents();
19
20 Utility.setLookAndFeel(this);
      }
      //...
}
```

Run the project to see the actor form as shown in Figure 2.1.

Step 9

In **ActorForm.java**, define **getter()** method for every object in the form and place them outside and beneath its default constructor:

```
1     //Getter method for jtActor
2     public JTable getJTActor(){
3         return this.jtActor;
4     }
5
6     //Getter method for jtfActorID
7     public JTextField
8 getJTFActorID(){
9         return this.jtfActorID;
10    }
11
12    //Getter method for
13 jtfLastUpdate
14    public JTextField
15 getJTFLastUpdate(){
16        return this.jtfLastUpdate;
17    }
18
19    //Getter method for
20 jtfFirstName
21    public JTextField
22 getJTFFirstName(){
23        return this.jtfFirstName;
24    }
25
26    //Getter method for jtfLastName
27    public JTextField
28 getJTFLastName(){
29        return this.jtfLastName;
30    }
```

```
31
32      //Getter method for jtableActor
33      public JTable getJTableActor(){
34          return this.jtActor;
35      }
36
37      //Getter method for jcbActorID
38      public JComboBox
39  getJCBActorID(){
40          return this.jcbActorID;
41      }
42
43      //Getter method for jcbFullName
44      public JComboBox
45  getJCBFullName(){
46          return this.jcbFullName;
47      }
48
49      //Getter method for jbEdit
50      public JButton getJBEdit(){
51          return this.jbEdit;
52      }
53
54      //Getter method for jbInsert
55      public JButton getJBInsert(){
56          return this.jbInsert;
57      }
58
59      //Getter method for jbDelete
60      public JButton getJBDelete(){
61          return this.jbDelete;
62      }
63
64      //Getter method for jbFirst
65      public JButton getJBFirst(){
66          return this.jbFirst;
67      }
68
69      //Getter method for jbPrev
70      public JButton getJBPrev(){
71          return this.jbPrev;
72      }
73
74      //Getter method for jbNext
        public JButton getJBNext(){
            return this.jbNext;
        }

        //Getter method for jbLast
        public JButton getJBLast(){
            return this.jbLast;
        }
```

POPULATING TABLE AND COMBOBOXES

POPULATING TABLE AND COMBOBOXES

Step 1

In **Actor_Utils.java**, add two new methods: **get_actor_list()** and **show_ta**
These methods are used to retrieve and display actors from database.

The **get_actor_list()** method takes in an **ActorForm** object, a SQL query s
an **ArrayList** of **Actor** objects. The method establishes a database con
method and prepares a statement using the SQL query string. If the item
statement sets a parameter to the item string using the **setString()** met
the resulting **ResultSet** is iterated through, creating an **Actor** object usin
adding it to the list **ArrayList**. If an SQL exception occurs, the method dis

The **show_table_actor()** method takes in an **ActorForm** object and a
displays the actors' details in **JTable**. The method creates a **DefaultTabl**
headers using the **set_column_header()** method. The method sets the t
the ActorForm object to the created model. The method then loops thro
actor's details to a new row of the table model using the **addRow()** metho

```
1     private static ArrayList<Actor> get_actor_list(ActorForm
2  frm, String sql, String item){
3          ArrayList<Actor> list = new ArrayList<>();
4          Connection conv = null;
5
6          try(Connection conn = getConnection();
7              PreparedStatement ps = conn.prepareStatement(sql))
8  {
9              if (item.equalsIgnoreCase("none")==false) {
10                  ps.setString(1,item);
11             }
12             ResultSet rs = ps.executeQuery();
13
14             Actor obj;
15             while(rs.next()){
16                 //Using five-params constructor
17                 obj = new Actor(rs.getInt("actor_id"),
18                     rs.getString("first_name"),
19                     rs.getString("last_name"),
20                     rs.getTimestamp("last_update"));
21
22                 list.add(obj);
23             }
24         }catch (SQLException ex){
25             JOptionPane.showMessageDialog(frm,
26 ex.getMessage(),
27                 "ERROR",JOptionPane.ERROR_MESSAGE);
28         }
```

```
        return list;
    }

    private static void show_table_actor(ActorForm frm,
ArrayList<Actor> list) throws SQLException{
        DefaultTableModel model = new DefaultTableModel(0,0);

        String header[] = {"Actor ID", "Full Name", "Last
Update"};


model.setColumnIdentifiers(set_column_header(frm.getJTActor(),
header));
        frm.getJTActor().setModel(model);

        Object[] row = new Object[3];

        for(int i=0; i<list.size(); i++){
            row[0] = list.get(i).getActorID();
            row[1] = list.get(i).getFirstName() + " " +
list.get(i).getLastName();
            row[2] = list.get(i).getLastUpdate();

            model.addRow(row);
        }
    }
```

Step 2

In **Actor_Utils.java**, define **refresh_controls()** method. It updates the querying data from database.

The method first sets the location and title of the frm **ActorForm** object. I the **jtActor** using the **table_renderer()** method.

The method calls the **get_actor_list()** method with a SQL query string t table and the item parameter set to **"none"**. The resulting **ArrayList** of **show_table_actor()** method, which displays the actors' details in the **jtAc**

Next, the method populates the **jcbActorID** JComboBox by executing a S from the **actor** table, ordering them by actor ID, and passing the query s object to the **populate_combobox()** method.

Finally, the method populates the **jcbFullName** JComboBox by executin concatenated first name and last name values from the **actor** table, orderi and passing the query string, **JComboBox**, and **ActorForm** object to the **p**

If a **SQLException** is caught during any of these operations, the meth **JOptionPane.showMessageDialog**.

```
    public static void
refresh_controls(ActorForm frm){

frm.setLocationRelativeTo(null);
        frm.setTitle("ACTOR FORM");

        //Shows the content of
actor table and populates combobox
        try{
            //Makes alternating
color for table rows

table_renderer(frm.getJTActor());

            //Populates table
            ArrayList<Actor> list =
get_actor_list(frm, "SELECT * FROM
actor", "none");
            show_table_actor(frm,
list);

            //Populates jcbActorID
            String sql_id = "SELECT
actor_id FROM actor ORDER BY
actor_id";

populate_combobox(sql_id,
frm.getJCBActorID(), frm);


            //Populates
getJCBFullName
            String sql_name =
 "SELECT DISTINCT
CONCAT(first_name, ' ', last_name)
FROM actor ORDER BY
CONCAT(first_name, ' ',
last_name)";


populate_combobox(sql_name,
frm.getJCBFullName(), frm);

        }catch (SQLException ex){

JOptionPane.showMessageDialog(frm,
ex.getMessage(),

 "ERROR",JOptionPane.ERROR_MESSAGE)
;
        }
    }
```

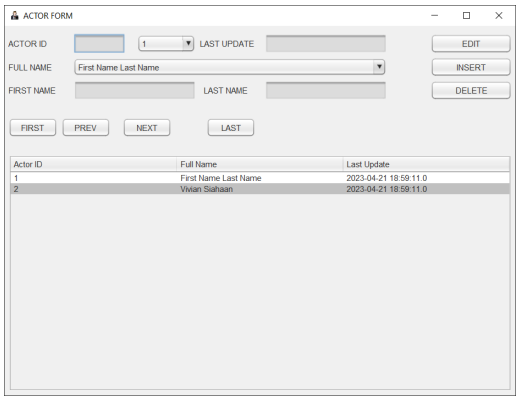


Figure 2.2 The content of **actor** table displayed

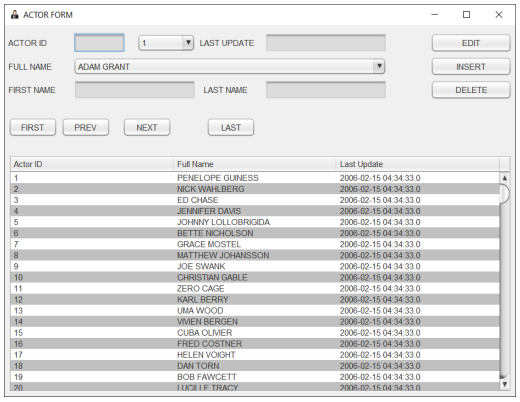


Figure 2.3 The the content of **actor** table in original **Sakila** database av

jtActor

Step 3

In **ActorForm**'s default constructor, the **Actor_Utils.refresh_controls(th** the controls in the **ActorForm** with data from a database using the **re** **Actor_Utils** class.

The **this.setIconImage()** method sets the icon of the **A** **this.setDefaultCloseOperation(this.HIDE_ON_CLOSE)** method sets th to hide the form instead of exiting the application when the close button is

```
1  package sakila;
2  import java.awt.Toolkit;
3  import java.awt.event.ActionEvent;
4  import java.awt.event.ActionListener;
5  import java.io.IOException;
6  import java.text.ParseException;
7  import java.util.logging.Level;
8  import java.util.logging.Logger;
9  import javax.swing.JButton;
10 import javax.swing.JComboBox;
11 import javax.swing.JLabel;
```

```
12 import javax.swing.JTable;
13 import javax.swing.JTextField;
14
15 public class ActorForm extends javax.swing.JFrame {
16     public ActorForm() {
17         initComponents();
18         Utility.setLookAndFeel(this);
19         Actor_Utils.refresh_controls(this);
20
21  this.setIconImage(Toolkit.getDefaultToolkit().getImage(getClass().
22 ;
23         this.setDefaultCloseOperation(this.HIDE_ON_CLOSE);
24     }
25     //...
26 }
```

Step 4	Run the project to see the content of **actor** table displayed in **jtActor** as sh If you use the data from **Sakila** MySQL database available in the internet, table displayed in **jtActor** as shown in Figure 2.3.

DISPLAYING AND NAVIGATING DATA ROW BY ROW

DISPLAYING AND NAVIGATING DATA ROW BY ROW

Step 1	In **Actor_Utils** class, define two new methods named **clear_co** **display_actor_data()**. The **clear_controls()** method clears the text fields of an **ActorForm** obje text of **jtfActorID**, **jtfFirstName**, **jtfLastName**, and **jtfLastUpdate** string. The **display_actor_data()** method is used to display the data of a selecte **ActorForm** object. It takes three parameters: **frm**, an **ActorForm** object, that represents the SQL query to execute; and item, an object that re selected item in a combo box. The method uses a **PreparedStatement** object to execute the SQL query parameter. If the query returns no rows, the **clear_controls()** method is c the form fields. If the query returns one or more rows, the method loops result set and sets the text of the text fields in frm with the data from the It also uses the **find_combo_value_selected()** method to set the selecte combo boxes **jcbActorID** and **jcbFullName** to match the data in the

Finally, the method closes the result set and the **PreparedState** **SQLException** is caught, the method displays an error message dialog.

```
1     private static void clear_controls(ActorForm frm){
2         frm.getJTFActorID().setText("");
3         frm.getJTFFirstName().setText("");
4         frm.getJTFLastName().setText("");
5         frm.getJTFLastUpdate().setText("");
6     }
7
8    //Displays actor data result row by row
9     private static <T> void display_actor_data(ActorForm frm, Strin
10 item){
11        try(Connection conn = getConnection()){
12            PreparedStatement ps = conn.prepareStatement(sql);
13            ps.setObject(1,item);
14            ResultSet rs = ps.executeQuery();
15
16            if (!rs.next()) {
17                // no row found, clear the form fields
18                clear_controls(frm);
19                return;
20            }
21
22            do{
23
24 frm.getJTFActorID().setText(String.valueOf(rs.getInt("actor_id")));
25                frm.getJTFFirstName().setText(rs.getString("first_n
26                frm.getJTFLastName().setText(rs.getString("last_nam
27
28 frm.getJTFLastUpdate().setText(String.valueOf(rs.getTimestamp("last
29
30                // Determines item selected from jcbActorID
31                find_combo_value_selected(frm.getJCBActorID(),
32 rs.getInt("actor_id"));
33
34                // Determines item selected from jcbFullName
35                String full_name = rs.getString("first_name") + " "
36 rs.getString("last_name");
37                find_combo_value_selected(frm.getJCBFullName(), ful
38            }while(rs.next());
39
40            rs.close();
41            ps.close();
42        }catch(SQLException ex){
43            JOptionPane.showMessageDialog(frm, ex.getMessage(),
44                    "ERROR",JOptionPane.ERROR_MESSAGE);
45        }
    }
```

Step 2

In the same class, define another method named **jcbActor_handler()**. It for when a user selects an item from the Actor ID or Full Name combo **ActorForm**.

The method starts by getting the selected item from the combo box and i empty string for the SQL query. It then checks which combo box trigge and sets the SQL query accordingly by calling either **Query_Actor.ge Query_Actor.get_sql_name()**.

Finally, the **display_actor_data()** method is called with the SQL query item as parameters to display the corresponding data in the form.

```
    public static void
jcbActor_handler(ActorForm frm,
JComboBox<String> jcb) {
        Object item =
jcb.getSelectedItem();
        String sql = "";
        if
(jcb.equals(frm.getJCBActorID())) {
            sql =
Query_Actor.get_sql_id();
        } else if
(jcb.equals(frm.getJCBFullName()))
{
            sql =
Query_Actor.get_sql_name();
        }

        display_actor_data(frm,
sql, item);
    }
```

Step 3

In **ActorForm**, double click on **jcbActorID** and **jcbFullName** combobox their corresponding event handler as follows:

```
    private void
jcbActorIDActionPerformed(java.awt.event.ActionEvent
evt) {
        Actor_Utils.jcbActor_handler(this,
this.getJCBActorID());
    }

    private void
jcbFullNameActionPerformed(java.awt.event.ActionEvent
evt) {
        Actor_Utils.jcbActor_handler(this,
this.getJCBFullName());
    }
```

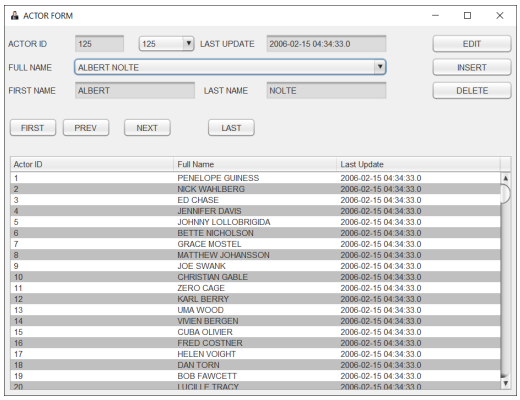


Figure 2.4 Displaying row by row the content of **actor** table

These two methods are event listeners for the **JComboBox** components and **jcbFullName** respectively. Whenever the user selects an item from the list in either of these **JComboBox** components, the corresponding event is triggered, which will call the **jcbActor_handler()** method in the **Actor** This method will then retrieve the SQL query based on the selected component and pass it along with the selected item to the **display_** method, which will display the data for the selected actor in the corres fields.

Step 4

Run the project. Choose one of items in **jcbActorID** and/or **jcbFullName** to see row by row the content of **actor** table as shown in Figure 2.4.

Step 5

Define four navigating methods in **Actor_Utils** class. These are methods t user to navigate through the data in the form, displaying the previous or n the first or last record.

The **show_first_row()** method displays the first row of data by getting th first index of the **JComboBox**, converting it to a **String**, and passing it as to the **display_actor_data()** method.

The **show_last_row()** method works similarly, but gets the item at the las **JComboBox**.

The **show_prev_row()** method decrements the **currentIndex** variable an it doesn't go below the first index, and then gets the item at the current **JComboBox** and passes it to **display_actor_data()**.

The **show_next_row()** method increments the currentIndex variable and doesn't go beyond the last index, and then gets the item at the current **JComboBox** and passes it to **display_actor_data()**.

```
1    public static void show_first_row(ActorForm frm){
```

```
        String item =
String.valueOf(frm.getJCBActorID().getItemAt(FIRST_INDEX));
        display_actor_data(frm, SQL_ID, item);
        currentIndex = FIRST_INDEX;
    }

    public static void show_last_row(ActorForm frm){
        int endIndex = frm.getJCBActorID().getItemCount() -
1;
        String item =
String.valueOf(frm.getJCBActorID().getItemAt(endIndex));
        display_actor_data(frm, SQL_ID, item);
        currentIndex = endIndex;
    }

    public static void show_prev_row(ActorForm frm){
        currentIndex--;
        if(currentIndex < FIRST_INDEX){
            currentIndex = FIRST_INDEX;
            return;
        }
        String item =
String.valueOf(frm.getJCBActorID().getItemAt(currentIndex));
        display_actor_data(frm, SQL_ID, item);
    }

    public static void show_next_row(ActorForm frm){
        int endIndex = frm.getJCBActorID().getItemCount() -
1;
        currentIndex++;
        if(currentIndex > endIndex){
            currentIndex = endIndex;
            return;
        }
        String item =
String.valueOf(frm.getJCBActorID().getItemAt(currentIndex));
        display_actor_data(frm, SQL_ID, item);
    }
```

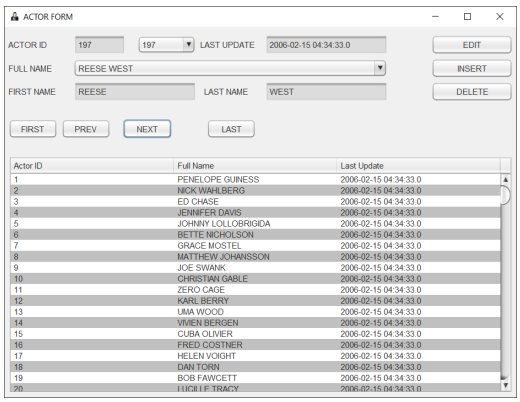


Figure 2.5 User clicks on one or more navigation buttons on actor

Step 6	Then in **ActorForm**, double click on each navigation buttons to corresponding event handler:
	
	`private void jbNextActionPerformed(java.awt.event.ActionEvent evt) { Actor_Utils.show_next_row(this); }`
	
	`private void jbFirstActionPerformed(java.awt.event.ActionEvent evt) { Actor_Utils.show_first_row(this); }`
	
	`private void jbPrevActionPerformed(java.awt.event.ActionEvent evt) { Actor_Utils.show_prev_row(this); }`
	
	`private void jbLastActionPerformed(java.awt.event.ActionEvent evt) { Actor_Utils.show_last_row(this); }`
	
	These are the action listener methods for the "Next", "First", "Prev", and " respectively. They call the corresponding methods in the **Actor_Utils** clas next/first/previous/last row in the **actor** table.

Step 7	Run the project. Click on one or more navigation buttons to see the resul Figure 2.5.
Step 8	Define **mouse_pressed_handler()** method in **Actor_Utils** class. It is us the mouse press event in the **JTable** component on the ActorForm. It selected row from the table and then displays the data associated with t other form fields using the **display_actor_data()** method. If no row i displays an information message to the user. Here is a summary of what the method does: 1. Check if a row is selected in the **JTable** component. 2. If no row is selected, display an information message to the user. 3. Retrieve the connection to the database. 4. Get the ID of the selected row from the first column of the selected

5. Call the **display_actor_data()** method to display the data associ selected row in the other form fields.
6. If there is a SQLException, log the error and display an error m user.

```
    public static void mouse_pressed_handler(ActorForm frm) {
        Objects.requireNonNull(frm, "frm must not be null");

        int selectedIndex = frm.getJTActor().getSelectedRow();
        if (selectedIndex == -1) {
            JOptionPane.showMessageDialog(frm, "Please select a row to view its
data.",
                "No row selected", JOptionPane.INFORMATION_MESSAGE)
            return;
        }

        try (Connection conn = getConnection()) {
            String id =
String.valueOf(frm.getJTActor().getModel().getValueAt(selectedIndex
0));

            // Displays actor data
            display_actor_data(frm, Query_Actor.get_sql_id(), id);

        } catch (SQLException ex) {

Logger.getLogger(ActorForm.class.getName()).log(Level.SEVERE, "Erro
displaying actor data", ex);
            String message = "Error displaying actor data: " +
ex.getMessage();
            String stackTrace = Arrays.toString(ex.getStackTrace())
            JOptionPane.showMessageDialog(frm, message + "\n\n" +
stackTrace, "ERROR", JOptionPane.ERROR_MESSAGE);
        }
    }
```

Step 9

Right click on **jtActor**. Then, choose **Events > Mouse > mousePresse** event handler:

```
    private void
jtActorMousePressed(java.awt.event.MouseEvent
evt) {

 Actor_Utils.mouse_pressed_handler(this);
    }
```

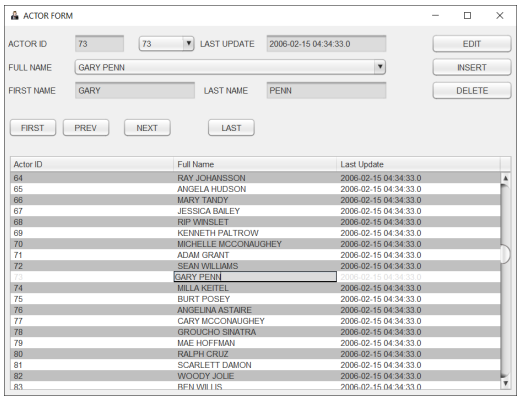


Figure 2.6 User double-clicks on any row in **jtActor**

Step 10	Run the project. Double click on any row in **jtActor** table. You corresponding row in **actor** table displayed in textfields and comboboxes Figure 2.6.

UPDATING RECORD
UPDATING RECORD

Step 1	In **Actor_Utils** class, define a new method named **update_row_by_actor_id()**. This method updates a row of data in the actor table by actor ID. It takes an **ActorForm** object, an integer for the actor ID, and two strings for the first and last name of the actor to be updated. First, a database connection is established using the **getConnection()** method. Then, a query is executed to check if the provided actor ID exists in the database. If the actor ID is not found, an error message is displayed, and the method returns. If the actor ID is found, a new **Actor** object is created using the provided ID, first name, last name, and the current timestamp. The data is then updated in the database using an SQL update query with the **PreparedStatement** object **updatePS**. Finally, all database resources and connections are closed. If any SQL or input errors occur, an error message is logged and displayed to the user using **JOptionPane**.

```
1       //Updates row of data in actor tabel by actor_id
2       public static void update_row_by_actor_id(ActorForm frm,
3   int act_id, String fname, String lname) throws SQLException{
```

```
        Connection conn = getConnection();
        ResultSet rs = null;
        String query_id = "SELECT actor_id FROM actor WHERE
actor_id = ?";
        String update_query = """
            UPDATE actor SET first_name = ?, last_name = ?
WHERE actor_id = ?""";
        try(PreparedStatement idPs =
conn.prepareStatement(query_id,

ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_READ_ONLY);
        PreparedStatement updatePS =
conn.prepareStatement(update_query,
                ResultSet.TYPE_SCROLL_SENSITIVE,
ResultSet.CONCUR_UPDATABLE))
        {
            idPs.setInt(1,act_id);
            if(!idPs.execute()){
                String message = "Can't find actor_id " +
act_id;

                JOptionPane.showMessageDialog(frm, message,
                    "ERROR",JOptionPane.ERROR_MESSAGE);
            } else{
                rs = idPs.getResultSet();
                rs.next();

                //Creates a Actor object using four-params
constructor
                Actor obj = new Actor(act_id, fname, lname,
new Timestamp(System.currentTimeMillis()));
                updatePS.setString(1, obj.getFirstName());
                updatePS.setString(2, obj.getLastName());
                updatePS.setInt(3, obj.getActorID());

                updatePS.executeUpdate();
                rs.close();
                updatePS.close();
                idPs.close();
                conn.close();
            }
        }catch(SQLException ex){

Logger.getLogger(ActorForm.class.getName()).log(Level.SEVERE,
"Error updating actor data", ex);
            String message = "Error updating actor data: " +
ex.getMessage();
            String stackTrace =
Arrays.toString(ex.getStackTrace());
            JOptionPane.showMessageDialog(null, message +
"\n\n" + stackTrace, "ERROR", JOptionPane.ERROR_MESSAGE);
        }catch(java.lang.NumberFormatException ex){

Logger.getLogger(ActorForm.class.getName()).log(Level.SEVERE,
```

```
"Invalid Input", ex);
            String message = "Invalid Input: " +
ex.getMessage();
            String stackTrace =
Arrays.toString(ex.getStackTrace());
            JOptionPane.showMessageDialog(null, message +
"\n\n" + stackTrace, "ERROR", JOptionPane.ERROR_MESSAGE);
        }
    }
```

Step 2

Then in the same class, define a new method **read_inputs()**. The method is responsible for reading and validating the user inputs for the actor data fields (actor ID, first name, and last name) on the **ActorForm**.

It takes an **ActorForm** object as its argument and returns a **HashMap** of the input data with keys "act_id", "fname", and "lname".

The method first initializes an empty **HashMap** called **input_data**, then reads the values of the actor ID, first name, and last name fields from the **ActorForm** object.

It then performs some validation checks on the input data. For example, it checks that the actor ID is a positive integer, and that the first name and last name fields are not empty. If any of these checks fail, the method displays an error message and throws an exception.

Finally, if all the input data is valid, the method adds the values to the **input_data** HashMap and returns it.

```
1   private static HashMap<String, String>
2   read_inputs(ActorForm frm) {
3           HashMap<String, String> input_data = new
4   HashMap<>();
5           String act_id =
6   String.valueOf(frm.getJCBActorID().getSelectedItem());
7           String fname =
8   frm.getJTFFirstName().getText();
9           String lname = frm.getJTFLastName().getText();
10
11          // Validate user input
12          int act_id_int = 0;
13          try {
14              act_id_int = Integer.parseInt(act_id);
15              if (act_id_int <= 0) {
```

```
                throw new
IllegalArgumentException("Actor ID cannot be negative
or zero");
            }
        } catch (NumberFormatException ex) {
            JOptionPane.showMessageDialog(frm,
"Invalid Actor ID: " + act_id,
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        } catch (IllegalArgumentException ex) {
            JOptionPane.showMessageDialog(frm,
ex.getMessage(),
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        }

        if (fname == null || fname.isEmpty()) {
            JOptionPane.showMessageDialog(frm, "First
name cannot be empty",
            "Error", JOptionPane.ERROR_MESSAGE);
            throw new IllegalArgumentException("First
name cannot be empty");
        }
```

```
        if (lname == null ||
lname.isEmpty()) {

JOptionPane.showMessageDialog(frm,
"Last name cannot be empty",
            "Error",
JOptionPane.ERROR_MESSAGE);
            throw new
IllegalArgumentException("Last name
cannot be empty");
        }

        input_data.put("act_id",
act_id);
        input_data.put("fname", fname);
        input_data.put("lname", lname);

        return input_data;
    }
```

Step 3

Still in the same class, define an named **edit_actual()**. It is called clicks the "Edit" button on the reads the inputs from the for **read_inputs()** method and corresponding row in the databa **update_row_by_actor_id()** metho refreshes all the objects on the f **refresh_controls()** method. If an is thrown during the update, it s message to the user using a **JOptio**

```
1   private static void
2  edit_actual(ActorForm frm){
3          try{
4              HashMap<String, Str
5  input_data = read_inputs(frm);
6              int act_id =
7  Integer.parseInt(input_data.get
8              String fname =
9  input_data.get("fname");
10             String lname =
11 input_data.get("lname");
12
13             update_row_by_actor
14 act_id, fname, lname);
15
16             //Refreshes all obj
```

```
17              refresh_controls(fr

            }catch(SQLException ex)

JOptionPane.showMessageDialog(f
ex.getMessage(),

 "ERROR",JOptionPane.ERROR_MESS
            }
        }
```

Step 4

Lastly, define two new met **enable_controls()** and **edit_ha edit_handler()** method is responsib the Edit button's click event in th When the Edit button is clicked checks whether the text on the butt it is, the button text is changed to "C controls on the form are disabled, ar edit the actor's data. If the b "CONFIRM", the method change "EDIT", calls the **edit_actual()** met the actor's data, refreshes the co form, and enables the controls.

Overall, the **edit_handler()** method for managing the edit functio **ActorForm** and is called when the **Edit** button.

```
1     private static void
2  enable_controls(boolean state,
3  frm){
4           frm.getJBFirst().setEna
5           frm.getJBPrev().setEnab
6           frm.getJBNext().setEnab
7           frm.getJBLast().setEnab
8           frm.getJBInsert().setEn
9           frm.getJBDelete().setEn
10
11 frm.getJTFActorID().setEnabled(
12     }
13
14     public static void
15 edit_handler(ActorForm frm){
16
17  if(frm.getJBEdit().getText().e
18 ){
19
```

```
20
21  frm.getJBEdit().setText("CONFIF
22
23              // Disables control
24              enable_controls(fal
25          }
26
27          else {
28              frm.getJBEdit().set

                // Actual editing
                edit_actual(frm);

                //Enables controls
                enable_controls(tru
            }
        }
```

Step 5

Run the project. Choose **ac** **jcbActorID** or **jcbFullName** comb can choose one of rows in **jtActor** **actor_id = 9**). Then, click on E shown in Figure 2.7.

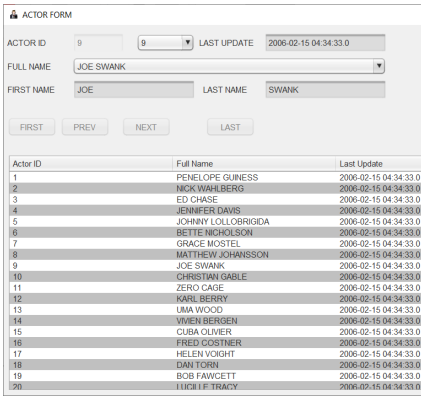


Figure 2.7 The actor form is in e

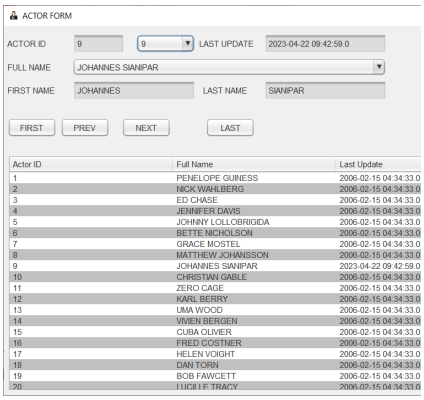


Figure 2.8 The edited row had be
database

Edit first name and/or last name.
CONFIRM button. The edited row h
into **actor** table as shown in Figure

UPDATING RECORD DIRECTLY ON JTABLE
UPDATING RECORD DIRECTLY ON JTABLE

Step 1

In **Actor_Utils** class, define a new method na
edit_database_from_jtable(). This is a method that updates a row in **a**
table based on changes made to **jtActor** in **ActorForm**.

Here is how the method works:

1. It checks if the **TableModelEvent** passed to the method is of
"UPDATE".
2. If it is, it retrieves the row number from the event using
getFirstRow() method.
3. It then gets the **TableModel** object from the event using
getSource() method.
4. It retrieves the actor ID and full name from the model using
getValueAt() method.
5. It then splits the full name string into an array of two strings using
split() method with a space as the delimiter.
6. It tries to update the row in the database using
update_row_by_actor_id() method with the actor ID and the first
last names as arguments.
7. If an exception occurs, it catches the exception and shows an e
message in a dialog box with the specific error message obtained f
the exception.

The purpose of this method is to update the **actor** table with the char made to the **jtActor** in the GUI. It takes the actor ID and full name from table and splits the full name into separate first and last name strings. The uses the **update_row_by_actor_id()** method to update the corresponding in the database with the new first and last names.

```
    public static void
edit_database_from_jtable(TableModelEvent e, ActorForm frm){
        if (e.getType() == TableModelEvent.UPDATE) {
            int row = e.getFirstRow();
            TableModel model = (TableModel)e.getSource();
            int act_id =
Integer.parseInt(String.valueOf(model.getValueAt(row, 0)));
            String fullname = (String) model.getValueAt(row,
1);
            String[] names = fullname.split(" ");

            try{
                update_row_by_actor_id(frm, act_id, names[0],
names[1]);

                //Refreshes all objects on form
                refresh_controls(frm);

            } catch (SQLIntegrityConstraintViolationException
ex) {

Logger.getLogger(ActorForm.class.getName()).log(Level.SEVERE,
"Duplicate entry", ex);
                JOptionPane.showMessageDialog(frm, "Error:
Duplicate entry\n" + ex.getMessage());
            } catch (SQLSyntaxErrorException ex) {

Logger.getLogger(ActorForm.class.getName()).log(Level.SEVERE,
"Invalid SQL syntax", ex);
                JOptionPane.showMessageDialog(frm, "Error:
Invalid SQL syntax\n" + ex.getMessage());
            } catch (SQLException ex) {

Logger.getLogger(ActorForm.class.getName()).log(Level.SEVERE,
"Database error", ex);
                JOptionPane.showMessageDialog(frm, "Error:
Database error\n" + ex.getMessage());
            }
        }
    }
```

Step 2

Create a new public class named **ActorTableModelListener**. It implem the **TableModelListener** interface. This class is intended to listen to char

in a **JTable** object and update the database accordingly through **edit_database_from_jtable()** method from the **Actor_Utils** class.

The **ActorTableModelListener** constructor takes two arguments: a **JTa** object and an **ActorForm** object. These objects are used to keep track of changes made in the **JTable** and update the database accordingly.

The **tableChanged()** method is an implementation of **TableModelListener** interface. This method is called whenever a chang made to the table model. In this method, the **edit_database_from_jtab** method is called to update the database based on the changes made to **JTable**. If the cell editor is not null, it is stopped to ensure that any char made to the cell are saved before moving on to the next cell.

```
1  package sakila;
2  import
3  javax.swing.event.TableModelEvent;
4  import
5  javax.swing.event.TableModelListener;
6  import javax.swing.JTable;
7
8  public class ActorTableModelListener
9  implements TableModelListener {
10     private final JTable jt;
11     private final ActorForm frm;
12
13     public
14 ActorTableModelListener(JTable jt,
15 ActorForm frm) {
16         this.jt = jt;
17         this.frm = frm;
18     }
19
20     @Override
21     public void
22 tableChanged(TableModelEvent e) {
23
   Actor_Utils.edit_database_from_jtable(e,
   frm);

         if (jt.getCellEditor() != null)
   {

   jt.getCellEditor().stopCellEditing();
         }
     }
   }
```

Step	Right click on **jtActor**. Then, choose **Events > Mouse > mouseClicl**

3	Define its event handler:
	
	`private void jtActorMouseClicked(java.awt.event.MouseEvent evt)`
	`// instantiate ActorTableModelListener and add it as a liste`
	`to the table model`
	`ActorTableModelListener tableModelListener = new`
	`ActorTableModelListener(this.getJTActor(), this);`
	
	`this.getJTActor().getModel().addTableModelListener(tableModelListen`
	`;`
	`}`
	
	The **jtActorMouseClicked()** method is an event handler that is execu when the mouse is clicked on the **jtActor** table. The method instantiate **ActorTableModelListener** and adds it as a listener to the table mode **jtActor**.
	
	This means that when the table is modified (e.g. a cell is edited), **tableChanged()** method in the **ActorTableModelListener** will be execu which in turn calls the **edit_database_from_jtable()** method in **Actor_Utils** class to update the database with the new values.
	
	Additionally, the method checks if there is a cell editor currently editing a and stops the editing to ensure that the updated value is saved before other action is taken.

Step 4	Run the project. Click on any cell in second column in any row in **jtActor** you want to edit. Then, change it. Then, click any where outside corresponding cell. The edited data had been saved into database.

INSERTING NEW RECORD
INSERTING NEW RECORD

Step 1	In **Actor_Utils** class, define a method named **insert_row()**. It inserts a new row into an actor table in database. The method takes an **ActorForm** object as input, which contains the user inputs for the new actor. The inputs are read and stored in a **HashMap** named **input_data**. The actor's first name, last name, and current time are extracted from the **input_data HashMap** and stored in separate variables.

Then, an SQL insert statement is created as a string, using the SQL syntax to insert a new row into the **actor** table with two values, **first_name** and **last_name**. The values are represented by question marks, which are placeholders that will be replaced with actual values later.

Next, a connection to the database is established using the **getConnection()** method. Then, a **PreparedStatement** object is created using the SQL insert statement. The **PreparedStatement** object is used to set the actual values for the question marks in the SQL statement using the **setString()** method.

After the values are set, the **executeUpdate()** method is called on the **PreparedStatement** object to execute the SQL insert statement and insert the new row into the **actor** table.

If any **SQLException** occurs during this process, the exception is caught and logged using a logger and a **JOptionPane** is displayed with the error message.

```
1     //Inserts new row into actor table
2     private static void insert_row(ActorForm frm) throws
3  SQLException{
4         HashMap<String, String> input_data =
5  read_inputs(frm);
6         int act_id =
7  Integer.parseInt(input_data.get("act_id"));
8         String fname = input_data.get("fname");
9         String lname = input_data.get("lname");
10
11        // SQL insert statement
12        String sql = """
13            INSERT INTO actor(first_name, last_name)
14 VALUES(?, ?)""";
15
16        try(Connection conn = getConnection();
17         PreparedStatement pstmt =
18 conn.prepareStatement(sql)){
19
20            //Creates a Actor object three-params constructor
21            Actor obj = new Actor(fname, lname, new
22 Timestamp(System.currentTimeMillis()));
23            pstmt.setString(1,obj.getFirstName());
24            pstmt.setString(2,obj.getLastName());
25
26            //Executes the sql insert statement
27            pstmt.executeUpdate();
28        } catch (SQLException ex) {
29
   Logger.getLogger(ActorForm.class.getName()).log(Level.SEVERE,
```

```
"Database error", ex);
            JOptionPane.showMessageDialog(frm, "Error:
Database error\n" + ex.getMessage());
        }
    }
```

Step 2

Still in **Actor_Utils.java**, define **insert_actual()** and **insert_handler()** methods.

1. **insert_actual(ActorForm frm)**: This method is responsible for actually inserting the new row into the **actor** table in the database. It takes an **ActorForm** object as input, which represents the form with the user inputs for the new actor. The method calls the **insert_row()** method, which creates an SQL insert statement and executes it to insert the new row into the database. If an **SQLException** occurs during the insertion process, the method catches the exception and displays a message dialog box with the error message. Finally, the method calls the **refresh_controls()** method to update the table and comboboxes with the new data.

2. **insert_handler(ActorForm frm)**: This method is called when the "insert" button is clicked on the **ActorForm** object. It takes an **ActorForm** object as input. The method first checks the text of the "insert" button. If it says "INSERT", it changes the text to "CONFIRM" and disables the "edit" button and form controls. It also clears the form controls and enables the "insert" button. This allows the user to enter new data into the form. If the text of the "insert" button is "CONFIRM", it changes the text back to "INSERT" and calls the **insert_actual()** method, which executes the actual insertion of the new row into the database. If an **SQLException** occurs during the insertion process, a message dialog box is displayed with the error message. After the insertion is complete, the "edit" button is re-enabled and the form controls are re-enabled so that the user can enter new data. Finally, the **refresh_controls()** method is called to update the table and comboboxes with the new data.

```
1  private static void insert_actual(ActorForm
2  frm){
3          try{
4              insert_row(frm);
5
6              //Refreshes table and comboboxes
7              refresh_controls(frm);
```

```
8
9           }catch(SQLException ex){
10              JOptionPane.showMessageDialog(frm,
11  ex.getMessage(),
12
13   "ERROR",JOptionPane.ERROR_MESSAGE);
14          }
15      }
16
17      public static void insert_handler(ActorForm
18  frm){
19
20   if(frm.getJBInsert().getText().equals("INSERT")
21  ){
22
23  frm.getJBInsert().setText("CONFIRM");
24
25              //Disables jbEdit
26              frm.getJBEdit().setEnabled(false);
27
28              // Disables controls
29              enable_controls(false, frm);
30
31              // Clears controls
32              clear_controls(frm);
33
34              // Enables
35              frm.getJBInsert().setEnabled(true);
36          }
37
38          else {
39              frm.getJBInsert().setText("INSERT");
40
41              // Actual insertion
42              insert_actual(frm);
43
                //Enables jbEdit
                frm.getJBEdit().setEnabled(true);

                //Enables controls
                enable_controls(true, frm);
            }
        }
```

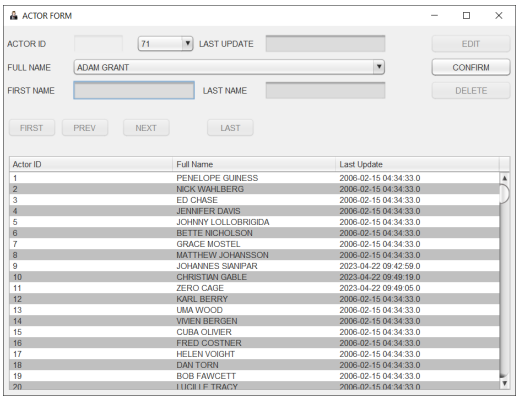


Figure 2.9 When user clicks on INSERT button, the actor form will be in state of insertion

Step 3

In **ActorForm.java**, double click on INSERT button to create its event listener:

```
    private void jbInsertActionPerformed(java.awt.event.ActionEvent evt) {
        Actor_Utils.insert_handler(this);
    }
```

Step 4

Run the project. Click on INSERT button. You will see the state of actor form when insertion is in progress as shown in Figure 2.9.

Then, type a new first name and last name. Then, click CONFIRM button to save the new record into **actor** table as shown in Figure 2.10.

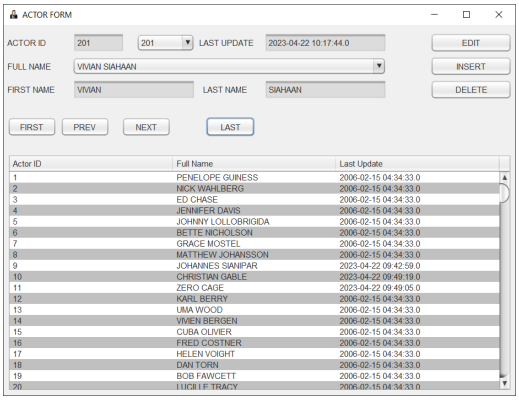


Figure 2.10 The new data had been saved into **actor** table

DELETING RECORD
DELETING RECORD

Step 1

Then in **Actor_Utils** class, define **delete_handler()** method. This metho handler for deleting a row of data from the **actor** table. It takes an **Actor** object as its parameter.

The method first prompts the user to confirm the deletion with a messag that displays the actor ID of the row to be deleted. If the user clicks the button, then the method proceeds to delete the row.

The SQL query used to delete the row is defined as a string variable r query. The actor ID of the row to be deleted is passed as a parameter query using a **PreparedStatement** object to avoid SQL injection attacks.

If the row is successfully deleted, then the method calls the **refresh_cont** method to update the table and comboboxes in the GUI. If an SQL exc occurs during the deletion, then the method displays an error messag message box.

```
1     public static void delete_handler(ActorForm frm){
2         int dialogButton = JOptionPane.YES_NO_OPTION;
3         int act_id =
4 Integer.parseInt(String.valueOf(frm.getJCBActorID().getSelectedItem
5
6         String message = String.format("Are you sure you want to de
7 the row Actor ID: %d)", act_id);
8         int answer = JOptionPane.showConfirmDialog(frm, message,
9 "DELETING ROW OF DATA", dialogButton);
10
11        if(answer == JOptionPane.YES_OPTION){
12            String query = "DELETE FROM actor WHERE actor_id = ?";
13            try(Connection conn = getConnection();
14                PreparedStatement ps = conn.prepareStatement(query)
15                // Use PreparedStatement to avoid SQL injection att
16                ps.setInt(1, act_id);
17                ps.executeUpdate();
18
19                // Refresh table and comboboxes
20                refresh_controls(frm);
21
22            } catch (SQLException ex){
23                JOptionPane.showMessageDialog(frm, ex.getMessage(),
24                    "ERROR",JOptionPane.ERROR_MESSAGE);
25            }
26        }
27    }
```

Step 2	In **ActorForm.java**, double click on DELETE button to generate its listener: `private void jbDeleteActionPerformed(java.awt.event.ActionEvent evt) { Actor_Utils.delete_handler(this); }`
Step 3	Run the project. Choose **actor_id** using jcb**ActorID** or **jcbFullName** coml Then, Click on DELETE button. The corresponding row of data had been d from database.

This is the full version of **Actor_Utils.java**:

```
package sakila;
import java.awt.Dimension;
import java.awt.image.BufferedImage;
import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.net.URL;
import java.util.logging.Level;
import java.util.logging.Logger;
import java.sql.*;
import java.sql.Date;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashMap;
import java.util.Objects;
import javax.imageio.ImageIO;
import javax.swing.ImageIcon;
import javax.swing.JComboBox;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.event.TableModelEvent;
import javax.swing.table.DefaultTableModel;
import javax.swing.table.TableModel;
import org.jfree.data.category.DefaultCategoryDataset;
import org.jfree.data.general.DefaultPieDataset;

public class Actor_Utils extends Utility{
    public static final int FIRST_INDEX = 0;
    public static final int INVALID_INDEX = -1;
```

```
    private static int currentIndex = FIRST_INDEX;
    private static final String SQL_ID = Query_Actor.get_sql_id();

    //Creates actor table
    public static void create_actor_table() {
        try (Connection conn = getConnection()) {
            Statement stmt = conn.createStatement();
            stmt.addBatch(Query_Actor.get_sql_actor());
            stmt.executeBatch();

            String message = String.format("Successfully creates actor table'
            JOptionPane.showMessageDialog(null, message,
            "INFORMATION",JOptionPane.INFORMATION_MESSAGE);

        } catch (SQLException ex) {
            JOptionPane.showMessageDialog(null, ex.getMessage(),
                "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    //Populates actor table with some rows of data
    public static void populate_actor_table(){
        try(Connection conn = getConnection()){
            String sql = """
                INSERT INTO actor(actor_id, first_name, last_name, last_updat
                VALUES(?, ?, ?, ?)""";

            //Creates a new Actor class with default constructor
            PreparedStatement ps1 = conn.prepareStatement(sql);
            Actor obj1 = new Actor();
            ps1.setInt(1,obj1.getActorID());
            ps1.setString(2,obj1.getFirstName());
            ps1.setString(3,obj1.getLastName());
            ps1.setTimestamp(4,obj1.getLastUpdate());

            // Creates a new Actor class with four-params constructor
            PreparedStatement ps2 = conn.prepareStatement(sql);
            Actor obj2 = new Actor(2, "Vivian", "Siahaan", new
Timestamp(System.currentTimeMillis()));
            ps2.setInt(1,obj2.getActorID());
            ps2.setString(2,obj2.getFirstName());
            ps2.setString(3,obj2.getLastName());
            ps2.setTimestamp(4,obj2.getLastUpdate());

            ps1.executeUpdate();
            ps2.executeUpdate();

        }catch(SQLException ex){
            JOptionPane.showMessageDialog(null, ex.getMessage(),
                "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }
```

```
    //Reads the content of actor table
    public static void read_actor_table(){
        try(Connection conn = getConnection()){
            Statement stmt = conn.createStatement();
            ResultSet rs = stmt.executeQuery("SELECT * FROM actor");

            while(rs.next()){
                int act_id = rs.getInt("actor_id");
                String fname = rs.getString("first_name");
                String lname = rs.getString("last_name");
                Timestamp lu = rs.getTimestamp("last_update");

                //Creates an Actor object using four-params constructor
                Actor obj = new Actor(act_id, fname, lname, lu);
                System.out.println(obj);
            }
            rs.close();
            stmt.close();

        }catch(SQLException ex){
            JOptionPane.showMessageDialog(null, ex.getMessage(),
                "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    private static ArrayList<Actor> get_actor_list(ActorForm frm, String sql,
String item){
        ArrayList<Actor> list = new ArrayList<>();
        Connection conv = null;

        try(Connection conn = getConnection();
            PreparedStatement ps = conn.prepareStatement(sql)){
            if (item.equalsIgnoreCase("none")==false) {
                 ps.setString(1,item);
            }
            ResultSet rs = ps.executeQuery();

            Actor obj;
            while(rs.next()){
                //Using five-params constructor
                obj = new Actor(rs.getInt("actor_id"),
                    rs.getString("first_name"),
                    rs.getString("last_name"),
                    rs.getTimestamp("last_update"));

                list.add(obj);
            }
        }catch (SQLException ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
                "ERROR",JOptionPane.ERROR_MESSAGE);
        }
        return list;
```

```
    }

    private static void show_table_actor(ActorForm frm, ArrayList<Actor> list
throws SQLException{
        DefaultTableModel model = new DefaultTableModel(0,0);

        String header[] = {"Actor ID", "Full Name", "Last Update"};

        model.setColumnIdentifiers(set_column_header(frm.getJTActor(),
header));
        frm.getJTActor().setModel(model);

        Object[] row = new Object[3];

        for(int i=0; i<list.size(); i++){
            row[0] = list.get(i).getActorID();
            row[1] = list.get(i).getFirstName() + " " +
list.get(i).getLastName();
            row[2] = list.get(i).getLastUpdate();

            model.addRow(row);
        }
    }

    public static void refresh_controls(ActorForm frm){
        frm.setLocationRelativeTo(null);
        frm.setTitle("ACTOR FORM");

        //Shows the content of actor table and populates combobox
        try{
            //Makes alternating color for table rows
            table_renderer(frm.getJTActor());

            //Populates table
            ArrayList<Actor> list = get_actor_list(frm, "SELECT * FROM actor'
"none");
            show_table_actor(frm, list);

            //Populates jcbActorID
            String sql_id = "SELECT actor_id FROM actor ORDER BY actor_id";
            populate_combobox(sql_id, frm.getJCBActorID(), frm);


            //Populates getJCBFullName
            String sql_name = "SELECT DISTINCT CONCAT(first_name, ' ',
last_name) FROM actor ORDER BY CONCAT(first_name, ' ', last_name)";

            populate_combobox(sql_name, frm.getJCBFullName(), frm);

        }catch (SQLException ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
            "ERROR",JOptionPane.ERROR_MESSAGE);
        }
```

```
    }

    private static void clear_controls(ActorForm frm){
        frm.getJTFActorID().setText("");
        frm.getJTFFirstName().setText("");
        frm.getJTFLastName().setText("");
        frm.getJTFLastUpdate().setText("");
    }

   //Displays actor data result row by row
    private static <T> void display_actor_data(ActorForm frm, String sql, T
item){
        try(Connection conn = getConnection()){
            PreparedStatement ps = conn.prepareStatement(sql);
            ps.setObject(1,item);
            ResultSet rs = ps.executeQuery();

            if (!rs.next()) {
                // no row found, clear the form fields
                clear_controls(frm);
                return;
            }

            do{

frm.getJTFActorID().setText(String.valueOf(rs.getInt("actor_id")));
                frm.getJTFFirstName().setText(rs.getString("first_name"));
                frm.getJTFLastName().setText(rs.getString("last_name"));

frm.getJTFLastUpdate().setText(String.valueOf(rs.getTimestamp("last_update"))

                // Determines item selected from jcbActorID
                find_combo_value_selected(frm.getJCBActorID(),
rs.getInt("actor_id"));

                // Determines item selected from jcbFullName
                String full_name = rs.getString("first_name") + " " +
rs.getString("last_name");
                find_combo_value_selected(frm.getJCBFullName(), full_name);
            }while(rs.next());

            rs.close();
            ps.close();
        }catch(SQLException ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    public static void jcbActor_handler(ActorForm frm, JComboBox<String> jcb)
        Object item = jcb.getSelectedItem();
        String sql = "";
        if (jcb.equals(frm.getJCBActorID())) {
```

```
            sql = Query_Actor.get_sql_id();
        } else if (jcb.equals(frm.getJCBFullName())) {
            sql = Query_Actor.get_sql_name();
        }

        display_actor_data(frm, sql, item);
    }

    public static void show_first_row(ActorForm frm){
        String item =
String.valueOf(frm.getJCBActorID().getItemAt(FIRST_INDEX));
        display_actor_data(frm, SQL_ID, item);
        currentIndex = FIRST_INDEX;
    }

    public static void show_last_row(ActorForm frm){
        int endIndex = frm.getJCBActorID().getItemCount() - 1;
        String item = String.valueOf(frm.getJCBActorID().getItemAt(endIndex));
        display_actor_data(frm, SQL_ID, item);
        currentIndex = endIndex;
    }

    public static void show_prev_row(ActorForm frm){
        currentIndex--;
        if(currentIndex < FIRST_INDEX){
            currentIndex = FIRST_INDEX;
            return;
        }
        String item =
String.valueOf(frm.getJCBActorID().getItemAt(currentIndex));
        display_actor_data(frm, SQL_ID, item);
    }

    public static void show_next_row(ActorForm frm){
        int endIndex = frm.getJCBActorID().getItemCount() - 1;
        currentIndex++;
        if(currentIndex > endIndex){
            currentIndex = endIndex;
            return;
        }
        String item =
String.valueOf(frm.getJCBActorID().getItemAt(currentIndex));
        display_actor_data(frm, SQL_ID, item);
    }

    public static void mouse_pressed_handler(ActorForm frm) {
        Objects.requireNonNull(frm, "frm must not be null");

        int selectedIndex = frm.getJTActor().getSelectedRow();
        if (selectedIndex == -1) {
            JOptionPane.showMessageDialog(frm, "Please select a row to view i
data.",
                "No row selected", JOptionPane.INFORMATION_MESSAGE);
```

```
            return;
        }

        try (Connection conn = getConnection()) {
            String id =
String.valueOf(frm.getJTActor().getModel().getValueAt(selectedIndex, 0));

            // Displays actor data
            display_actor_data(frm, Query_Actor.get_sql_id(), id);

        } catch (SQLException ex) {
            Logger.getLogger(ActorForm.class.getName()).log(Level.SEVERE,
"Error displaying actor data", ex);
            String message = "Error displaying actor data: " + ex.getMessage(
            String stackTrace = Arrays.toString(ex.getStackTrace());
            JOptionPane.showMessageDialog(frm, message + "\n\n" + stackTrace,
"ERROR", JOptionPane.ERROR_MESSAGE);
        }

    }

    //Updates row of data in actor tabel by actor_id
    public static void update_row_by_actor_id(ActorForm frm, int act_id, Str
fname, String lname) throws SQLException{
        Connection conn = getConnection();
        ResultSet rs = null;
        String query_id = "SELECT actor_id FROM actor WHERE actor_id = ?";
        String update_query = """
            UPDATE actor SET first_name = ?, last_name = ? WHERE actor_id =
?""";
        try(PreparedStatement idPs = conn.prepareStatement(query_id,

ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_READ_ONLY);
        PreparedStatement updatePS = conn.prepareStatement(update_query,
                ResultSet.TYPE_SCROLL_SENSITIVE, ResultSet.CONCUR_UPDATABLE)
        {
            idPs.setInt(1,act_id);
            if(!idPs.execute()){
                String message = "Can't find actor_id " + act_id;

                JOptionPane.showMessageDialog(frm, message,
                    "ERROR",JOptionPane.ERROR_MESSAGE);
            } else{
                rs = idPs.getResultSet();
                rs.next();

                //Creates a Actor object using four-params constructor
                Actor obj = new Actor(act_id, fname, lname, new
Timestamp(System.currentTimeMillis()));
                updatePS.setString(1, obj.getFirstName());
                updatePS.setString(2, obj.getLastName());
                updatePS.setInt(3, obj.getActorID());
```

```
                updatePS.executeUpdate();
                rs.close();
                updatePS.close();
                idPs.close();
                conn.close();
            }
        }catch(SQLException ex){
            Logger.getLogger(ActorForm.class.getName()).log(Level.SEVERE,
"Error updating actor data", ex);
            String message = "Error updating actor data: " + ex.getMessage()
            String stackTrace = Arrays.toString(ex.getStackTrace());
            JOptionPane.showMessageDialog(null, message + "\n\n" + stackTrace
"ERROR", JOptionPane.ERROR_MESSAGE);
        }catch(java.lang.NumberFormatException ex){
            Logger.getLogger(ActorForm.class.getName()).log(Level.SEVERE,
"Invalid Input", ex);
            String message = "Invalid Input: " + ex.getMessage();
            String stackTrace = Arrays.toString(ex.getStackTrace());
            JOptionPane.showMessageDialog(null, message + "\n\n" + stackTrace
"ERROR", JOptionPane.ERROR_MESSAGE);
        }
    }

    private static HashMap<String, String> read_inputs(ActorForm frm) {
        HashMap<String, String> input_data = new HashMap<>();
        String act_id = String.valueOf(frm.getJCBActorID().getSelectedItem()
        String fname = frm.getJTFFirstName().getText();
        String lname = frm.getJTFLastName().getText();

        // Validate user input
        int act_id_int = 0;
        try {
            act_id_int = Integer.parseInt(act_id);
            if (act_id_int <= 0) {
                throw new IllegalArgumentException("Actor ID cannot be negat
or zero");
            }
        } catch (NumberFormatException ex) {
            JOptionPane.showMessageDialog(frm, "Invalid Actor ID: " + act_id,
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        } catch (IllegalArgumentException ex) {
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        }

        if (fname == null || fname.isEmpty()) {
            JOptionPane.showMessageDialog(frm, "First name cannot be empty",
            "Error", JOptionPane.ERROR_MESSAGE);
            throw new IllegalArgumentException("First name cannot be empty");
        }
```

```
        if (lname == null || lname.isEmpty()) {
            JOptionPane.showMessageDialog(frm, "Last name cannot be empty",
            "Error", JOptionPane.ERROR_MESSAGE);
            throw new IllegalArgumentException("Last name cannot be empty");
        }

        input_data.put("act_id", act_id);
        input_data.put("fname", fname);
        input_data.put("lname", lname);

        return input_data;
    }

    private static void edit_actual(ActorForm frm){
        try{
            HashMap<String, String> input_data = read_inputs(frm);
            int act_id = Integer.parseInt(input_data.get("act_id"));
            String fname = input_data.get("fname");
            String lname = input_data.get("lname");

            update_row_by_actor_id(frm, act_id, fname, lname);

            //Refreshes all objects on form
            refresh_controls(frm);

        }catch(SQLException ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    private static void enable_controls(boolean state, ActorForm frm){
        frm.getJBFirst().setEnabled(state);
        frm.getJBPrev().setEnabled(state);
        frm.getJBNext().setEnabled(state);
        frm.getJBLast().setEnabled(state);
        frm.getJBInsert().setEnabled(state);
        frm.getJBDelete().setEnabled(state);
        frm.getJTFActorID().setEnabled(state);
    }

    public static void edit_handler(ActorForm frm){
        if(frm.getJBEdit().getText().equals("EDIT")){
            frm.getJBEdit().setText("CONFIRM");

            // Disables controls
            enable_controls(false, frm);
        }

        else {
            frm.getJBEdit().setText("EDIT");

            // Actual editing
```

```
            edit_actual(frm);

            //Enables controls
            enable_controls(true, frm);
        }
    }

    public static void edit_database_from_jtable(TableModelEvent e, ActorForm
frm){
        if (e.getType() == TableModelEvent.UPDATE) {
            int row = e.getFirstRow();
            TableModel model = (TableModel)e.getSource();
            int act_id = Integer.parseInt(String.valueOf(model.getValueAt(row,
0)));
            String fullname = (String) model.getValueAt(row, 1);
            String[] names = fullname.split(" ");

            try{
                update_row_by_actor_id(frm, act_id, names[0], names[1]);

                //Refreshes all objects on form
                refresh_controls(frm);

            } catch (SQLIntegrityConstraintViolationException ex) {
                Logger.getLogger(ActorForm.class.getName()).log(Level.SEVERE,
"Duplicate entry", ex);
                JOptionPane.showMessageDialog(frm, "Error: Duplicate entry\n"
ex.getMessage());
            } catch (SQLSyntaxErrorException ex) {
                Logger.getLogger(ActorForm.class.getName()).log(Level.SEVERE,
"Invalid SQL syntax", ex);
                JOptionPane.showMessageDialog(frm, "Error: Invalid SQL
syntax\n" + ex.getMessage());
            } catch (SQLException ex) {
                Logger.getLogger(ActorForm.class.getName()).log(Level.SEVERE,
"Database error", ex);
                JOptionPane.showMessageDialog(frm, "Error: Database error\n"
ex.getMessage());
            }
        }
    }

    //Inserts new row into actor table
    private static void insert_row(ActorForm frm) throws SQLException{
        HashMap<String, String> input_data = read_inputs(frm);
        int act_id = Integer.parseInt(input_data.get("act_id"));
        String fname = input_data.get("fname");
        String lname = input_data.get("lname");

        // SQL insert statement
        String sql = """
            INSERT INTO actor(first_name, last_name) VALUES(?, ?)""";
```

```
        try(Connection conn = getConnection();
         PreparedStatement pstmt = conn.prepareStatement(sql)){

            //Creates a Actor object three-params constructor
            Actor obj = new Actor(fname, lname, new
Timestamp(System.currentTimeMillis()));
            pstmt.setString(1,obj.getFirstName());
            pstmt.setString(2,obj.getLastName());

            //Executes the sql insert statement
            pstmt.executeUpdate();
        } catch (SQLException ex) {

Logger.getLogger(ActorForm.class.getName()).log(Level.SEVERE, "Database
error", ex);
            JOptionPane.showMessageDialog(frm, "Error: Database error\n"
+ ex.getMessage());
        }
    }

    private static void insert_actual(ActorForm frm){
        try{
            insert_row(frm);

            //Refreshes table and comboboxes
            refresh_controls(frm);

        }catch(SQLException ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    public static void insert_handler(ActorForm frm){
        if(frm.getJBInsert().getText().equals("INSERT")){
            frm.getJBInsert().setText("CONFIRM");

            //Disables jbEdit
            frm.getJBEdit().setEnabled(false);

            // Disables controls
            enable_controls(false, frm);

            // Clears controls
            clear_controls(frm);

            // Enables
            frm.getJBInsert().setEnabled(true);
        }

        else {
            frm.getJBInsert().setText("INSERT");
```

```
            // Actual insertion
            insert_actual(frm);

            //Enables jbEdit
            frm.getJBEdit().setEnabled(true);

            //Enables controls
            enable_controls(true, frm);
        }
    }

    public static void delete_handler(ActorForm frm){
        int dialogButton = JOptionPane.YES_NO_OPTION;
        int act_id =
Integer.parseInt(String.valueOf(frm.getJCBActorID().getSelectedItem()));

        String message = String.format("Are you sure you want to delete
the row Actor ID: %d)", act_id);
        int answer = JOptionPane.showConfirmDialog(frm, message,
"DELETING ROW OF DATA", dialogButton);

        if(answer == JOptionPane.YES_OPTION){
            String query = """
                DELETE FROM actor WHERE actor_id = ?""";
            try(Connection conn = getConnection();
                PreparedStatement ps = conn.prepareStatement(query)){
                // Use PreparedStatement to avoid SQL injection attacks
                ps.setInt(1, act_id);
                ps.executeUpdate();

                // Refresh table and comboboxes
                refresh_controls(frm);

            } catch (SQLException ex){
                JOptionPane.showMessageDialog(frm, ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
            }
        }
    }
}
```

This is the full version of **ActorForm.java**:

```
package sakila;
import java.awt.Toolkit;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.IOException;
import java.text.ParseException;
import java.util.logging.Level;
```

```
import java.util.logging.Logger;
import javax.swing.JButton;
import javax.swing.JComboBox;
import javax.swing.JLabel;
import javax.swing.JTable;
import javax.swing.JTextField;

public class ActorForm extends javax.swing.JFrame {
    public ActorForm() {
        initComponents();
        Utility.setLookAndFeel(this);
        Actor_Utils.refresh_controls(this);

 this.setIconImage(Toolkit.getDefaultToolkit().getImage(getClass().getResour
;
//        this.setDefaultCloseOperation(this.HIDE_ON_CLOSE);
    }

    //Getter method for jtActor
    public JTable getJTActor(){
        return this.jtActor;
    }

    //Getter method for jtfActorID
    public JTextField getJTFActorID(){
        return this.jtfActorID;
    }

    //Getter method for jtfLastUpdate
    public JTextField getJTFLastUpdate(){
        return this.jtfLastUpdate;
    }

    //Getter method for jtfFirstName
    public JTextField getJTFFirstName(){
        return this.jtfFirstName;
    }

    //Getter method for jtfLastName
    public JTextField getJTFLastName(){
        return this.jtfLastName;
    }

    //Getter method for jtableActor
    public JTable getJTableActor(){
        return this.jtActor;
    }

    //Getter method for jcbActorID
    public JComboBox getJCBActorID(){
        return this.jcbActorID;
    }
```

```
    //Getter method for jcbFullName
    public JComboBox getJCBFullName(){
        return this.jcbFullName;
    }

    //Getter method for jbEdit
    public JButton getJBEdit(){
        return this.jbEdit;
    }

    //Getter method for jbInsert
    public JButton getJBInsert(){
        return this.jbInsert;
    }

    //Getter method for jbDelete
    public JButton getJBDelete(){
        return this.jbDelete;
    }

    //Getter method for jbFirst
    public JButton getJBFirst(){
        return this.jbFirst;
    }

    //Getter method for jbPrev
    public JButton getJBPrev(){
        return this.jbPrev;
    }

    //Getter method for jbNext
    public JButton getJBNext(){
        return this.jbNext;
    }

    //Getter method for jbLast
    public JButton getJBLast(){
        return this.jbLast;
    }

    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">
    private void initComponents() {
        //...
        pack();
    }// </editor-fold>

    private void jcbActorIDActionPerformed(java.awt.event.ActionEvent evt) {
        Actor_Utils.jcbActor_handler(this, this.getJCBActorID());
    }

    private void jbNextActionPerformed(java.awt.event.ActionEvent evt) {
        Actor_Utils.show_next_row(this);
```

```
    }

    private void jbFirstActionPerformed(java.awt.event.ActionEvent evt) {
       Actor_Utils.show_first_row(this);
    }

    private void jbPrevActionPerformed(java.awt.event.ActionEvent evt) {
        Actor_Utils.show_prev_row(this);
    }

    private void jbInsertActionPerformed(java.awt.event.ActionEvent evt) {
        Actor_Utils.insert_handler(this);
    }

    private void jbLastActionPerformed(java.awt.event.ActionEvent evt) {
       Actor_Utils.show_last_row(this);
    }

    private void jbEditActionPerformed(java.awt.event.ActionEvent evt) {
        Actor_Utils.edit_handler(this);
    }

    private void jbDeleteActionPerformed(java.awt.event.ActionEvent evt) {
        Actor_Utils.delete_handler(this);
    }

    private void jcbFullNameActionPerformed(java.awt.event.ActionEvent evt) {
        Actor_Utils.jcbActor_handler(this, this.getJCBFullName());
    }

    private void jtActorMousePressed(java.awt.event.MouseEvent evt) {
        Actor_Utils.mouse_pressed_handler(this);
    }

    private void jtActorMouseClicked(java.awt.event.MouseEvent evt) {
        // instantiate ActorTableModelListener and add it as a listener to th
        ActorTableModelListener tableModelListener = new ActorTableModelListe
this);
        this.getJTActor().getModel().addTableModelListener(tableModelListener
    }

    public static void main(String args[]) {
        try {
            for (javax.swing.UIManager.LookAndFeelInfo info :
javax.swing.UIManager.getInstalledLookAndFeels()) {
                if ("Nimbus".equals(info.getName())) {
                    javax.swing.UIManager.setLookAndFeel(info.getClassName())
                    break;
                }
            }
        } catch (ClassNotFoundException ex) {
```

```
java.util.logging.Logger.getLogger(ActorForm.class.getName()).log(java.util.l
null, ex);
        } catch (InstantiationException ex) {

java.util.logging.Logger.getLogger(ActorForm.class.getName()).log(java.util.l
null, ex);
        } catch (IllegalAccessException ex) {

java.util.logging.Logger.getLogger(ActorForm.class.getName()).log(java.util.l
null, ex);
        } catch (javax.swing.UnsupportedLookAndFeelException ex) {

java.util.logging.Logger.getLogger(ActorForm.class.getName()).log(java.util.l
null, ex);
        }
        /* Create and display the form */
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new ActorForm().setVisible(true);
            }
        });
    }

    // Variables declaration - do not modify
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JLabel jLabel3;
    private javax.swing.JLabel jLabel4;
    private javax.swing.JLabel jLabel5;
    private javax.swing.JScrollPane jScrollPane1;
    private javax.swing.JButton jbDelete;
    private javax.swing.JButton jbEdit;
    private javax.swing.JButton jbFirst;
    private javax.swing.JButton jbInsert;
    private javax.swing.JButton jbLast;
    private javax.swing.JButton jbNext;
    private javax.swing.JButton jbPrev;
    private javax.swing.JComboBox<String> jcbActorID;
    private javax.swing.JComboBox<String> jcbFullName;
    private javax.swing.JTable jtActor;
    private javax.swing.JTextField jtfActorID;
    private javax.swing.JTextField jtfFirstName;
    private javax.swing.JTextField jtfLastName;
    private javax.swing.JTextField jtfLastUpdate;
    // End of variables declaration
}
```

LANGUAGE FORM LANGUAGE FORM

CREATING AND POPULATING LANGUAGE TABLE

CREATING AND POPULATING LANGUAGE TABLE

Step 1	Create a new class named **Query_Language**. The class contains five private static fields: **sql_min**, **sql_max**, **sql_id**, **sql_name**, and **sql_language**. **sql_min** is a SQL query string that selects the minimum value of the **language_id** column from the **language** table. **sql_max** is a SQL query string that selects the maximum value of the **language_id** column from the

language table. **sql_id** is a parameterized SQL query string that selects all columns from the **language** table where the **language_id** column matches a specified parameter. **sql_name** is a parameterized SQL query string that selects all columns from the **language** table where the **name** column matches a specified parameter.

sql_language is a SQL query string that creates a new **language** table with columns for **language_id**, **name**, and **last_update**, where **language_id** is an auto-incrementing primary key and **last_update** is a timestamp that defaults to the current timestamp and updates on each subsequent update to the row.

The class also contains getter methods for each of the five SQL query strings, which allow other classes to access them.

```
1  package sakila;
2
3  public class Query_Language {
4      private static final String
5  sql_min = "SELECT MIN(language_id)
6  FROM language";
7      private static final String
8  sql_max = "SELECT MAX(language_id)
9  FROM language";
10     private static final String sql_id =
11
12 "SELECT * FROM language WHERE language_id  =
13 ?";
14     private static final String sql_name =
15 "SELECT * FROM language WHERE name = ?";
16
17     private static final String
18 sql_language = """
19         CREATE TABLE language (
20           language_id TINYINT
21 UNSIGNED NOT NULL AUTO_INCREMENT,
22           name CHAR(20) NOT NULL,
23           last_update TIMESTAMP NOT
24 NULL DEFAULT CURRENT_TIMESTAMP ON
25 UPDATE CURRENT_TIMESTAMP,
26           PRIMARY KEY (language_id)
27         ) ENGINE=InnoDB DEFAULT
28 CHARSET=utf8mb4;""";
29
30     //Getter methods
31     public static String
   get_sql_min() {
```

```
32          return sql_min;
33      }
34
35      public static String
36  get_sql_max() {
37          return sql_max;
38      }

        public static String
    get_sql_id() {
            return sql_id;
        }

        public static String
    get_sql_name() {
            return sql_name;
        }

        public static String
    get_sql_language() {
            return sql_language;
        }
    }
```

Step 2

Then, create a public class named **Language** in which each instance variable represents each column in **language** table in the database. It contains three private instance variables: **lang_id**, **name**, and **last_update**.

The class contains three constructors. The default constructor initializes the **lang_id**, **name**, and **last_update** variables with default values. The two-parameter constructor initializes the **name** and **last_update** variables with the specified values. The three-parameter constructor initializes all three variables with the specified values.

The class also contains getter and setter methods for each instance variable, which allow other classes to access and modify them. The **setLanguageID()** method checks that the id parameter is greater than zero before setting the **lang_id** variable. The **setName()** method checks that the name parameter is not null, not empty, and not longer than 20 characters before setting the name variable. The **setLastUpdate()** method checks that the date parameter is not null before setting the **last_update** variable.

The class overrides the **hashCode()** and **equals()** methods to compare **Language** objects based on their **lang_id**, **name**, and **last_update** variables. The class also overrides the **toString()** method to provide a string representation of a **Language** object.

```
1  package sakila;
2  import java.util.Objects;
3  import java.sql.Timestamp;
4
5  public class Language {
6      private int lang_id;
7      private String name;
8      private Timestamp last_update;
9
10     //Default constructor
11     Language(){
12         this(1, "Language xxxe", new
13 Timestamp(System.currentTimeMillis()));
14     }
15
16     //Two-params constructor
17     Language(String name, Timestamp lu)
18 {
19         setName(name);
20         setLastUpdate(lu);
21     }
22
23     //Three-params constructor
24     Language(int id, String name,
25 Timestamp lu){
26         this(name, lu);
27         setLanguageID(id);
28     }
29
30     // Getter methods
31     public int getLanguageID() {return
32 lang_id;}
33     public String getName() {return
34 name;}
35     public Timestamp getLastUpdate()
36 {return last_update;}
37
38     //Setter methods
39     public void setLanguageID(int id)
40 {
41         if (id <= 0) {
42             throw new
43 IllegalArgumentException("Language ID
44 must be greater than zero.");
45         }
```

```
46          this.lang_id = id;
47      }
48
49      public void setName(String name) {
50          if (name == null ||
51  name.trim().isEmpty()) {
52              throw new
53  IllegalArgumentException("Name cannot
54  be null or empty");
55          }
56          if (name.length() > 20) {
57              throw new
58  IllegalArgumentException("Name cannot
59  be longer than 20 characters");
60          }
61          this.name = name;
62      }
63
64      public void setLastUpdate(Timestamp
65  date){
66          if (date == null) {
67          throw new
68  IllegalArgumentException("Date cannot
69  be null");
70          }
71          this.last_update = date;
72      }
73
74      // Override the hashCode() method
75      @Override
76      public int hashCode() {
77          return Objects.hash(lang_id,
78  name, last_update);
79      }
80
81
82      // Override the equals() method
        @Override
        public boolean equals(Object o) {
            if (this == o) return true;
            if (o == null || getClass() !=
  o.getClass()) return false;
            Language lang = (Language) o;
            return lang_id == lang.lang_id
  &&
                    Objects.equals(name,
  lang.name) &&

  Objects.equals(last_update,
  lang.last_update);
        }

        @Override
        public String toString(){
```

```
        return "\nLanguage ID   : " +
getLanguageID() +
                "\nName          : " +
getName() +
                "\nLast Update   : " +
getLastUpdate();
    }
}
```

Step 3

Create a new public class named **Language_Utils**. It extends the **Utility** class. It contains utility methods for working with **language** table. Here is a summary of the methods in this class:

- **create_language_table()** - This method creates the **language** table in the database using SQL code stored in the **Query_Language** class.
- **populate_language_table()** - This method inserts some sample data into the **language** table.
- **read_language_table()** - This method reads the contents of the **language** table and prints the data to the console.

This class also includes some constants and variables used by these methods, as well as some import statements for classes used in the methods (e.g. java.sql., javax.swing.).

```
1  package sakila;
2  import java.util.logging.Level;
3  import java.util.logging.Logger;
4  import java.sql.*;
5  import java.util.ArrayList;
6  import java.util.Arrays;
7  import java.util.HashMap;
8  import java.util.Objects;
9  import javax.swing.JComboBox;
10 import javax.swing.JOptionPane;
11 import javax.swing.event.TableModelEvent;
12 import javax.swing.table.DefaultTableModel;
13 import javax.swing.table.TableModel;
14
15 public class Language_Utils extends Utility{
16     public static final int FIRST_INDEX = 0;
17     public static final int INVALID_INDEX = -1;
18
19     private static int currentIndex =
20 FIRST_INDEX;
21
```

```
    private static final String SQL_ID =
Query_Language.get_sql_id();

    //Creates language table
    public static void create_language_table() {
        try (Connection conn = getConnection()) {
            Statement stmt =
conn.createStatement();

stmt.addBatch(Query_Language.get_sql_language());
            stmt.executeBatch();

            String message =
String.format("Successfully creates language
table");
            JOptionPane.showMessageDialog(null,
message,

 "INFORMATION",JOptionPane.INFORMATION_MESSAGE);

        } catch (SQLException ex) {
            JOptionPane.showMessageDialog(null,
ex.getMessage(),

 "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    //Populates language table with some rows of
data
    public static void populate_language_table(){
        try(Connection conn = getConnection()){
            String sql = """
                INSERT INTO language(language_id,
name, last_update)
                VALUES(?, ?, ?)""";

            //Creates a new Language class with
default constructor
            PreparedStatement ps1 =
conn.prepareStatement(sql);
            Language obj1 = new Language();
            ps1.setInt(1,obj1.getLanguageID());
            ps1.setString(2,obj1.getName());

ps1.setTimestamp(3,obj1.getLastUpdate());

            // Creates a new Language class with
four-params constructor
            PreparedStatement ps2 =
conn.prepareStatement(sql);
            Language obj2 = new Language(2,
"Bataknese", new
```

```
Timestamp(System.currentTimeMillis()));
            ps2.setInt(1,obj2.getLanguageID());
            ps2.setString(2,obj2.getName());

ps2.setTimestamp(3,obj2.getLastUpdate());

            ps1.executeUpdate();
            ps2.executeUpdate();

        }catch(SQLException ex){
            JOptionPane.showMessageDialog(null,
ex.getMessage(),

 "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    //Reads the content of language table
    public static void read_language_table(){
        try(Connection conn = getConnection()){
            Statement stmt =
conn.createStatement();
            ResultSet rs =
stmt.executeQuery("SELECT * FROM language");

            while(rs.next()){
                int act_id =
rs.getInt("language_id");
                String name =
rs.getString("name");
                Timestamp lu =
rs.getTimestamp("last_update");

                //Creates an Language object
using three-params constructor
                Language obj = new
Language(act_id, name, lu);
                System.out.println(obj);
            }
            rs.close();
            stmt.close();

        }catch(SQLException ex){
            JOptionPane.showMessageDialog(null,
ex.getMessage(),

 "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }
}
```

Step 4	In the driver class, **Sakila.java**, invoke **create_language_table()**, **populate_language_table()**, and **read_language_table()** as shown in line 12 – 14:

```
1  package sakila;
2
3  public class Sakila {
4      public static void main(String[]
5  args) {
6  //        Utility.testConnection();
7  //
8  Actor_Utils.create_actor_table();
9  //
10 Actor_Utils.populate_actor_table();
11 //        Actor_Utils.read_actor_table();
12 //        ActorForm frm = new
13 ActorForm();
14 //        frm.setVisible(true);
15
16

   Language_Utils.create_language_table();

   Language_Utils.populate_language_table();

   Language_Utils.read_language_table();
       }
   }
```

Run project to see the result in console:

```
Language ID   : 1
Name          : Language xxxe
Last Update   : 2023-04-22 13:26:10.0

Language ID   : 2
Name          : Bataknese
Last Update   : 2023-04-22 13:26:10.0
```

DESIGNING GUI
DESIGNING GUI

Step 1	In the project, create a new **JFrame Form** and name it as **LanguageForm.java**. In the Design tab, add three **JLabels** to the form and set their

	corresponding **text** properties as LANGUAGE ID, NAME, and LAST UPDATE.
Step 2	Then, add three **JTextField** to the form and set their corresponding **Variable Name** as **jtfLanguageID**, **jtfName**, and **jtfLastUpdate**.
Step 3	Then, add seven **JButton** to the form and set their corresponding **Variable Name** as **jbFirst**, **jbPrev**, **jbNext**, **jbLast**, **jbEdit**, **jbInsert**, and **jbDelete**. Set their corresponding **text** properties as FIRST, PREV, NEXT, LAST, EDIT, INSERT, and DELETE.
Step 4	Then, add two **JComboBoxes** to the form and set their corresponding **Variable Name** as **jcbLanguageID** and **jcbName**.
Step 5	Lastly, add a new **JTable** to the form set set its **Variable Name** as **jtLanguage**. Then, right-click on it, then choose **Table Contents...** and set the number of columns to 3 and the number of rows to 25.
Step 6	In the driver class, **Sakila.java**, create a new object of **LanguageForm** class using its default constructor as shown in 15 – 16:

```
1  package sakila;
2
3  public class Sakila {
4      public static void main(String[]
5  args) {
6  //        Utility.testConnection();
7  //
8  Actor_Utils.create_actor_table();
9  //
10 Actor_Utils.populate_actor_table();
11 //        Actor_Utils.read_actor_table();
12 //        ActorForm frm = new
13 ActorForm();
14 //        frm.setVisible(true);
15
16 //
17 Language_Utils.create_language_table();
18 //
   Language_Utils.populate_language_table();
```

```
        //
Language_Utils.read_language_table();
        LanguageForm frm = new
LanguageForm();
        frm.setVisible(true);
    }
}
```

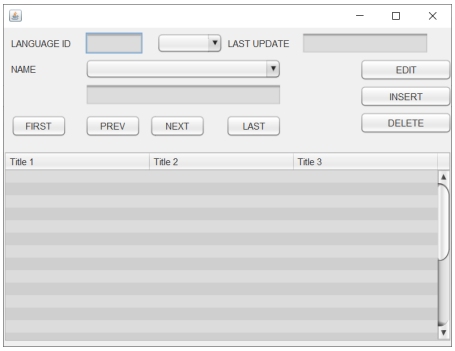


Figure 3.1 The layout of language form

Step 8

In **LanguageForm**'s constructor, invoke **setLookAndFeel()** to set the look and feel of the form as shown in line 17.

```
1  package sakila;
2
3  import java.awt.Toolkit;
4  import java.awt.event.ActionEvent;
5  import
6  java.awt.event.ActionListener;
7  import javax.swing.JButton;
8  import javax.swing.JComboBox;
9  import javax.swing.JMenuItem;
10 import javax.swing.JOptionPane;
11 import javax.swing.JPopupMenu;
12 import javax.swing.JTable;
13 import javax.swing.JTextField;
14
15 public class LanguageForm extends
16 javax.swing.JFrame {
17     public LanguageForm() {
18         initComponents();
19
20 Utility.setLookAndFeel(this);
    }
    //...
}
```

Run the project to see the language form as shown in Figure 3.1.

Step 9	In **LanguageForm.java**, define **getter()** method for every object in the form and place them outside and beneath its default constructor:

```
1     //Getter method for
2  jtfLanguageID
3     public JTextField
4  getJTFLanguageID(){
5           return this.jtfLangID;
6     }
7
8     //Getter method for
9  jtfLastUpdate
10    public JTextField
11 getJTFLastUpdate(){
12          return this.jtfLastUpdate;
13    }
14
15    //Getter method for jtfName
16    public JTextField getJTFName(){
17          return this.jtfName;
18    }
19
20    //Getter method for jtLang
21    public JTable getJTLanguage(){
22          return this.jtLanguage;
23    }
24
25    //Getter method for jcbLangID
26    public JComboBox
27 getJCBLanguageID(){
28          return this.jcbLanguageID;
29    }
30
31    //Getter method for jcbName
32    public JComboBox getJCBName(){
33
34
35
36
37
38
39
40
41
42
43
44
45
46
```

```
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
```

```
        return this.jcbName;
    }

    //Getter method for jbEdit
    public JButton getJBEdit(){
        return this.jbEdit;
    }

    //Getter method for jbInsert
    public JButton getJBInsert(){
        return this.jbInsert;
    }

    //Getter method for jbDelete
    public JButton getJBDelete(){
        return this.jbDelete;
    }

    //Getter method for jbFirst
    public JButton getJBFirst(){
        return this.jbFirst;
    }

    //Getter method for jbPrev
    public JButton getJBPrev(){
        return this.jbPrev;
    }

    //Getter method for jbNext
    public JButton getJBNext(){
        return this.jbNext;
    }

    //Getter method for jbLast
    public JButton getJBLast(){
        return this.jbLast;
    }
```

POPULATING TABLE AND COMBOBOXES

POPULATING TABLE AND COMBOBOXES

Step 1	In **Language_Utils.java**, add two new methods: **get_language_list()** ar methods are used to display the content of the **language** table in a **JTable**

get_language_list() method takes in a **LanguageForm** object, a SQL qu returns an **ArrayList** of **Language** objects obtained from executing the The item parameter is used as a filter in the SQL query, to only select r item value. If item is "none", all rows are selected.

show_table_language() method takes in a **LanguageForm** object and parameters. It creates a **DefaultTableModel** object and sets the colum **LanguageForm** object to the headers defined in the header array. It th adds each **Language** object as a row to the **DefaultTableModel**. Fir **DefaultTableModel**.

```
1     private static ArrayList<Language>
2  get_language_list(LanguageForm frm, String sql, String item){
3          ArrayList<Language> list = new ArrayList<>();
4          Connection conv = null;
5
6          try(Connection conn = getConnection();
7              PreparedStatement ps = conn.prepareStatement(sql)){
8              if (item.equalsIgnoreCase("none")==false) {
9                   ps.setString(1,item);
10             }
11             ResultSet rs = ps.executeQuery();
12
13             Language obj;
14             while(rs.next()){
15                 //Using three-params constructor
16                 obj = new Language(rs.getInt("language_id"),
17                     rs.getString("name"),
18                     rs.getTimestamp("last_update"));
19
20                 list.add(obj);
21             }
22         }catch (SQLException ex){
23             JOptionPane.showMessageDialog(frm, ex.getMessage(),
24                 "ERROR",JOptionPane.ERROR_MESSAGE);
25         }
26         return list;
27     }
28
29     private static void show_table_language(LanguageForm frm,
30 ArrayList<Language> list) throws SQLException{
31         DefaultTableModel model = new DefaultTableModel(0,0);
32
33         String header[] = {"Language ID", "Name", "Last Update"};
34
35
36 model.setColumnIdentifiers(set_column_header(frm.getJTLanguage(),
37 header));
38         frm.getJTLanguage().setModel(model);
39
40         Object[] row = new Object[3];
41
```

```
        for(int i=0; i<list.size(); i++){
            row[0] = list.get(i).getLanguageID();
            row[1] = list.get(i).getName();
            row[2] = list.get(i).getLastUpdate();

            model.addRow(row);
        }
    }
```

Step 2

In **Language_Utils.java**, define **refresh_controls()** method. The **Language_Utils** class is used to update the language form with the la summary of what the method does:

1. Sets the location and title of the language form.
2. Calls the **table_renderer()** method to make the rows in the **langu**
3. Calls the **get_language_list()** method to retrieve a list of language
4. Calls the **show_table_language()** method to display the list of lan
5. Calls the **populate_combobox()** method to populate the **jcbLar** with data from the database.

Overall, the **refresh_controls()** method updates the language form with ensures that the **language** table and combo boxes are populated with the

```
    public static void
refresh_controls(LanguageForm frm){

frm.setLocationRelativeTo(null);
        frm.setTitle("LANGUAGE
FORM");

        //Shows the content of
language table and populates
combobox
        try{
            //Makes alternating
color for table rows

table_renderer(frm.getJTLanguage());

            //Populates table
            ArrayList<Language> list
= get_language_list(frm, "SELECT *
FROM language", "none");
            show_table_language(frm,
list);

            //Populates
jcbLanguageID
            String sql_id = "SELECT
language_id FROM language ORDER BY
```

```
28 language_id";
29
30 populate_combobox(sql_id,
   frm.getJCBLanguageID(), frm);


            //Populates getJCBName()
            String sql_name =
 "SELECT DISTINCT name FROM language
ORDER BY name";


populate_combobox(sql_name,
frm.getJCBName(), frm);

        }catch (SQLException ex){

JOptionPane.showMessageDialog(frm,
ex.getMessage(),

 "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }
```

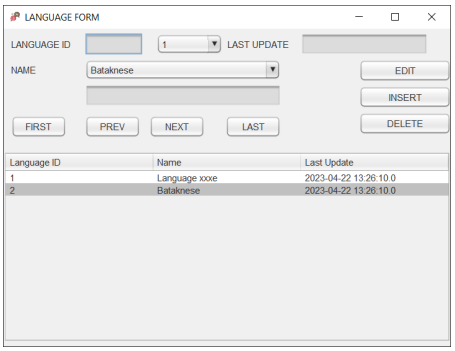


Figure 3.2 The content of **language** table displaye

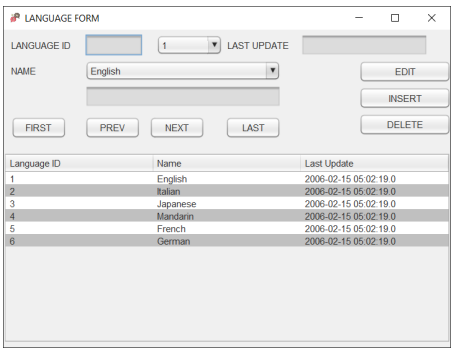


Figure 3.3 The the content of **language** table in original **Sakila** databas

jtLanguage

Step 3	In **LanguageForm**'s default constructor, the **Language_Utils.refresh_** populates the controls in the **LanguageForm** with data from a database from the **Language_Utils** class. The **this.setIconImage()** method sets the icon of the **La** **this.setDefaultCloseOperation(this.HIDE_ON_CLOSE)** method se **LanguageForm** to hide the form instead of exiting the application when tl

```
1  package sakila;
2  import java.awt.Toolkit;
3  import java.awt.event.ActionEvent;
4  import java.awt.event.ActionListener;
5  import java.io.IOException;
6  import java.text.ParseException;
7  import java.util.logging.Level;
8  import java.util.logging.Logger;
9  import javax.swing.JButton;
10 import javax.swing.JComboBox;
11 import javax.swing.JLabel;
12 import javax.swing.JTable;
13 import javax.swing.JTextField;
14
15 public class LanguageForm extends javax.swing.JFrame {
16     public LanguageForm() {
17         initComponents();
18         Utility.setLookAndFeel(this);
19         Language_Utils.refresh_controls(this);
20
21  this.setIconImage(Toolkit.getDefaultToolkit().getImage(getClass().
22 ;
23         this.setDefaultCloseOperation(this.HIDE_ON_CLOSE);
24     }
25     //...
26 }
```

Step 4	Run the project to see the content of **language** table displayed in **jtActor** If you use the data from **Sakila** MySQL database available in the int **language** table displayed in **jtLanguage** as shown in Figure 3.3.

DISPLAYING AND NAVIGATING DATA ROW BY ROW

DISPLAYING AND NAVIGATING DATA ROW BY ROW

Step 1	In **Language_Utils** class, define two new methods named **clear_co**

display_language_data().

The **clear_controls()** method is used to clear the input fields in the **La** when needed. It sets the text of the three input fields (**jtfLanguageID**, **j** **jtfLastUpdate**) to an empty string.

The **display_language_data()** method is used to display the data of a sir in the **LanguageForm**. It takes a SQL query as input, which should returr of data for a language, and an item that is used to populate a parameter The method executes the query and displays the resulting data in the inpu form. If the query returns no rows, the method clears the input fiel **clear_controls()** method.

The **find_combo_value_selected()** method is called by **display_langua** set the selected item in the **jcbLanguageID** and **jcbName** combo boxes language ID or name of the language being displayed.

```
1     private static void clear_controls(LanguageForm frm){
2         frm.getJTFLanguageID().setText("");
3         frm.getJTFName().setText("");
4         frm.getJTFLastUpdate().setText("");
5     }
6
7    //Displays language data result row by row
8     private static <T> void display_language_data(LanguageForm frm,
9 T item){
10        try(Connection conn = getConnection()){
11            PreparedStatement ps = conn.prepareStatement(sql);
12            ps.setObject(1,item);
13            ResultSet rs = ps.executeQuery();
14
15            if (!rs.next()) {
16                // no row found, clear the form fields
17                clear_controls(frm);
18                return;
19            }
20
21            do{
22
23 frm.getJTFLanguageID().setText(String.valueOf(rs.getInt("language_i
24                frm.getJTFName().setText(rs.getString("name"));
25
26 frm.getJTFLastUpdate().setText(String.valueOf(rs.getTimestamp("last
27
28                // Determines item selected from jcbLanguageID
29                find_combo_value_selected(frm.getJCBLanguageID(),
30 rs.getInt("language_id"));
31
32                // Determines item selected from jcbName
33                find_combo_value_selected(frm.getJCBName(),
34 rs.getString("name"));
```

```
            }while(rs.next());

            rs.close();
            ps.close();
        }catch(SQLException ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }
```

Step 2

In the same class, define another method named **jcbLanguage_handler** changes to the language ID and name combo boxes in the **LanguageForm** the selected item from the combo box, and then determines which combo the event by checking if the combo box is equal to the language ID comb name combo box. Depending on which combo box triggered the event, the the corresponding SQL query from the **Query_Language** class and pas **display_language_data()** method along with the selected **display_language_data()** method then retrieves the language data from based on the selected item and updates the text fields and combo **LanguageForm**.

```
    public static void
jcbLanguage_handler(LanguageForm frm,
JComboBox<String> jcb) {
        Object item =
jcb.getSelectedItem();
        String sql = "";
        if
(jcb.equals(frm.getJCBLanguageID()))
{
            sql =
Query_Language.get_sql_id();
        } else if
(jcb.equals(frm.getJCBName())) {
            sql =
Query_Language.get_sql_name();
        }

        display_language_data(frm,
sql, item);
    }
```

Step 3

In **LanguageForm**, double click on **jcbLanguageID** and **jcbName** com define their corresponding event handler as follows:

```
    private void
jcbLanguageIDActionPerformed(java.awt.event.ActionEvent
```

```
3   evt) {
4           Language_Utils.jcbLanguage_handler(this,
5   this.jcbLanguageID);
6       }
7
        private void
    jcbNameActionPerformed(java.awt.event.ActionEvent evt)
    {
            Language_Utils.jcbLanguage_handler(this,
    this.jcbName);
        }
```

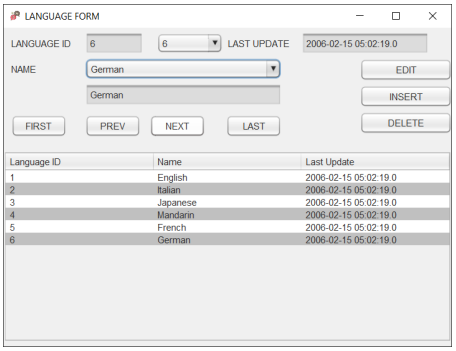


Figure 3.4 Displaying row by row the content of **language** tab

These are event listener methods for the two combo boxes in the **Langua** When an item is selected in the combo box, the c **jcbLanguage_handler()** method from the **Language_Utils** class is call the event. The **jcbLanguage_handler()** method gets the selected item fro box and determines which SQL query to execute based on the combo box t the event. It then calls the **display_language_data()** method to display data in the form.

Step 4	Run the project. Choose one of items in **jcbLanguageID** and/or **jcbName** to see row by row the content of **language** table as shown in Figure 3.4.
Step 5	Define four navigating methods in **Language_Utils** class. These method navigate through the rows of data displayed in the language form. **show_first_row()** displays the first row of data by getting the item at the the jcbLanguageID combo box, and passing it to the **display_language_da** It also sets the **currentIndex** to the first index. **show_last_row()** displays the last row of data by getting the item at the the **jcbLanguageID** combo box, and passing it to the **display_lang** method. It also sets the **currentIndex** to the last index.

show_prev_row displays the previous row of data by decrementing the **cu** and then getting the item at the current index of the **jcbLanguageID** co passing it to the **display_language_data()** method. If the current index is first index, it sets the current index to the first index and returns.

show_next_row() displays the next row of data by incrementing the curr then getting the item at the current index of the **jcbLanguageID** co passing it to the **display_language_data()** method. If the current index is the last index, it sets the current index to the last index and returns.

```
1     public static void show_first_row(LanguageForm frm){
2         String item =
3 String.valueOf(frm.getJCBLanguageID().getItemAt(FIRST_INDEX));
4         display_language_data(frm, SQL_ID, item);
5         currentIndex = FIRST_INDEX;
6     }
7 
8     public static void show_last_row(LanguageForm frm){
9         int endIndex = frm.getJCBLanguageID().getItemCount() -
10 1;
11        String item =
12 String.valueOf(frm.getJCBLanguageID().getItemAt(endIndex));
13        display_language_data(frm, SQL_ID, item);
14        currentIndex = endIndex;
15    }
16 
17    public static void show_prev_row(LanguageForm frm){
18        currentIndex--;
19        if(currentIndex < FIRST_INDEX){
20            currentIndex = FIRST_INDEX;
21            return;
22        }
23        String item =
24 String.valueOf(frm.getJCBLanguageID().getItemAt(currentIndex));
25        display_language_data(frm, SQL_ID, item);
26    }
27 
28    public static void show_next_row(LanguageForm frm){
29        int endIndex = frm.getJCBLanguageID().getItemCount() -
30 1;
31        currentIndex++;
32        if(currentIndex > endIndex){
33            currentIndex = endIndex;
34            return;
35        }
36        String item =
   String.valueOf(frm.getJCBLanguageID().getItemAt(currentIndex));
        display_language_data(frm, SQL_ID, item);
    }
```

Step 6

Then in **LanguageForm**, double click on each navigation buttons to corresponding event handler:

```
    private void jbNextActionPerformed(java.awt.event.ActionEvent evt) {
        Language_Utils.show_next_row(this);
    }

    private void jbFirstActionPerformed(java.awt.event.ActionEvent evt) {
       Language_Utils.show_first_row(this);
    }

    private void jbPrevActionPerformed(java.awt.event.ActionEvent evt) {
        Language_Utils.show_prev_row(this);
    }

    private void jbLastActionPerformed(java.awt.event.ActionEvent evt) {
       Language_Utils.show_last_row(this);
    }
```

These are the action listener methods for the "Next", "First", "Prev", and " respectively. They call the corresponding methods in the **Language_U** show the next/first/previous/last row in the **language** table.

Step 7

Run the project. Click on one or more navigation buttons to see the resul Figure 3.5.

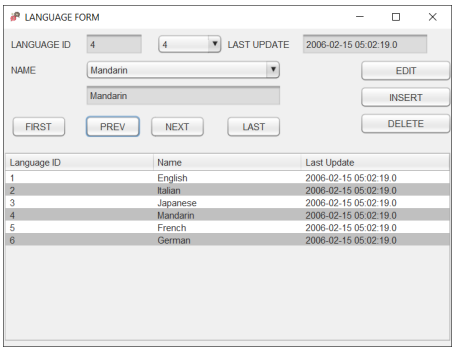


Figure 3.5 User clicks on one or more navigation buttons on langua

Step

Define **mouse_pressed_handler()** method in **Language_Utils** class.

8

mouse press event on a form's language table (**jtLanguage**). It takes a sin
frm of type **LanguageForm**, which represents the form that contains t
Objects.requireNonNull() method is called to ensure that the **frm** arg
null.

The method first retrieves the index of the currently selected row in t
table using the **getSelectedRow()** method. If no row is selected, it
informational message using **JOptionPane.showMessageDialog()** and re

If a row is selected, it retrieves the **Connection** object using the **getC**
method, which is assumed to return a valid connection to a database. It t
the value of the first column in the selected row using the **getValueAt()** n
table model, and converts it to a **String** using **String.valueOf()**.
Finally, it calls the **display_language_data()** method, passing in the frm
SQL query string obtained from **Query_Language.get_sql_id()**, and
retrieved from the table. This method is assumed to display the languag
form. If a **SQLException** is thrown while executing the SQL query
exception using the **Logger** class and displays an error me
JOptionPane.showMessageDialog(). The error message includes t
message and stack trace.

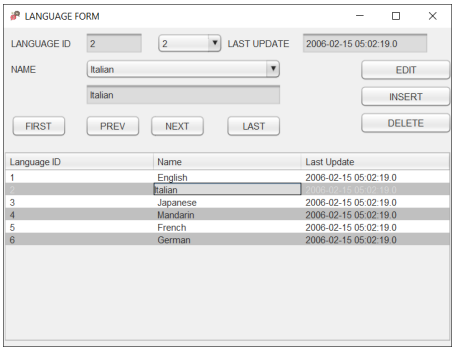


Figure 3.6 User double-clicks on any row in **jtLanguage**

```
1     public static void mouse_pressed_handler(LanguageForm frm) {
2         Objects.requireNonNull(frm, "frm must not be null");
3
4         int selectedIndex = frm.getJTLanguage().getSelectedRow();
5         if (selectedIndex == -1) {
6             JOptionPane.showMessageDialog(frm, "Please select a row to view its
7 data.",
8                 "No row selected", JOptionPane.INFORMATION_MESSAGE)
9             return;
10        }
11
12        try (Connection conn = getConnection()) {
13            String id =
14 String.valueOf(frm.getJTLanguage().getModel().getValueAt(selectedIn
15 0));
16
```

```
            // Displays language data
            display_language_data(frm, Query_Language.get_sql_id(),
id);

        } catch (SQLException ex) {

Logger.getLogger(LanguageForm.class.getName()).log(Level.SEVERE, "E
displaying language data", ex);
            String message = "Error displaying language data: " +
ex.getMessage();
            String stackTrace = Arrays.toString(ex.getStackTrace())
            JOptionPane.showMessageDialog(frm, message + "\n\n" +
stackTrace, "ERROR", JOptionPane.ERROR_MESSAGE);
        }

    }
```

Step 9

Right click on **jtLanguage**. Then, choose **Events > Mouse > mousePre** its event handler:

```
    private void
jtLanguageMousePressed(java.awt.event.MouseEvent
evt) {

 Language_Utils.mouse_pressed_handler(this);
    }
```

Step 10

Run the project. Double click on any row in **jtLanguage** table. You corresponding row in **language** table displayed in textfields and combobo in Figure 3.6.

UPDATING RECORD
UPDATING RECORD

Step 1

In **Language_Utils** class, define a new method named **update_row_by_lang_id()**. It updates a row in the **language** table in a database, identified by the given **lang_id**, with the given name value. It takes a **LanguageForm** object frm, an int **lang_id**, and a String **name** as arguments.

The method first obtains a **Connection** object using the **getConnection()** method, which is assumed to return a valid connection to a database. It then creates two **PreparedStatement** objects: **idPs** to retrieve the language ID from the **language** table, and **updatePS** to update the row with the given **lang_id** with the new name value.

The **idPs** query uses a prepared statement with a single parameter (**lang_id**) and checks if a row with that **language_id** exists in the **language** table. If not, it displays an error message and returns.

If a row with the given **lang_id** is found, the **updatePS** query updates the **name** column for that row with the given name value. It creates a new **Language** object with the **lang_id** and **name** values, and sets the corresponding parameter values in the prepared statement. It then executes the update query using **executeUpdate()**.

If an **SQLException** is thrown while executing the queries, it logs the exception and displays an error message using **JOptionPane.showMessageDialog()**. The error message includes the exception message and stack trace.

If a **NumberFormatException** is thrown, it logs the exception and displays an error message indicating that the input is invalid. The error message also includes the exception message and stack trace.

Finally, the method closes the **ResultSet**, **PreparedStatement**, and **Connection** objects.

```
1   //Updates row of data in language tabel by language_id
2   public static void update_row_by_lang_id(LanguageForm frm,
3   int lang_id, String name) throws SQLException{
4           Connection conn = getConnection();
5           ResultSet rs = null;
6           String query_id = "SELECT language_id FROM language WHERE language_id =
7   ?";
8           String update_query = """
9                   UPDATE language SET name = ? WHERE language_id =
10  ?""";
11          try(PreparedStatement idPs =
12  conn.prepareStatement(query_id,
13
14  ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_READ_ONLY);
15          PreparedStatement updatePS =
16  conn.prepareStatement(update_query,
17                  ResultSet.TYPE_SCROLL_SENSITIVE,
18  ResultSet.CONCUR_UPDATABLE))
19          {
20              idPs.setInt(1,lang_id);
```

```
            if(!idPs.execute()){
                String message = "Can't find language_id " +
lang_id;

                JOptionPane.showMessageDialog(frm, message,
                    "ERROR",JOptionPane.ERROR_MESSAGE);
            } else{
                rs = idPs.getResultSet();
                rs.next();

                //Creates a Language object using three-params
constructor
                Language obj = new Language(lang_id, name, new
Timestamp(System.currentTimeMillis()));
                updatePS.setString(1, obj.getName());
                updatePS.setInt(2, obj.getLanguageID());

                updatePS.executeUpdate();
                rs.close();
                updatePS.close();
                idPs.close();
                conn.close();
            }
        }catch(SQLException ex){

Logger.getLogger(LanguageForm.class.getName()).log(Level.SEVERE,
"Error updating language data", ex);
            String message = "Error updating language data: " +
ex.getMessage();
            String stackTrace =
Arrays.toString(ex.getStackTrace());
            JOptionPane.showMessageDialog(null, message + "\n\n"
+ stackTrace, "ERROR", JOptionPane.ERROR_MESSAGE);
        }catch(java.lang.NumberFormatException ex){

Logger.getLogger(LanguageForm.class.getName()).log(Level.SEVERE,
"Invalid Input", ex);
            String message = "Invalid Input: " +
ex.getMessage();
            String stackTrace =
Arrays.toString(ex.getStackTrace());
            JOptionPane.showMessageDialog(null, message + "\n\n"
+ stackTrace, "ERROR", JOptionPane.ERROR_MESSAGE);
        }
    }
```

Step 2	Then in the same class, define a new method **read_inputs()**. It reads and validates user input from the **LanguageForm** form and returns a **HashMap** containing the input data. It takes a **LanguageForm** object as input.

The method first creates an empty **HashMap** object to store the input data. It then reads the **lang_id** and **name** fields from the form using the **getJCBLanguageID()** and **getJTFName()** methods, respectively.

Next, the method attempts to parse the **lang_id** input as an integer and checks that it is greater than zero. If the input is not a valid integer or is less than or equal to zero, an exception is thrown and an error message is displayed to the user.

Similarly, if the **name** input is empty or null, an exception is thrown and an error message is displayed to the user.

Finally, if the input data is valid, the method adds the **lang_id** and **name** values to the **HashMap** object and returns it.

```
1   private static HashMap<String, String>
2   read_inputs(LanguageForm frm) {
3           HashMap<String, String> input_data = new
4   HashMap<>();
5           String lang_id =
6   String.valueOf(frm.getJCBLanguageID().getSelectedItem());
7           String name = frm.getJTFName().getText();
8
9           // Validate user input
10          int lang_id_int = 0;
11          try {
12              lang_id_int = Integer.parseInt(lang_id);
13              if (lang_id_int <= 0) {
14                  throw new
15  IllegalArgumentException("Language ID cannot be negative
16  or zero");
17              }
18          } catch (NumberFormatException ex) {
19              JOptionPane.showMessageDialog(frm, "Invalid
20  Language ID: " + lang_id,
21              "Error", JOptionPane.ERROR_MESSAGE);
22              throw ex;
23          } catch (IllegalArgumentException ex) {
24              JOptionPane.showMessageDialog(frm,
25  ex.getMessage(),
26              "Error", JOptionPane.ERROR_MESSAGE);
27              throw ex;
28          }
29
30          if (name == null || name.isEmpty()) {
31              JOptionPane.showMessageDialog(frm, "Name
32  cannot be empty",
33              "Error", JOptionPane.ERROR_MESSAGE);
34              throw new IllegalArgumentException("Name
    cannot be empty");
```

```
        }

        input_data.put("lang_id", lang_id);
        input_data.put("name", name);

        return input_data;
    }
```

Step 3

Still in the same class, define another method named **edit_actual()**. It is responsible for updating a row of data in the **language** table with the new name entered by the user.

First, it calls the **read_inputs()** method to get the language ID and name entered by the user and validates the input. If the input is valid, it calls the **update_row_by_lang_id()** method to update the row in the database with the new name.

If the update is successful, it calls the **refresh_controls()** method to update all the objects on the form. If there is an SQL exception, it displays an error message to the user.

```
1   private static void
2  edit_actual(LanguageForm frm){
3          try{
4              HashMap<String, String>
5  input_data = read_inputs(frm);
6              int lang_id =
7  Integer.parseInt(input_data.get("lang_id"));
8              String name =
9  input_data.get("name");
10
11             update_row_by_lang_id(frm,
12 lang_id, name);
13
14             //Refreshes all objects on form
15             refresh_controls(frm);
16
          }catch(SQLException ex){

JOptionPane.showMessageDialog(frm,
ex.getMessage(),

 "ERROR",JOptionPane.ERROR_MESSAGE);
          }
    }
```

Step

Lastly, define two new methods named **enable_controls()** and

4	**edit_handler()**. The **edit_handler()** method is responsible for handling the edit button click event on the **LanguageForm**. When the user clicks the edit button, the method first checks whether the text on the button is "EDIT" or "CONFIRM". If the text is "EDIT", it changes the text on the button to "CONFIRM" and disables some of the form controls. If the text is "CONFIRM", it changes the text on the button back to "EDIT", enables the form controls, and calls the **edit_actual()** method to update the selected row in the database.
	
	`private static void enable_controls(boolean state, LanguageForm frm){`
	`frm.getJBFirst().setEnabled(state);`
	`frm.getJBPrev().setEnabled(state);`
	`frm.getJBNext().setEnabled(state);`
	`frm.getJBLast().setEnabled(state);`
	`frm.getJBInsert().setEnabled(state);`
	`frm.getJBDelete().setEnabled(state);`
	
	`frm.getJTFLanguageID().setEnabled(state);`
	`}`
	
	`public static void edit_handler(LanguageForm frm){`
	
	`if(frm.getJBEdit().getText().equals("EDIT") ){`
	
	`frm.getJBEdit().setText("CONFIRM");`
	
	`// Disables controls`
	`enable_controls(false, frm);`
	`}`
	
	`else {`
	`frm.getJBEdit().setText("EDIT");`
	
	`// Actual editing`
	`edit_actual(frm);`
	
	`//Enables controls`
	`enable_controls(true, frm);`
	`}`
	`}`

Step 5	Run the project. Choose **language_id** using **jcbLanguageID** or **jcbName** combobox. Or, you can choose one of rows in **jtLanguage** (in this case, **language_id = 1**). Then, click on EDIT button as shown in Figure 3.7.

	Edit the language name. Then, click on CONFIRM button. The edited row had been saved into **language** table as shown in Figure 3.8.

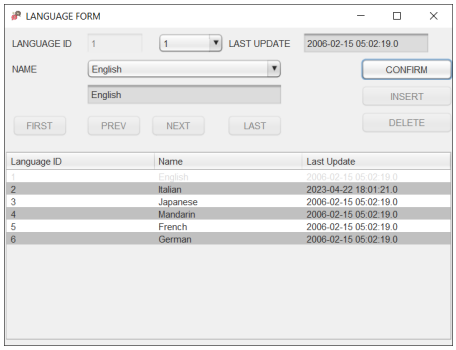


Figure 3.7 The language form is in editing state

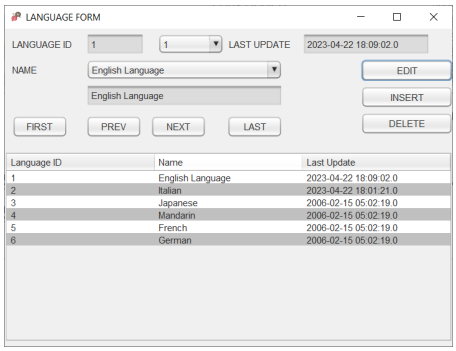


Figure 3.8 The edited row had been saved into database

UPDATING RECORD DIRECTLY ON JTABLE

UPDATING RECORD DIRECTLY ON JTABLE

Step 1	In **Language_Utility** class, define a new method named **edit_database_from_jtable()**. This method handles the event when the user updates a row in the **JTable**. It first checks if the event type is a table update event. If it is, it retrieves the row number and the updated data from the table model. It then calls the **update_row_by_lang_id()** method to update the corresponding row in the database. If the update is successful, it refreshes all the objects on the form. If there is an SQL exception, it catches the exception and displays an error message to the user. The type of exception is also logged using the **Logger** class. The different types of exceptions that can be caught are:

- **SQLIntegrityConstraintViolationException**: This exception is thrown when there is a violation of a database integrity constraint, such as a unique or primary key constraint.
- **SQLSyntaxErrorException**: This exception is thrown when there is a syntax error in the SQL statement.
- **SQLException**: This is a generic exception for errors that occur when communicating with the database.

```
    public static void edit_database_from_jtable(TableModelEvent
e, LanguageForm frm){
        if (e.getType() == TableModelEvent.UPDATE) {
            int row = e.getFirstRow();
            TableModel model = (TableModel)e.getSource();
            int lang_id =
Integer.parseInt(String.valueOf(model.getValueAt(row, 0)));
            String name = (String) model.getValueAt(row, 1);

            try{
                update_row_by_lang_id(frm, lang_id, name);

                //Refreshes all objects on form
                refresh_controls(frm);

            } catch (SQLIntegrityConstraintViolationException
ex) {

Logger.getLogger(LanguageForm.class.getName()).log(Level.SEVERE,
"Duplicate entry", ex);
                JOptionPane.showMessageDialog(frm, "Error:
Duplicate entry\n" + ex.getMessage());
            } catch (SQLSyntaxErrorException ex) {

Logger.getLogger(LanguageForm.class.getName()).log(Level.SEVERE,
"Invalid SQL syntax", ex);
                JOptionPane.showMessageDialog(frm, "Error:
Invalid SQL syntax\n" + ex.getMessage());
            } catch (SQLException ex) {

Logger.getLogger(LanguageForm.class.getName()).log(Level.SEVERE,
"Database error", ex);
                JOptionPane.showMessageDialog(frm, "Error:
Database error\n" + ex.getMessage());
            }
        }
    }
```

Step 2

Create a new public class named **LanguageTableModelListener**. It implements the **TableModelListener** interface. It has two instance variables, **jt** of type **JTable** and **frm** of type **LanguageForm**. The

constructor takes a **JTable** and a **LanguageForm** object as parameters and assigns them to the instance variables.

The **tableChanged()** method is called when the table model is changed. It calls the **edit_database_from_jtable()** method of the **Language_Utils** class with the event object **e** and the **frm** object as parameters. It then checks if there is an active cell editor and stops editing if there is one.

```
1
2
3
4
5
6
```

```
7  package sakila;
8  import javax.swing.event.TableModelEvent;
9  import
10 javax.swing.event.TableModelListener;
11 import javax.swing.JTable;
12
13 public class LanguageTableModelListener
14 implements TableModelListener {
15     private final JTable jt;
16     private final LanguageForm frm;
17
18     public
19 LanguageTableModelListener(JTable jt,
20 LanguageForm frm) {
21         this.jt = jt;
22         this.frm = frm;
23     }

     @Override
     public void
 tableChanged(TableModelEvent e) {

 Language_Utils.edit_database_from_jtable(e,
 frm);

         if (jt.getCellEditor() != null) {

 jt.getCellEditor().stopCellEditing();
         }
     }
 }
```

Step 3

Right click on **jtLanguage**. Define its event handler:

```
1     private void jtLangua
2         // instantiate La
3 to the table model
4         LanguageTableMode
5 LanguageTableModelListene
6
7  this.getJTLanguage().get
;
    }
```

A mouse click event is being clicked, a **LanguageTableM** to the table model. This liste by the user.

Step 4	Run the project. Click on ar that you want to edit. The corresponding cell. The edite

INSERTING NEW RECORD
INSERTING NEW RECORD

Step 1	In **Language_Utils** class, define a method named **insert_row()**. This method inserts a new row into the **language** table of the Sakila database. It reads the input data from the form, which is the name of the language, and uses a prepared statement to execute an SQL INSERT statement. Here's a breakdown of the code: 1. The method signature specifies that it may throw a **SQLException**, which is a checked exception that indicates a problem with the database operation. 2. The method reads the input data from the form using the **read_inputs()** method, which returns a **HashMap** containing the name of the language. 3. The SQL INSERT statement is assigned to a string variable **sql**. The statement contains a parameter placeholder (?) for the name of the language. 4. A connection is obtained using the **getConnection()** method, which is a helper method that returns a connection to the database. 5. A prepared statement is created using the connection and the SQL INSERT statement. The **setString()** method is called to set the value of the parameter to the name of the language. 6. The **executeUpdate()** method of the prepared statement is called to execute the SQL INSERT statement and insert the new row into the **language** table. 7. If an **SQLException** occurs, the method catches the exception, logs it, and displays an error message to the user. Overall, this method provides a simple way to insert new languages into the Sakila database using data from the form.

```
1     //Inserts new row into language table
2     private static void insert_row(LanguageForm frm) throws
3 SQLException{
4         HashMap<String, String> input_data = read_inputs(frm);
```

```
        String name = input_data.get("name");

        // SQL insert statement
        String sql = """
            INSERT INTO language(name) VALUES(?)""";

        try(Connection conn = getConnection();
         PreparedStatement pstmt = conn.prepareStatement(sql)){

            //Creates a Language object two-params constructor
            Language obj = new Language(name, new
Timestamp(System.currentTimeMillis()));
            pstmt.setString(1,obj.getName());

            //Executes the sql insert statement
            pstmt.executeUpdate();
        } catch (SQLException ex) {

Logger.getLogger(LanguageForm.class.getName()).log(Level.SEVERE,
"Database error", ex);
            JOptionPane.showMessageDialog(frm, "Error: Database
error\n" + ex.getMessage());
        }
    }
```

Step 2

Still in **Language_Utils.java**, define **insert_actual()** and **insert_handler()** methods. It handles the insertion of a new language into the database. It first checks if the text of the "Insert" button is "INSERT". If it is, then it changes the text to "CONFIRM" and disables the "Edit" button, all controls, and the language ID and name combo boxes. It then clears all the controls and enables the "Insert" button. If the "Insert" button text is "CONFIRM", then it changes the text to "INSERT" and calls the **insert_actual()** method which actually inserts the new language into the database. After the insertion is successful, it refreshes the controls, enables the "Edit" button, and enables all the controls and the language ID and name combo boxes. If there is an error during the insertion process, it displays an error message using a **JOptionPane**.

```
    private static void
insert_actual(LanguageForm frm){
        try{
            insert_row(frm);

            //Refreshes table and comboboxes
            refresh_controls(frm);

        }catch(SQLException ex){

```

```
            JOptionPane.showMessageDialog(frm,
ex.getMessage(),

 "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    public static void
insert_handler(LanguageForm frm){

 if(frm.getJBInsert().getText().equals("INSERT")
){

frm.getJBInsert().setText("CONFIRM");

            //Disables jbEdit
            frm.getJBEdit().setEnabled(false);

            // Disables controls
            enable_controls(false, frm);

frm.getJCBLanguageID().setEnabled(false);
            frm.getJCBName().setEnabled(false);

            // Clears controls
            clear_controls(frm);

            // Enables
            frm.getJBInsert().setEnabled(true);
        }

        else {
            frm.getJBInsert().setText("INSERT");

            // Actual insertion
            insert_actual(frm);

            //Enables jbEdit
            frm.getJBEdit().setEnabled(true);

            //Enables controls
            enable_controls(true, frm);

frm.getJCBLanguageID().setEnabled(true);
            frm.getJCBName().setEnabled(true);
        }
    }
```

Step 3

In **LanguageForm.java**, double click on INSERT button to create its event listener:

```
    private void
jbInsertActionPerformed(java.awt.event.ActionEvent
evt) {
        Language_Utils.insert_handler(this);
    }
```

Step 4

Run the project. Click on INSERT button. You will see the state of language form when insertion is in progress as shown in Figure 3.9.

Then, type a language name. Then, click CONFIRM button to save the new record into **language** table as shown in Figure 3.10.

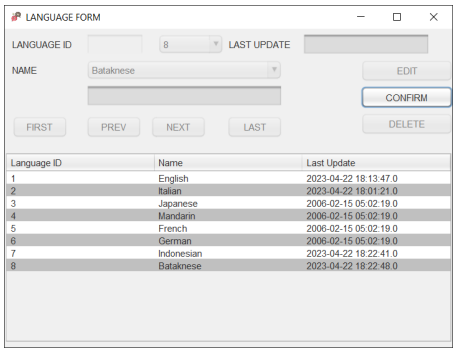


Figure 3.9 When user clicks on INSERT button, the language form will be in state of insertion

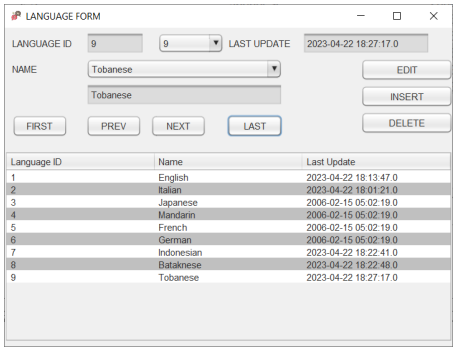


Figure 3.10 The new data had been saved into **language** table

DELETING RECORD
DELETING RECORD

Step 1

Then in **Language_Utils** class, define **delete_handler()** method. It ha
deletion of a row from the **language** table in the Sakila database. It
displaying a confirmation message dialog box to the user to confirm whe

want to delete the row or not. If the user clicks the "Yes" button, it execute DELETE statement to delete the row from the language table corresponding language_id value selected in the **jcbLanguageID** combo b

The SQL DELETE statement is prepared using a **PreparedStatement** to a injection attacks. The **language_id** value is set as a parameter in the statement using the **setInt()** method. If the deletion is succes **refresh_controls()** method is called to refresh the table and combo box form.

If an exception occurs during the deletion process, an error message dial displayed to the user with the error message obtained from the exception.

```
    public static void delete_handler(LanguageForm frm){
        int dialogButton = JOptionPane.YES_NO_OPTION;
        int lang_id =
Integer.parseInt(String.valueOf(frm.getJCBLanguageID().getSelectedI

        String message = String.format("Are you sure you want to de
row Language ID: %d)", lang_id);
        int answer = JOptionPane.showConfirmDialog(frm, message, "D
ROW OF DATA", dialogButton);

        if(answer == JOptionPane.YES_OPTION){
            String query = """
                DELETE FROM language WHERE language_id = ?""";
            try(Connection conn = getConnection();
                PreparedStatement ps = conn.prepareStatement(query)
                // Use PreparedStatement to avoid SQL injection att
                ps.setInt(1, lang_id);
                ps.executeUpdate();

                // Refresh table and comboboxes
                refresh_controls(frm);

            } catch (SQLException ex){
                JOptionPane.showMessageDialog(frm, ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
            }
        }
    }
```

Step 2

In **LanguageForm.java**, double click on DELETE button to generate listener:

```
    private void
jbDeleteActionPerformed(java.awt.event.ActionEvent
evt) {
        Language_Utils.delete_handler(this);
```

	`}`
Step 3	Run the project. Choose **language_id** using **jcbLanguageID** or j combobox. Then, Click on DELETE button. The corresponding row of data deleted from database.

This is the full version of **Language_Utils.java**:

```
package sakila;
import java.util.logging.Level;
import java.util.logging.Logger;
import java.sql.*;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashMap;
import java.util.Objects;
import javax.swing.JComboBox;
import javax.swing.JOptionPane;
import javax.swing.event.TableModelEvent;
import javax.swing.table.DefaultTableModel;
import javax.swing.table.TableModel;

public class Language_Utils extends Utility{
    public static final int FIRST_INDEX = 0;
    public static final int INVALID_INDEX = -1;

    private static int currentIndex = FIRST_INDEX;
    private static final String SQL_ID = Query_Language.get_sql_id();

    //Creates language table
    public static void create_language_table() {
        try (Connection conn = getConnection()) {
            Statement stmt = conn.createStatement();
            stmt.addBatch(Query_Language.get_sql_language());
            stmt.executeBatch();

            String message = String.format("Successfully creates language
table");
            JOptionPane.showMessageDialog(null, message,
            "INFORMATION",JOptionPane.INFORMATION_MESSAGE);

        } catch (SQLException ex) {
            JOptionPane.showMessageDialog(null, ex.getMessage(),
                "ERROR",JOptionPane.ERROR_MESSAGE);
```

```
        }
    }

    //Populates language table with some rows of data
    public static void populate_language_table(){
        try(Connection conn = getConnection()){
            String sql = """
                INSERT INTO language(language_id, name, last_update)
                VALUES(?, ?, ?)""";

            //Creates a new Language class with default constructor
            PreparedStatement ps1 = conn.prepareStatement(sql);
            Language obj1 = new Language();
            ps1.setInt(1,obj1.getLanguageID());
            ps1.setString(2,obj1.getName());
            ps1.setTimestamp(3,obj1.getLastUpdate());

            // Creates a new Language class with four-params constructor
            PreparedStatement ps2 = conn.prepareStatement(sql);
            Language obj2 = new Language(2, "Bataknese", new
Timestamp(System.currentTimeMillis()));
            ps2.setInt(1,obj2.getLanguageID());
            ps2.setString(2,obj2.getName());
            ps2.setTimestamp(3,obj2.getLastUpdate());

            ps1.executeUpdate();
            ps2.executeUpdate();

        }catch(SQLException ex){
            JOptionPane.showMessageDialog(null, ex.getMessage(),
                "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    //Reads the content of language table
    public static void read_language_table(){
        try(Connection conn = getConnection()){
            Statement stmt = conn.createStatement();
            ResultSet rs = stmt.executeQuery("SELECT * FROM language");

            while(rs.next()){
                int act_id = rs.getInt("language_id");
                String name = rs.getString("name");
                Timestamp lu = rs.getTimestamp("last_update");

                //Creates an Language object using three-params constructor
                Language obj = new Language(act_id, name, lu);
                System.out.println(obj);
            }
            rs.close();
            stmt.close();

        }catch(SQLException ex){
```

```
            JOptionPane.showMessageDialog(null, ex.getMessage(),
                "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    private static ArrayList<Language> get_language_list(LanguageForm frm,
String sql, String item){
        ArrayList<Language> list = new ArrayList<>();
        Connection conv = null;

        try(Connection conn = getConnection();
            PreparedStatement ps = conn.prepareStatement(sql)){
            if (item.equalsIgnoreCase("none")==false) {
                 ps.setString(1,item);
            }
            ResultSet rs = ps.executeQuery();

            Language obj;
            while(rs.next()){
                //Using three-params constructor
                obj = new Language(rs.getInt("language_id"),
                    rs.getString("name"),
                    rs.getTimestamp("last_update"));

                list.add(obj);
            }
        }catch (SQLException ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
                "ERROR",JOptionPane.ERROR_MESSAGE);
        }
        return list;
    }

    private static void show_table_language(LanguageForm frm,
ArrayList<Language> list) throws SQLException{
        DefaultTableModel model = new DefaultTableModel(0,0);

        String header[] = {"Language ID", "Name", "Last Update"};

        model.setColumnIdentifiers(set_column_header(frm.getJTLanguage(),
header));
        frm.getJTLanguage().setModel(model);

        Object[] row = new Object[3];

        for(int i=0; i<list.size(); i++){
            row[0] = list.get(i).getLanguageID();
            row[1] = list.get(i).getName();
            row[2] = list.get(i).getLastUpdate();

            model.addRow(row);
        }
    }
```

```
    public static void refresh_controls(LanguageForm frm){
        frm.setLocationRelativeTo(null);
        frm.setTitle("LANGUAGE FORM");

        //Shows the content of language table and populates combobox
        try{
            //Makes alternating color for table rows
            table_renderer(frm.getJTLanguage());

            //Populates table
            ArrayList<Language> list = get_language_list(frm, "SELECT * FROM
language", "none");
            show_table_language(frm, list);

            //Populates jcbLanguageID
            String sql_id = "SELECT language_id FROM language ORDER BY
language_id";
            populate_combobox(sql_id, frm.getJCBLanguageID(), frm);


            //Populates getJCBName()
            String sql_name = "SELECT DISTINCT name FROM language ORDER BY
name";

            populate_combobox(sql_name, frm.getJCBName(), frm);

        }catch (SQLException ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
            "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    private static void clear_controls(LanguageForm frm){
        frm.getJTFLanguageID().setText("");
        frm.getJTFName().setText("");
        frm.getJTFLastUpdate().setText("");
    }

   //Displays language data result row by row
    private static <T> void display_language_data(LanguageForm frm, String sq
T item){
        try(Connection conn = getConnection()){
            PreparedStatement ps = conn.prepareStatement(sql);
            ps.setObject(1,item);
            ResultSet rs = ps.executeQuery();

            if (!rs.next()) {
                // no row found, clear the form fields
                clear_controls(frm);
                return;
            }

            do{
```

```
frm.getJTFLanguageID().setText(String.valueOf(rs.getInt("language_id")));
                frm.getJTFName().setText(rs.getString("name"));

frm.getJTFLastUpdate().setText(String.valueOf(rs.getTimestamp("last_update"))

                // Determines item selected from jcbLanguageID
                find_combo_value_selected(frm.getJCBLanguageID(),
rs.getInt("language_id"));

                // Determines item selected from jcbName
                find_combo_value_selected(frm.getJCBName(),
rs.getString("name"));
            }while(rs.next());

            rs.close();
            ps.close();
        }catch(SQLException ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    public static void jcbLanguage_handler(LanguageForm frm, JComboBox<String
jcb) {
        Object item = jcb.getSelectedItem();
        String sql = "";
        if (jcb.equals(frm.getJCBLanguageID())) {
            sql = Query_Language.get_sql_id();
        } else if (jcb.equals(frm.getJCBName())) {
            sql = Query_Language.get_sql_name();
        }

        display_language_data(frm, sql, item);
    }

    public static void show_first_row(LanguageForm frm){
        String item =
String.valueOf(frm.getJCBLanguageID().getItemAt(FIRST_INDEX));
        display_language_data(frm, SQL_ID, item);
        currentIndex = FIRST_INDEX;
    }

    public static void show_last_row(LanguageForm frm){
        int endIndex = frm.getJCBLanguageID().getItemCount() - 1;
        String item =
String.valueOf(frm.getJCBLanguageID().getItemAt(endIndex));
        display_language_data(frm, SQL_ID, item);
        currentIndex = endIndex;
    }

    public static void show_prev_row(LanguageForm frm){
        currentIndex--;
```

```
        if(currentIndex < FIRST_INDEX){
            currentIndex = FIRST_INDEX;
            return;
        }
        String item =
String.valueOf(frm.getJCBLanguageID().getItemAt(currentIndex));
        display_language_data(frm, SQL_ID, item);
    }

    public static void show_next_row(LanguageForm frm){
        int endIndex = frm.getJCBLanguageID().getItemCount() - 1;
        currentIndex++;
        if(currentIndex > endIndex){
            currentIndex = endIndex;
            return;
        }
        String item =
String.valueOf(frm.getJCBLanguageID().getItemAt(currentIndex));
        display_language_data(frm, SQL_ID, item);
    }

    public static void mouse_pressed_handler(LanguageForm frm) {
        Objects.requireNonNull(frm, "frm must not be null");

        int selectedIndex = frm.getJTLanguage().getSelectedRow();
        if (selectedIndex == -1) {
            JOptionPane.showMessageDialog(frm, "Please select a row to view 
data.",
                "No row selected", JOptionPane.INFORMATION_MESSAGE);
            return;
        }

        try (Connection conn = getConnection()) {
            String id =
String.valueOf(frm.getJTLanguage().getModel().getValueAt(selectedIndex, 0));

            // Displays actor data
            display_language_data(frm, Query_Language.get_sql_id(), id);

        } catch (SQLException ex) {
            Logger.getLogger(LanguageForm.class.getName()).log(Level.SEVERE,
"Error displaying language data", ex);
            String message = "Error displaying language data: " +
ex.getMessage();
            String stackTrace = Arrays.toString(ex.getStackTrace());
            JOptionPane.showMessageDialog(frm, message + "\n\n" + stackTrace,
"ERROR", JOptionPane.ERROR_MESSAGE);
        }

    }

    //Updates row of data in language tabel by language_id
```

```
    public static void update_row_by_lang_id(LanguageForm frm, int lang_id,
String name) throws SQLException{
        Connection conn = getConnection();
        ResultSet rs = null;
        String query_id = "SELECT language_id FROM language WHERE language_id
?";
        String update_query = """
            UPDATE language SET name = ? WHERE language_id = ?""";
        try(PreparedStatement idPs = conn.prepareStatement(query_id,

ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_READ_ONLY);
        PreparedStatement updatePS = conn.prepareStatement(update_query,
                ResultSet.TYPE_SCROLL_SENSITIVE, ResultSet.CONCUR_UPDATABLE))
        {
            idPs.setInt(1,lang_id);
            if(!idPs.execute()){
                String message = "Can't find language_id " + lang_id;

                JOptionPane.showMessageDialog(frm, message,
                    "ERROR",JOptionPane.ERROR_MESSAGE);
            } else{
                rs = idPs.getResultSet();
                rs.next();

                //Creates a Language object using three-params constructor
                Language obj = new Language(lang_id, name, new
Timestamp(System.currentTimeMillis()));
                updatePS.setString(1, obj.getName());
                updatePS.setInt(2, obj.getLanguageID());

                updatePS.executeUpdate();
                rs.close();
                updatePS.close();
                idPs.close();
                conn.close();
            }
        }catch(SQLException ex){
            Logger.getLogger(LanguageForm.class.getName()).log(Level.SEVERE,
"Error updating language data", ex);
            String message = "Error updating language data: " +
ex.getMessage();
            String stackTrace = Arrays.toString(ex.getStackTrace());
            JOptionPane.showMessageDialog(null, message + "\n\n" + stackTrace
"ERROR", JOptionPane.ERROR_MESSAGE);
        }catch(java.lang.NumberFormatException ex){
            Logger.getLogger(LanguageForm.class.getName()).log(Level.SEVERE,
"Invalid Input", ex);
            String message = "Invalid Input: " + ex.getMessage();
            String stackTrace = Arrays.toString(ex.getStackTrace());
            JOptionPane.showMessageDialog(null, message + "\n\n" + stackTrace
"ERROR", JOptionPane.ERROR_MESSAGE);
        }
    }
```

```
    private static HashMap<String, String> read_inputs(LanguageForm frm) {
        HashMap<String, String> input_data = new HashMap<>();
        String lang_id =
String.valueOf(frm.getJCBLanguageID().getSelectedItem());
        String name = frm.getJTFName().getText();

        // Validate user input
        int lang_id_int = 0;
        try {
            lang_id_int = Integer.parseInt(lang_id);
            if (lang_id_int <= 0) {
                throw new IllegalArgumentException("Language ID cannot be
negative or zero");
            }
        } catch (NumberFormatException ex) {
            JOptionPane.showMessageDialog(frm, "Invalid Language ID: " +
lang_id,
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        } catch (IllegalArgumentException ex) {
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        }

        if (name == null || name.isEmpty()) {
            JOptionPane.showMessageDialog(frm, "Name cannot be empty",
            "Error", JOptionPane.ERROR_MESSAGE);
            throw new IllegalArgumentException("Name cannot be empty");
        }

        input_data.put("lang_id", lang_id);
        input_data.put("name", name);

        return input_data;
    }

    private static void edit_actual(LanguageForm frm){
        try{
            HashMap<String, String> input_data = read_inputs(frm);
            int lang_id = Integer.parseInt(input_data.get("lang_id"));
            String name = input_data.get("name");

            update_row_by_lang_id(frm, lang_id, name);

            //Refreshes all objects on form
            refresh_controls(frm);

        }catch(SQLException ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
        }
```

```
    }

    private static void enable_controls(boolean state, LanguageForm frm){
        frm.getJBFirst().setEnabled(state);
        frm.getJBPrev().setEnabled(state);
        frm.getJBNext().setEnabled(state);
        frm.getJBLast().setEnabled(state);
        frm.getJBInsert().setEnabled(state);
        frm.getJBDelete().setEnabled(state);
        frm.getJTFLanguageID().setEnabled(state);
    }

    public static void edit_handler(LanguageForm frm){
        if(frm.getJBEdit().getText().equals("EDIT")){
            frm.getJBEdit().setText("CONFIRM");

            // Disables controls
            enable_controls(false, frm);
        }

        else {
            frm.getJBEdit().setText("EDIT");

            // Actual editing
            edit_actual(frm);

            //Enables controls
            enable_controls(true, frm);
        }
    }

    public static void edit_database_from_jtable(TableModelEvent e,
LanguageForm frm){
        if (e.getType() == TableModelEvent.UPDATE) {
            int row = e.getFirstRow();
            TableModel model = (TableModel)e.getSource();
            int lang_id = Integer.parseInt(String.valueOf(model.getValueAt(r
0)));
            String name = (String) model.getValueAt(row, 1);

            try{
                update_row_by_lang_id(frm, lang_id, name);

                //Refreshes all objects on form
                refresh_controls(frm);

            } catch (SQLIntegrityConstraintViolationException ex) {

Logger.getLogger(LanguageForm.class.getName()).log(Level.SEVERE, "Duplicate
entry", ex);
                JOptionPane.showMessageDialog(frm, "Error: Duplicate entry\n'
ex.getMessage());
            } catch (SQLSyntaxErrorException ex) {
```

```
Logger.getLogger(LanguageForm.class.getName()).log(Level.SEVERE, "Invalid SQL
syntax", ex);
                JOptionPane.showMessageDialog(frm, "Error: Invalid SQL
syntax\n" + ex.getMessage());
            } catch (SQLException ex) {

Logger.getLogger(LanguageForm.class.getName()).log(Level.SEVERE, "Database
error", ex);
                JOptionPane.showMessageDialog(frm, "Error: Database error\n"
ex.getMessage());
            }
        }
    }

    //Inserts new row into language table
    private static void insert_row(LanguageForm frm) throws SQLException{
        HashMap<String, String> input_data = read_inputs(frm);
        String name = input_data.get("name");

        // SQL insert statement
        String sql = """
            INSERT INTO language(name) VALUES(?)""";

        try(Connection conn = getConnection();
         PreparedStatement pstmt = conn.prepareStatement(sql)){

            //Creates a Language object two-params constructor
            Language obj = new Language(name, new
Timestamp(System.currentTimeMillis()));
            pstmt.setString(1,obj.getName());

            //Executes the sql insert statement
            pstmt.executeUpdate();
        } catch (SQLException ex) {
            Logger.getLogger(LanguageForm.class.getName()).log(Level.SEVERE,
"Database error", ex);
            JOptionPane.showMessageDialog(frm, "Error: Database error\n" +
ex.getMessage());
        }
    }

    private static void insert_actual(LanguageForm frm){
        try{
            insert_row(frm);

            //Refreshes table and comboboxes
            refresh_controls(frm);

        }catch(SQLException ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
        }
```

```
    }

    public static void insert_handler(LanguageForm frm){
        if(frm.getJBInsert().getText().equals("INSERT")){
            frm.getJBInsert().setText("CONFIRM");

            //Disables jbEdit
            frm.getJBEdit().setEnabled(false);

            // Disables controls
            enable_controls(false, frm);
            frm.getJCBLanguageID().setEnabled(false);
            frm.getJCBName().setEnabled(false);

            // Clears controls
            clear_controls(frm);

            // Enables
            frm.getJBInsert().setEnabled(true);
        }

        else {
            frm.getJBInsert().setText("INSERT");

            // Actual insertion
            insert_actual(frm);

            //Enables jbEdit
            frm.getJBEdit().setEnabled(true);

            //Enables controls
            enable_controls(true, frm);
            frm.getJCBLanguageID().setEnabled(true);
            frm.getJCBName().setEnabled(true);
        }
    }

    public static void delete_handler(LanguageForm frm){
        int dialogButton = JOptionPane.YES_NO_OPTION;
        int lang_id =
Integer.parseInt(String.valueOf(frm.getJCBLanguageID().getSelectedItem()));

        String message = String.format("Are you sure you want to delete the r
Language ID: %d)", lang_id);
        int answer = JOptionPane.showConfirmDialog(frm, message, "DELETING RO
OF DATA", dialogButton);

        if(answer == JOptionPane.YES_OPTION){
            String query = """
                DELETE FROM language WHERE language_id = ?""";
            try(Connection conn = getConnection();
                PreparedStatement ps = conn.prepareStatement(query)){
                // Use PreparedStatement to avoid SQL injection attacks
```

```
                ps.setInt(1, lang_id);
                ps.executeUpdate();

                // Refresh table and comboboxes
                refresh_controls(frm);

            } catch (SQLException ex){
                JOptionPane.showMessageDialog(frm, ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
            }
        }
    }
}
```

This is the full version of **LanguageForm.java**:

```
package sakila;
import java.awt.Toolkit;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.IOException;
import java.text.ParseException;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.JButton;
import javax.swing.JComboBox;
import javax.swing.JLabel;
import javax.swing.JTable;
import javax.swing.JTextField;

public class LanguageForm extends javax.swing.JFrame {
    public LanguageForm() {
        initComponents();
        Utility.setLookAndFeel(this);
        Language_Utils.refresh_controls(this);

 this.setIconImage(Toolkit.getDefaultToolkit().getImage(getClass().getResour
;
//        this.setDefaultCloseOperation(this.HIDE_ON_CLOSE);
    }

    //Getter method for jtfLanguageID
    public JTextField getJTFLanguageID(){
        return this.jtfLangID;
    }

    //Getter method for jtfLastUpdate
    public JTextField getJTFLastUpdate(){
        return this.jtfLastUpdate;
    }
```

```
    //Getter method for jtfName
    public JTextField getJTFName(){
        return this.jtfName;
    }

    //Getter method for jtLang
    public JTable getJTLanguage(){
        return this.jtLanguage;
    }

    //Getter method for jcbLangID
    public JComboBox getJCBLanguageID(){
        return this.jcbLanguageID;
    }

    //Getter method for jcbName
    public JComboBox getJCBName(){
        return this.jcbName;
    }

    //Getter method for jbEdit
    public JButton getJBEdit(){
        return this.jbEdit;
    }

    //Getter method for jbInsert
    public JButton getJBInsert(){
        return this.jbInsert;
    }

    //Getter method for jbDelete
    public JButton getJBDelete(){
        return this.jbDelete;
    }

    //Getter method for jbFirst
    public JButton getJBFirst(){
        return this.jbFirst;
    }

    //Getter method for jbPrev
    public JButton getJBPrev(){
        return this.jbPrev;
    }

    //Getter method for jbNext
    public JButton getJBNext(){
        return this.jbNext;
    }

    //Getter method for jbLast
    public JButton getJBLast(){
        return this.jbLast;
```

```
    }

    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">
    private void initComponents() {
        //...
        pack();
    }// </editor-fold>

    private void jcbLanguageIDActionPerformed(java.awt.event.ActionEvent evt)
        Language_Utils.jcbLanguage_handler(this, this.jcbLanguageID);
    }

    private void jcbNameActionPerformed(java.awt.event.ActionEvent evt) {
        Language_Utils.jcbLanguage_handler(this, this.jcbName);
    }

    private void jbEditActionPerformed(java.awt.event.ActionEvent evt) {
        Language_Utils.edit_handler(this);
    }

    private void jbInsertActionPerformed(java.awt.event.ActionEvent evt) {
        Language_Utils.insert_handler(this);
    }

    private void jbDeleteActionPerformed(java.awt.event.ActionEvent evt) {
        Language_Utils.delete_handler(this);
    }

    private void jbLastActionPerformed(java.awt.event.ActionEvent evt) {
        Language_Utils.show_last_row(this);
    }

    private void jbNextActionPerformed(java.awt.event.ActionEvent evt) {
        Language_Utils.show_next_row(this);
    }

    private void jbPrevActionPerformed(java.awt.event.ActionEvent evt) {
        Language_Utils.show_prev_row(this);
    }

    private void jbFirstActionPerformed(java.awt.event.ActionEvent evt) {
        Language_Utils.show_first_row(this);
    }

    private void jtLanguageMousePressed(java.awt.event.MouseEvent evt) {
        Language_Utils.mouse_pressed_handler(this);
    }

    private void jtLanguageMouseClicked(java.awt.event.MouseEvent evt) {
        // instantiate LanguageTableModelListener and add it as a listener to
```

```
        LanguageTableModelListener tableModelListener = new
LanguageTableModelListener(this.getJTLanguage(), this);
        this.getJTLanguage().getModel().addTableModelListener(tableModelListe
    }

    public static void main(String args[]) {
        try {
            for (javax.swing.UIManager.LookAndFeelInfo info :
javax.swing.UIManager.getInstalledLookAndFeels()) {
                if ("Nimbus".equals(info.getName())) {
                    javax.swing.UIManager.setLookAndFeel(info.getClassName())
                    break;
                }
            }
        } catch (ClassNotFoundException ex) {

java.util.logging.Logger.getLogger(LanguageForm.class.getName()).log(java.ut
null, ex);
        } catch (InstantiationException ex) {

java.util.logging.Logger.getLogger(LanguageForm.class.getName()).log(java.ut
null, ex);
        } catch (IllegalAccessException ex) {

java.util.logging.Logger.getLogger(LanguageForm.class.getName()).log(java.ut
null, ex);
        } catch (javax.swing.UnsupportedLookAndFeelException ex) {

java.util.logging.Logger.getLogger(LanguageForm.class.getName()).log(java.ut
null, ex);
        }
        /* Create and display the form */
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new LanguageForm().setVisible(true);
            }
```

```
        });
    }

    // Variables declaration - do not modify
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JLabel jLabel4;
    private javax.swing.JScrollPane
jScrollPane1;
    private javax.swing.JButton jbDelete;
    private javax.swing.JButton jbEdit;
    private javax.swing.JButton jbFirst;
    private javax.swing.JButton jbInsert;
    private javax.swing.JButton jbLast;
    private javax.swing.JButton jbNext;
    private javax.swing.JButton jbPrev;
    private javax.swing.JComboBox<String>
jcbLanguageID;
    private javax.swing.JComboBox<String>
jcbName;
    private javax.swing.JTable jtLanguage;
    private javax.swing.JTextField jtfLangID;
    private javax.swing.JTextField
jtfLastUpdate;
    private javax.swing.JTextField jtfName;
    // End of variables declaration
}
```

CATEGORY FORM CATEGORY FORM

CREATING AND POPULATING CATEGORY TABLE

CREATING AND POPULATING CATEGORY TABLE

Step 1	Create a new class named **Query_Category**. It contains SQL queries related to the category table in the Sakila database. The **category** table has columns **category_id**, **name**, and **last_update**. The class has four private static final **String** fields that store SQL queries as strings:

- **sql_min**: a query that retrieves the minimum **category_id** value from the **category** table.
- **sql_max**: a query that retrieves the maximum **category_id** value from the **category** table.
- **sql_id**: a parameterized query that retrieves a row from the **category** table with a specific **category_id** value.
- **sql_name**: a parameterized query that retrieves a row from the **category** table with a specific name value.

The class has one private static final **String** field that stores an SQL query as a multi-line string:

- **sql_category**: a query that creates the **category** table with its columns and constraints.

The class also has getter methods for each of the private static final **String** fields, which allows other classes to access the SQL queries stored in this class.

```
1  package sakila;
2
3  public class Query_Category {
4      private static final String
5  sql_min = "SELECT MIN(category_id)
6  FROM category";
7      private static final String
8  sql_max = "SELECT MAX(category_id)
9  FROM category";
10     private static final String sql_id =
11 "SELECT * FROM category WHERE category_id  =
12 ?";
13     private static final String sql_name =
14 "SELECT * FROM category WHERE name = ?";
15
16
17     private static final String
18 sql_category = """
19         CREATE TABLE category (
20           category_id TINYINT
21 UNSIGNED NOT NULL AUTO_INCREMENT,
22           name VARCHAR(25) NOT
23 NULL,
24           last_update TIMESTAMP NOT
25 NULL DEFAULT CURRENT_TIMESTAMP ON
26 UPDATE CURRENT_TIMESTAMP,
27           PRIMARY KEY
28 (category_id)
29         ) ENGINE=InnoDB DEFAULT
30 CHARSET=utf8mb4;""";
```

```
31
32      //Getter methods
33      public static String
34  get_sql_min() {
35          return sql_min;
36      }
37
38      public static String
    get_sql_max() {
          return sql_max;
      }

      public static String
    get_sql_id() {
          return sql_id;
      }

      public static String
    get_sql_name() {
          return sql_name;
      }

      public static String
    get_sql_category() {
          return sql_category;
      }
  }
```

Step 2

Then, create a public class named **Category**. It represents a category entity in the Sakila database. The category has an int category ID (**cat_id**), a String **name**, and a Timestamp **last_update**. Here is a summary of what this class contains:

The class has three private instance variables that store the category ID, name, and last update of a category object.
The class has three constructors:

1. A default constructor that sets the category ID to 1, the name to "Category xxx", and the last update to the current timestamp.
2. A two-params constructor that takes a **String name** and a **Timestamp** last update as parameters and sets the name and last update of a category object.
3. A three-params constructor that takes an int category ID, a **String** name, and a **Timestamp** last update as parameters and calls the two-

params constructor to set the name and last update of a category object, and sets the category ID using the **setCategoryID()** method.

The class has getter methods for each of the private instance variables. The class has setter methods for the category ID, name, and last update, with some input validation.
The class overrides the **hashCode()** and **equals()** methods to ensure that category objects can be compared and hashed properly. The class overrides the **toString()** method to return a string representation of a category object.

```
1  package sakila;
2  import java.util.Objects;
3  import java.sql.Timestamp;
4
5  public class Category {
6      private int cat_id;
7      private String name;
8      private Timestamp last_update;
9
10     //Default constructor
11     Category(){
12         this(1, "Category xxx", new
13 Timestamp(System.currentTimeMillis()));
14     }
15
16     //Two-params constructor
17     Category(String name, Timestamp lu)
18 {
19         setName(name);
20         setLastUpdate(lu);
21     }
22
23     //Three-params constructor
24     Category(int id, String name,
25 Timestamp lu){
26         this(name, lu);
27         setCategoryID(id);
28     }
29
30     // Getter methods
31     public int getCategoryID() {return
32 cat_id;}
33     public String getName() {return
34 name;}
35     public Timestamp getLastUpdate()
36 {return last_update;}
37
38
```

```
    //Setter methods
    public void setCategoryID(int id)
{
        if (id <= 0) {
            throw new
IllegalArgumentException("Category ID must be
greater than zero.");
        }
        this.cat_id = id;
    }

    public void setName(String name) {
        if (name == null ||
name.trim().isEmpty()) {
            throw new
IllegalArgumentException("Name cannot
be null or empty");
        }
        if (name.length() > 25) {
            throw new
IllegalArgumentException("Name cannot
be longer than 25 characters");
        }
        this.name = name;
    }

    public void setLastUpdate(Timestamp
date){
        if (date == null) {
        throw new
IllegalArgumentException("Date cannot
be null");
        }
        this.last_update = date;
    }

    // Override the hashCode() method
    @Override
    public int hashCode() {
        return Objects.hash(cat_id,
name, last_update);
    }

    // Override the equals() method
    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() !=
o.getClass()) return false;
        Category cat = (Category) o;
        return cat_id == cat.cat_id &&
               Objects.equals(name,
cat.name) &&
```

```
Objects.equals(last_update,
cat.last_update);
    }

    @Override
    public String toString(){
        return "\nCategory ID   : " +
getCategoryID() +
                "\nName          : " +
getName() +
                "\nLast Update   : " +
getLastUpdate();
    }
}
```

Step 3

Create a new public class named **Category_Utils**. It extends the **Utility** class. It contains methods for creating, populating, and reading a table called **category** in the database using SQL queries. It also defines constants and variables used in the class.

The class has a constant called **FIRST_INDEX** with a value of 0 and another constant called **INVALID_INDEX** with a value of -1. It also has a static variable called **currentIndex** with an initial value of **FIRST_INDEX**.

The class has a private static final string variable called **SQL_ID** which stores a SQL query obtained from another class called **Query_Category**. This variable is used to retrieve the ID of a category from the database.

The class has a method called **create_category_table()** which creates a **category** table in the database. This method uses a SQL query obtained from **Query_Category** to create the table. If the table is created successfully, a message is displayed using a **JOptionPane**.

The class also has a method called **populate_category_table()** which inserts some rows of data into the category table. This method creates two **Category** objects using the default constructor and a four-parameter constructor respectively. It then uses prepared statements to insert the data into the category table.

The class also has a method called **read_category_table()** which reads the content of the category table and creates **Category** objects for each row of data. It then prints out the information of each **Category** object to the console.

Overall, this class provides utility methods for interacting with the **category** table in a database using SQL queries.

```
1  package sakila;
2  import java.util.logging.Level;
3  import java.util.logging.Logger;
4  import java.sql.*;
5  import java.util.ArrayList;
6  import java.util.Arrays;
7  import java.util.HashMap;
8  import java.util.Objects;
9  import javax.swing.JComboBox;
10 import javax.swing.JOptionPane;
11 import javax.swing.event.TableModelEvent;
12 import javax.swing.table.DefaultTableModel;
13 import javax.swing.table.TableModel;
14
15 public class Category_Utils extends Utility{
16     public static final int FIRST_INDEX = 0;
17     public static final int INVALID_INDEX = -1;
18
19     private static int currentIndex =
20 FIRST_INDEX;
21     private static final String SQL_ID =
22 Query_Category.get_sql_id();
23
24     //Creates category table
25     public static void create_category_table() {
26         try (Connection conn = getConnection()) {
27             Statement stmt =
28 conn.createStatement();
29
30 stmt.addBatch(Query_Category.get_sql_category());
31             stmt.executeBatch();
32
33             String message =
34 String.format("Successfully creates category
35 table");
36             JOptionPane.showMessageDialog(null,
37 message,
38
39  "INFORMATION",JOptionPane.INFORMATION_MESSAGE);
40
41
```

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

 "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    //Populates category table with some rows of
data
    public static void populate_category_table(){
        try(Connection conn = getConnection()){
            String sql = """
                INSERT INTO category(category_id,
name, last_update)
                VALUES(?, ?, ?)""";

            //Creates a new Category class with
default constructor
            PreparedStatement ps1 =
conn.prepareStatement(sql);
            Category obj1 = new Category();
            ps1.setInt(1,obj1.getCategoryID());
            ps1.setString(2,obj1.getName());

ps1.setTimestamp(3,obj1.getLastUpdate());

            // Creates a new Category class with
four-params constructor
            PreparedStatement ps2 =
conn.prepareStatement(sql);
            Category obj2 = new Category(2,
"Ethnics", new
Timestamp(System.currentTimeMillis()));
            ps2.setInt(1,obj2.getCategoryID());
            ps2.setString(2,obj2.getName());

ps2.setTimestamp(3,obj2.getLastUpdate());

            ps1.executeUpdate();
            ps2.executeUpdate();

        }catch(SQLException ex){
            JOptionPane.showMessageDialog(null,
ex.getMessage(),

 "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    //Reads the content of category table
    public static void read_category_table(){
        try(Connection conn = getConnection()){
```

```
            Statement stmt =
conn.createStatement();
            ResultSet rs =
stmt.executeQuery("SELECT * FROM category");

            while(rs.next()){
                int cat_id =
rs.getInt("category_id");
                String name =
rs.getString("name");
                Timestamp lu =
rs.getTimestamp("last_update");

                //Creates a Category object using
three-params constructor
                Category obj = new
Category(cat_id, name, lu);
                System.out.println(obj);
            }
            rs.close();
            stmt.close();

        }catch(SQLException ex){
            JOptionPane.showMessageDialog(null,
ex.getMessage(),

 "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }
}
```

Step 4

In the driver class, **Sakila.java**, invoke **create_category_table()**, **populate_category_table()**, and **read_category_table()** as shown in line 18 – 20:

```
1  package sakila;
2
3  public class Sakila {
4      public static void main(String[]
5  args) {
6  //        Utility.testConnection();
7  //
8  Actor_Utils.create_actor_table();
9  //
10 Actor_Utils.populate_actor_table();
11 //        Actor_Utils.read_actor_table();
12 //        ActorForm frm = new
13 ActorForm();
14 //        frm.setVisible(true);
15
16
```

```
//
Language_Utils.create_language_table();
//
Language_Utils.populate_language_table();
//
Language_Utils.read_language_table();
//        LanguageForm frm = new
LanguageForm();
//        frm.setVisible(true);


Category_Utils.create_category_table();

Category_Utils.populate_category_table();

Category_Utils.read_category_table();
    }
}
```

Run project to see the result in console:

```
Category ID   : 1
Name          : Category xxx
Last Update   : 2023-04-22 19:40:40.0

Category ID   : 2
Name          : Ethnics
Last Update   : 2023-04-22 19:40:40.0
```

DESIGNING GUI
DESIGNING GUI

Step 1	In the project, create a new **JFrame Form** and name it as **CategoryForm.java**. In the Design tab, add three **JLabels** to the form and set their corresponding **text** properties as CATEGORY ID, NAME, and LAST UPDATE.
Step 2	Then, add three **JTextField** to the form and set their corresponding **Variable Name** as **jtfCategoryID**, **jtfName**, and **jtfLastUpdate**.
Step 3	Then, add seven **JButton** to the form and set their corresponding **Variable Name** as **jbFirst**, **jbPrev**, **jbNext**, **jbLast**, **jbEdit**, **jbInsert**, and **jbDelete**. Set their corresponding **text**

	properties as FIRST, PREV, NEXT, LAST, EDIT, INSERT, and DELETE.
Step 4	Then, add two **JComboBoxes** to the form and set their corresponding **Variable Name** as **jcbCategoryID** and **jcbName**.
Step 5	Lastly, add a new **JTable** to the form set set its **Variable Name** as **jtCategory**. Then, right-click on it, then choose **Table Contents...** and set the number of columns to 3 and the number of rows to 25.
Step 6	In the driver class, **Sakila.java**, create a new object of **CategoryForm** class using its default constructor as shown in 21 – 22:

```
1  package sakila;
2
3  public class Sakila {
4      public static void main(String[]
5  args) {
6  //        Utility.testConnection();
7  //
8  Actor_Utils.create_actor_table();
9  //
10 Actor_Utils.populate_actor_table();
11 //        Actor_Utils.read_actor_table();
12 //        ActorForm frm = new
13 ActorForm();
14 //        frm.setVisible(true);
15
16 //
17 Language_Utils.create_language_table();
18 //
19 Language_Utils.populate_language_table();
20 //
21 Language_Utils.read_language_table();
22 //        LanguageForm frm = new
23 LanguageForm();
24 //        frm.setVisible(true);

   //
   Category_Utils.create_category_table();
   //
   Category_Utils.populate_category_table();
   //
   Category_Utils.read_category_table();
```

```
        CategoryForm frm = new
CategoryForm();
        frm.setVisible(true);
    }
}
```

Step 8

In **CategoryForm**'s constructor, invoke **setLookAndFeel()** to set the look and feel of the form as shown in line 17.

```
1  package sakila;
2
3  import java.awt.Toolkit;
4  import java.awt.event.ActionEvent;
5  import
6  java.awt.event.ActionListener;
7  import javax.swing.JButton;
8  import javax.swing.JComboBox;
9  import javax.swing.JMenuItem;
10 import javax.swing.JOptionPane;
11 import javax.swing.JPopupMenu;
12 import javax.swing.JTable;
13 import javax.swing.JTextField;
14
15 public class CategoryForm extends
16 javax.swing.JFrame {
17     public CategoryForm() {
18         initComponents();
19
20 Utility.setLookAndFeel(this);
    }
    //...
}
```

Run the project to see the category form as shown in Figure 4.1.

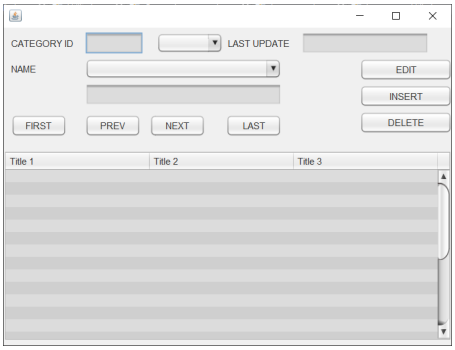


Figure 4.1 The layout of category form

Step 9	In **CategoryForm.java**, define **getter()** method for every object in the form and place them outside and beneath its default constructor:

```
1     //Getter method for
2  jtfCategoryID
3     public JTextField
4  getJTFCategoryID(){
5           return this.jtfCategoryID;
6     }
7
8     //Getter method for
9  jtfLastUpdate
10    public JTextField
11 getJTFLastUpdate(){
12          return this.jtfLastUpdate;
13    }
14
15    //Getter method for jtfName
16    public JTextField getJTFName(){
17          return this.jtfName;
18    }
19
20    //Getter method for jtCategory
21    public JTable getJTCategory(){
22          return this.jtCategory;
23    }
24
25    //Getter method for
26 jcbCategoryID
27    public JComboBox
28 getJCBCategoryID(){
29          return this.jcbCategoryID;
30    }
31
32    //Getter method for jcbName
33    public JComboBox getJCBName(){
34          return this.jcbName;
35    }
36
37    //Getter method for jbEdit
38    public JButton getJBEdit(){
39          return this.jbEdit;
40    }
41
42    //Getter method for jbInsert
43    public JButton getJBInsert(){
44          return this.jbInsert;
45    }
46
47    //Getter method for jbDelete
```

```
48      public JButton getJBDelete(){
49          return this.jbDelete;
50      }
51
52      //Getter method for jbFirst
53      public JButton getJBFirst(){
54          return this.jbFirst;
55      }
56
57      //Getter method for jbPrev
58      public JButton getJBPrev(){
59          return this.jbPrev;
60      }
61
62      //Getter method for jbNext
63      public JButton getJBNext(){
64          return this.jbNext;
        }

        //Getter method for jbLast
        public JButton getJBLast(){
            return this.jbLast;
        }
```

POPULATING TABLE AND COMBOBOXES
POPULATING TABLE AND COMBOBOXES

Step 1

In **Category_Utils.java**, add two new methods: **get_category_list()** and **s**

The **get_category_list()** method takes in a **CategoryForm** object, an SQL It establishes a connection to the database, sets the item parameter in th "none"), executes the query, and stores the results in an **ArrayList** of **Cate**

The **show_table_category()** method takes in a **CategoryForm** object an input. It initializes a **DefaultTableModel** object with three columns ("Cat sets this model as the model for the **jtCategory** JTable object in the **Cat** list of **Category** objects, adding each row to the table model.

```
1       private static ArrayList<Category>
2   get_category_list(CategoryForm frm, String sql, String item){
3           ArrayList<Category> list = new ArrayList<>();
4           Connection conv = null;
5
6           try(Connection conn = getConnection();
7               PreparedStatement ps = conn.prepareStatement(sql)){
```

```
            if (item.equalsIgnoreCase("none")==false) {
                 ps.setString(1,item);
            }
            ResultSet rs = ps.executeQuery();

            Category obj;
            while(rs.next()){
                //Using three-params constructor
                obj = new Category(rs.getInt("category_id"),
                    rs.getString("name"),
                    rs.getTimestamp("last_update"));

                list.add(obj);
            }
        }catch (SQLException ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
                "ERROR",JOptionPane.ERROR_MESSAGE);
        }
        return list;
    }

    private static void show_table_category(CategoryForm frm,
ArrayList<Category> list) throws SQLException{
        DefaultTableModel model = new DefaultTableModel(0,0);

        String header[] = {"Category ID", "Name", "Last Update"};


model.setColumnIdentifiers(set_column_header(frm.getJTCategory(),
header));
        frm.getJTCategory().setModel(model);

        Object[] row = new Object[3];

        for(int i=0; i<list.size(); i++){
            row[0] = list.get(i).getCategoryID();
            row[1] = list.get(i).getName();
            row[2] = list.get(i).getLastUpdate();

            model.addRow(row);
        }
    }
```

Step 2

In **Category_Utils.java**, define **refresh_controls()** method. It takes in a **C**

This method first sets the location and title of the **CategoryForm** object tasks:

- Apply alternating row colors to the **jtCategory** JTable object **table_renderer()** method.

- Retrieve the list of all categories from the database using the **g** them in the table in the **CategoryForm** using the **show_table_cat**
- Populate the **jcbCategoryID** JComboBox object in the **CategoryF** sorted by category ID using the **populate_combobox()** method.
- Populate the **jcbName** JComboBox object in the **CategoryForm** sorted by name using the **populate_combobox()** method.

If any SQL exception occurs during the execution of any of these tasks, displayed using the **JOptionPane.showMessageDialog()** method.

```
    public static void
refresh_controls(CategoryForm frm){

frm.setLocationRelativeTo(null);
        frm.setTitle("CATEGORY
FORM");

        //Shows the content of
category table and populates
combobox
        try{
            //Makes alternating
color for table rows

table_renderer(frm.getJTCategory());

            //Populates table
            ArrayList<Category> list
= get_category_list(frm, "SELECT *
FROM category", "none");
            show_table_category(frm,
list);

            //Populates
jcbCategoryID
            String sql_id = "SELECT
category_id FROM category ORDER BY
category_id";

populate_combobox(sql_id,
frm.getJCBCategoryID(), frm);


            //Populates getJCBName()
            String sql_name =
 "SELECT DISTINCT name FROM category
ORDER BY name";


populate_combobox(sql_name,
frm.getJCBName(), frm);
```

```
        }catch (SQLException ex){

JOptionPane.showMessageDialog(frm,
ex.getMessage(),

 "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }
```

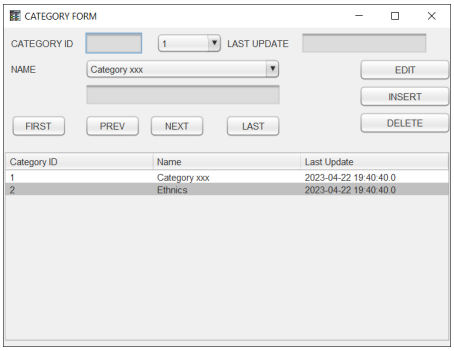


Figure 4.2 The content of **category** table displaye

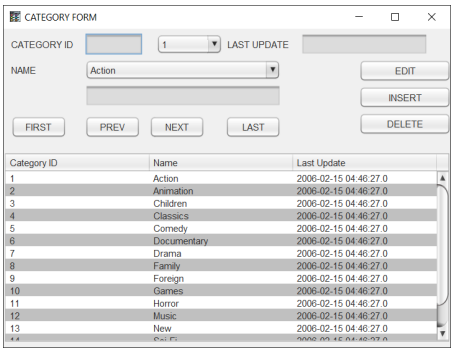


Figure 4.3 The the content of **category** table in original **Sakila** databas

jtCategory

Step 3

In **CategoryForm**'s default constructor, the **Category_Utils.refresh_** populates the controls in the **CategoryForm** with data from a database from the **Category_Utils** class.

The **this.setIconImage()** method sets the icon of the **Ca** **this.setDefaultCloseOperation(this.HIDE_ON_CLOSE)** method sets th to hide the form instead of exiting the application when the close button is

```
1  package sakila;
2  import java.awt.Toolkit;
3  import java.awt.event.ActionEvent;
4  import java.awt.event.ActionListener;
5  import java.io.IOException;
```

```
6  import java.text.ParseException;
7  import java.util.logging.Level;
8  import java.util.logging.Logger;
9  import javax.swing.JButton;
10 import javax.swing.JComboBox;
11 import javax.swing.JLabel;
12 import javax.swing.JTable;
13 import javax.swing.JTextField;
14
15 public class CategoryForm extends javax.swing.JFrame {
16     public CategoryForm() {
17         initComponents();
18         Utility.setLookAndFeel(this);
19         Category_Utils.refresh_controls(this);
20
21  this.setIconImage(Toolkit.getDefaultToolkit().getImage(getClass().
22 ;
23         this.setDefaultCloseOperation(this.HIDE_ON_CLOSE);
24     }
25     //...
26 }
```

Step 4	Run the project to see the content of **category** table displayed in **jtCatego** If you use the data from **Sakila** MySQL database available in the int **category** table displayed in **jtCategory** as shown in Figure 4.3.

DISPLAYING AND NAVIGATING DATA ROW BY ROW

DISPLAYING AND NAVIGATING DATA ROW BY ROW

Step 1	In **Category_Utils** class, define two new methods named **clear_co** **display_category_data()**. • **clear_controls(CategoryForm frm)**: This method takes a **Ca** object as input and clears the contents of the **jtfCategoryID**, **j** **jtfLastUpdate** components in the **CategoryForm** by setting their strings. • **display_category_data(CategoryForm frm, String sql, T** method takes a **CategoryForm** object, a SQL query string, an generic type T as input. It executes the SQL query with the provi parameter, retrieves the result set, and displays the retrieved cat the **CategoryForm**. Specifically, for each row in the result set, it of the **jtfCategoryID**, **jtfName**, and **jtfLastUpdate** compor

corresponding values retrieved from the database. It also selects item in the **jcbCategoryID** and **jcbName** comboboxes **find_combo_value_selected()** method. If the result set is empty **clear_controls()** method to clear the form fields.

If any SQL exception occurs during the execution of the query, a dialog error message is displayed using the **JOptionPane.showMessageDialog(**

```
1     private static void clear_controls(CategoryForm frm){
2         frm.getJTFCategoryID().setText("");
3         frm.getJTFName().setText("");
4         frm.getJTFLastUpdate().setText("");
5     }
6
7    //Displays category data result row by row
8     private static <T> void display_category_data(CategoryForm frm,
9 T item){
10        try(Connection conn = getConnection()){
11            PreparedStatement ps = conn.prepareStatement(sql);
12            ps.setObject(1,item);
13            ResultSet rs = ps.executeQuery();
14
15            if (!rs.next()) {
16                // no row found, clear the form fields
17                clear_controls(frm);
18                return;
19            }
20
21            do{
22
23 frm.getJTFCategoryID().setText(String.valueOf(rs.getInt("category_i
24                frm.getJTFName().setText(rs.getString("name"));
25
26 frm.getJTFLastUpdate().setText(String.valueOf(rs.getTimestamp("last
27
28                // Determines item selected from jcbCategoryID
29                find_combo_value_selected(frm.getJCBCategoryID(),
30 rs.getInt("category_id"));
31
32                // Determines item selected from jcbName
33                find_combo_value_selected(frm.getJCBName(),
34 rs.getString("name"));
35            }while(rs.next());
36
37            rs.close();
38            ps.close();
39        }catch(SQLException ex){
40            JOptionPane.showMessageDialog(frm, ex.getMessage(),
41                    "ERROR",JOptionPane.ERROR_MESSAGE);
42        }
      }
```

Step 2

In the same class, define another method named **jcbCategory_handler()** is a handler for the language selection event on two **JComboB CategoryForm**. The method retrieves the selected item from the **JCo** determines which query should be used to display the data in the form ba **JComboBox** was selected. It then calls the **display_category_data()** metl the data.

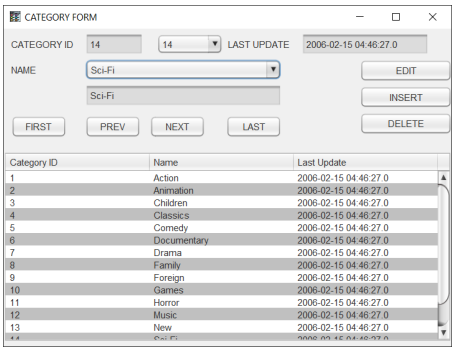
```
    public static void
jcbCategory_handler(CategoryForm frm,
JComboBox<String> jcb) {
        Object item =
jcb.getSelectedItem();
        String sql = "";
        if
(jcb.equals(frm.getJCBCategoryID()))
{
            sql =
Query_Category.get_sql_id();
        } else if
(jcb.equals(frm.getJCBName())) {
            sql =
Query_Category.get_sql_name();
        }

        display_category_data(frm,
sql, item);
    }
```

Step 3

In **CategoryForm**, double click on **jcbCategoryID** and **jcbName** com define their corresponding event handler as follows:

```
    private void
jcbCategoryIDActionPerformed(java.awt.event.ActionEvent
evt) {
        Category_Utils.jcbLanguage_handler(this,
this.jcbCategoryID);
    }

    private void
jcbNameActionPerformed(java.awt.event.ActionEvent evt)
{
        Category_Utils.jcbLanguage_handler(this,
this.jcbName);
    }
```

These two methods are event handlers for when the user selects an item ir two **JComboBoxes**, **jcbCategoryID** and **jcbName**. When an item is sele

	of the two **JComboBoxes**, the corresponding event handler meth **jcbCategory_handler()** method in the **Category_Utils** class, pas **CategoryForm** instance and the selected **JComboBox** as argu **jcbCategory_handler()** method then determines which query to use selected **JComboBox** and displays the data in the form **display_category_data()** method.
Step 4	Run the project. Choose one of items in **jcbCategoryID** and/or **jcbName** to see row by row the content of **category** table as shown in Figure 4.4.  **Figure 4.4** Displaying row by row the content of **category** tab
Step 5	Define four navigating methods in **Category_Utils** class. These are navigating the rows of the category form data. • **show_first_row(CategoryForm frm)** shows the first row of the by getting the first item from the **jcbCategoryID** JComboBox and the **display_category_data()** method. It also sets currentIndex to the first item. • **show_last_row(CategoryForm frm)** shows the last row of the by getting the last item from the **jcbCategoryID** JComboBox and the **display_category_data()** method. It also sets currentIndex to the last item. • **show_prev_row(CategoryForm frm)** shows the previous row of data by decrementing **currentIndex** and getting the item at the from the **jcbCategoryID** JComboBox. It then passes the **display_category_data()** method. If **currentIndex** is less than t it resets currentIndex to the first index. • **show_next_row(CategoryForm frm)** shows the next row of the by incrementing **currentIndex** and getting the item at the curre the **jcbCategoryID** JComboBox. It then passes the it **display_category_data()** method. If **currentIndex** is greater index, it resets currentIndex to the last index.

```
1     public static void show_first_row(CategoryForm frm){
2         String item =
3 String.valueOf(frm.getJCBCategoryID().getItemAt(FIRST_INDEX));
```

```
        display_category_data(frm, SQL_ID, item);
        currentIndex = FIRST_INDEX;
    }

    public static void show_last_row(CategoryForm frm){
        int endIndex = frm.getJCBCategoryID().getItemCount() -
1;
        String item =
String.valueOf(frm.getJCBCategoryID().getItemAt(endIndex));
        display_category_data(frm, SQL_ID, item);
        currentIndex = endIndex;
    }

    public static void show_prev_row(CategoryForm frm){
        currentIndex--;
        if(currentIndex < FIRST_INDEX){
            currentIndex = FIRST_INDEX;
            return;
        }
        String item =
String.valueOf(frm.getJCBCategoryID().getItemAt(currentIndex));
        display_category_data(frm, SQL_ID, item);
    }

    public static void show_next_row(CategoryForm frm){
        int endIndex = frm.getJCBCategoryID().getItemCount() -
1;
        currentIndex++;
        if(currentIndex > endIndex){
            currentIndex = endIndex;
            return;
        }
        String item =
String.valueOf(frm.getJCBCategoryID().getItemAt(currentIndex));
        display_category_data(frm, SQL_ID, item);
    }
```

Step 6

Then in **CategoryForm**, double click on each navigation buttons to corresponding event handler:

```
    private void
jbNextActionPerformed(java.awt.event.ActionEvent
evt) {
        Category_Utils.show_next_row(this);
    }

    private void
jbFirstActionPerformed(java.awt.event.ActionEvent
evt) {
       Category_Utils.show_first_row(this);

```

```
    }

    private void
jbPrevActionPerformed(java.awt.event.ActionEvent
evt) {
        Category_Utils.show_prev_row(this);
    }

    private void
jbLastActionPerformed(java.awt.event.ActionEvent
evt) {
       Category_Utils.show_last_row(this);
    }
```

These are the action listener methods for the "Next", "First", "Prev", and "
respectively. They call the corresponding methods in the **Category_Utils**
the next/first/previous/last row in the **category** table.

Step 7

Run the project. Click on one or more navigation buttons to see the resul
Figure 4.5.

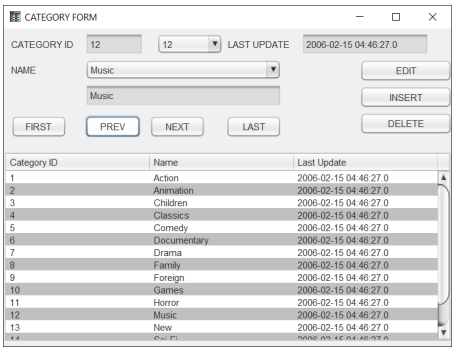


Figure 4.5 User clicks on one or more navigation buttons on catego

Step 8

Define **mouse_pressed_handler()** method in **Category_Utils** class. Th
used to handle the mouse click event on the **JTable** in the **CategoryI**
checks if a row has been selected, and displays an information message
selected. If a row is selected, it retrieves the category id of the selected r
calls the **display_category_data()** method to display the category data on

Inside the try block, it first gets a database connection, and then retrieves
id of the selected row using the **getValueAt()** method of the **JTable's** n
calls the **display_category_data()** method, passing in the id and the S
retrieve the category data from the database.

If an **SQLException** is caught, it logs the error and displays an error m
with the error message and stack trace.

```
    public static void mouse_pressed_handler(CategoryForm frm) {
        Objects.requireNonNull(frm, "frm must not be null");

        int selectedIndex = frm.getJTCategory().getSelectedRow();
        if (selectedIndex == -1) {
            JOptionPane.showMessageDialog(frm, "Please select a row to view its
data.",
                "No row selected", JOptionPane.INFORMATION_MESSAGE)
            return;
        }

        try (Connection conn = getConnection()) {
            String id =
String.valueOf(frm.getJTCategory().getModel().getValueAt(selectedIn
0));

            // Displays category data
            display_category_data(frm, Query_Category.get_sql_id(),
id);

        } catch (SQLException ex) {

Logger.getLogger(CategoryForm.class.getName()).log(Level.SEVERE, "E
displaying category data", ex);
            String message = "Error displaying category data: " +
ex.getMessage();
            String stackTrace = Arrays.toString(ex.getStackTrace())
            JOptionPane.showMessageDialog(frm, message + "\n\n" +
stackTrace, "ERROR", JOptionPane.ERROR_MESSAGE);
        }

    }
```

Step 9

Right click on **jtCategory**. Then, choose **Events > Mouse > mousePress** event handler:

```
    private void
jtCategoryMousePressed(java.awt.event.MouseEvent
evt) {

 Category_Utils.mouse_pressed_handler(this);
    }
```

Step 10

Run the project. Double click on any row in **jtCategory** table. You corresponding row in **category** table displayed in textfields and combobo in Figure 4.6.

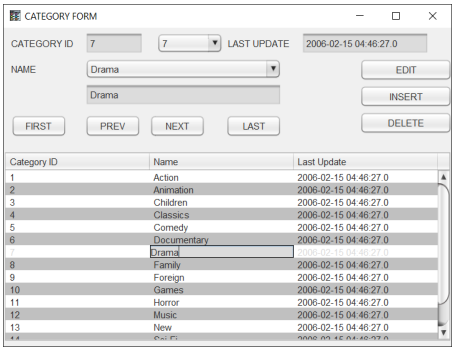


Figure 4.6 User double-clicks on any row in **jtCategory**

UPDATING RECORD
UPDATING RECORD

Step 1

In **Category_Utils** class, define a new method named **update_row_by_cat_id()**. It updates a row in **category** table by the **category_id**. The method takes in a **CategoryForm** object, an integer representing the **category_id**, and a **String** representing the name of the category to be updated.

The method first establishes a connection to the database using the **getConnection()** method. It then creates two **PreparedStatement** objects, one for selecting the row by **category_id** and another for updating the row with the new name. The **category_id** is used as a parameter for the SELECT query to find the row to update.

If the SELECT query does not return any results, the method displays an error message using **JOptionPane.showMessageDialog()**. If the SELECT query does return a result, the ResultSet is moved to the next row to get the **category_id**. A Category object is then created using the three-params constructor with the new name and the current timestamp.

The UPDATE query is then executed with the new name and the category_id as parameters. The **ResultSet**, **PreparedStatement**, and **Connection** objects are then closed in a try-with-resources block.

If an **SQLException** or **NumberFormatException** occurs, the method logs the error using **Logger** and displays an error message using **JOptionPane.showMessageDialog()**.

```
1       //Updates row of data in category tabel by category_id
```

```
    public static void update_row_by_cat_id(CategoryForm frm,
int cat_id, String name) throws SQLException{
        Connection conn = getConnection();
        ResultSet rs = null;
        String query_id = "SELECT category_id FROM category WHERE
category_id = ?";
        String update_query = """
            UPDATE category SET name = ? WHERE category_id =
?""";
        try(PreparedStatement idPs =
conn.prepareStatement(query_id,

ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_READ_ONLY);
        PreparedStatement updatePS =
conn.prepareStatement(update_query,
                ResultSet.TYPE_SCROLL_SENSITIVE,
ResultSet.CONCUR_UPDATABLE))
        {
            idPs.setInt(1,cat_id);
            if(!idPs.execute()){
                String message = "Can't find category_id " +
cat_id;

                JOptionPane.showMessageDialog(frm, message,
                    "ERROR",JOptionPane.ERROR_MESSAGE);
            } else{
                rs = idPs.getResultSet();
                rs.next();

                //Creates a Category object using three-params
constructor
                Category obj = new Category(cat_id, name, new
Timestamp(System.currentTimeMillis()));
                updatePS.setString(1, obj.getName());
                updatePS.setInt(2, obj.getCategoryID());

                updatePS.executeUpdate();
                rs.close();
                updatePS.close();
                idPs.close();
                conn.close();
            }
        }catch(SQLException ex){

Logger.getLogger(CategoryForm.class.getName()).log(Level.SEVERE,
"Error updating category data", ex);
            String message = "Error updating category data: " +
ex.getMessage();
            String stackTrace =
Arrays.toString(ex.getStackTrace());
            JOptionPane.showMessageDialog(null, message + "\n\n"
+ stackTrace, "ERROR", JOptionPane.ERROR_MESSAGE);
        }catch(java.lang.NumberFormatException ex){
```

```
Logger.getLogger(CategoryForm.class.getName()).log(Level.SEVERE,
"Invalid Input", ex);
            String message = "Invalid Input: " +
ex.getMessage();
            String stackTrace =
Arrays.toString(ex.getStackTrace());
            JOptionPane.showMessageDialog(null, message + "\n\n"
+ stackTrace, "ERROR", JOptionPane.ERROR_MESSAGE);
        }
    }
```

Step 2

Then in the same class, define a new method **read_inputs()**. It reads and validates user inputs from a **CategoryForm** object and returns a **HashMap** with the validated input data. The **CategoryForm** object is passed as a parameter to the method.

The method first creates a new **HashMap** object to store the input data. It then reads the **category_id** and name inputs from the **CategoryForm** object using the **getJCBCategoryID()** and **getJTFName()** methods, respectively.

The **category_id** input is validated by parsing it as an integer and checking if it is greater than zero. If it is not, an **IllegalArgumentException** is thrown with an error message, and an error dialog is displayed using **JOptionPane.showMessageDialog()**. If the **category_id** input is not a valid integer, a **NumberFormatException** is thrown, and an error dialog is displayed with an error message.

The name input is validated by checking if it is null or empty. If it is, an **IllegalArgumentException** is thrown with an error message, and an error dialog is displayed using **JOptionPane.showMessageDialog()**.

The validated **category_id** and **name** inputs are then put into the **HashMap** object and returned.

If an exception is thrown during the validation process, the exception is re-thrown to be handled by the calling method.

```
1  
2  
3  
4  
5  
6  
    private static HashMap<String, String>
read_inputs(CategoryForm frm) {
        HashMap<String, String> input_data = new
HashMap<>();
        String cat_id =
String.valueOf(frm.getJCBCategoryID().getSelectedItem());
```

```
        String name = frm.getJTFName().getText();

        // Validate user input
        int cat_id_int = 0;
        try {
            cat_id_int = Integer.parseInt(cat_id);
            if (cat_id_int <= 0) {
                throw new
IllegalArgumentException("Category ID cannot be negative
or zero");
            }
        } catch (NumberFormatException ex) {
            JOptionPane.showMessageDialog(frm, "Invalid
Category ID: " + cat_id,
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        } catch (IllegalArgumentException ex) {
            JOptionPane.showMessageDialog(frm,
ex.getMessage(),
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        }

        if (name == null || name.isEmpty()) {
            JOptionPane.showMessageDialog(frm, "Name
cannot be empty",
            "Error", JOptionPane.ERROR_MESSAGE);
            throw new IllegalArgumentException("Name
cannot be empty");
        }

        input_data.put("cat_id", cat_id);
        input_data.put("name", name);

        return input_data;
    }
```

Step 3

Still in the same class, define another method named **edit_actual()**. It handles the editing of a row in the **category** table in the database. The **CategoryForm** object is passed as a parameter to the method.

The method first calls the **read_inputs()** method to read and validate the user input data from the **CategoryForm** object. The validated **category_id** and name inputs are then extracted from the **HashMap** object returned by the **read_inputs()** method.

The **update_row_by_cat_id()** method is then called with the **CategoryForm** object, the validated **category_id** and **name** inputs as

parameters. This method updates the row in the **category** table in the database with the new data.

After the row is updated, the **refresh_controls()** method is called to refresh all objects on the **CategoryForm** object.

If a **SQLException** is caught during the updating process, an error dialog is displayed with the exception message.

```
    private static void
edit_actual(CategoryForm frm){
        try{
            HashMap<String, String>
input_data = read_inputs(frm);
            int cat_id =
Integer.parseInt(input_data.get("cat_id"));
            String name =
input_data.get("name");

            update_row_by_cat_id(frm,
cat_id, name);

            //Refreshes all objects on form
            refresh_controls(frm);

        }catch(SQLException ex){

JOptionPane.showMessageDialog(frm,
ex.getMessage(),

 "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }
```

Step 4

Lastly, define two new methods named **enable_controls()** and **edit_handler()**. It handles the editing of a row in the **category** table when the "EDIT" button is clicked on the **CategoryForm** object.

If the text of the "EDIT" button is "EDIT", the method sets the text of the button to "CONFIRM". It then calls the **enable_controls()** method with the parameter "false" to disable all controls on the **CategoryForm** object.

If the text of the "EDIT" button is "CONFIRM", the method sets the text of the button back to "EDIT". It then calls the **edit_actual()** method to perform the actual editing of the row in the **category** table. Finally, it calls the **enable_controls()** method with the parameter "true" to re-enable all controls on the **CategoryForm** object.

```
    private static void
enable_controls(boolean state, CategoryForm
frm){
        frm.getJBFirst().setEnabled(state);
        frm.getJBPrev().setEnabled(state);
        frm.getJBNext().setEnabled(state);
        frm.getJBLast().setEnabled(state);
        frm.getJBInsert().setEnabled(state);
        frm.getJBDelete().setEnabled(state);

frm.getJTFCategoryID().setEnabled(state);
    }

    public static void
edit_handler(CategoryForm frm){

 if(frm.getJBEdit().getText().equals("EDIT")
){

frm.getJBEdit().setText("CONFIRM");

            // Disables controls
            enable_controls(false, frm);
        }

        else {
            frm.getJBEdit().setText("EDIT");

            // Actual editing
            edit_actual(frm);

            //Enables controls
            enable_controls(true, frm);
        }
    }
```

Step 5	Run the project. Choose **category_id** using **jcbCategoryID** or **jcbName** combobox. Or, you can choose one of rows in **jtCategory** (in this case, **category_id = 13**). Then, click on EDIT button as shown in Figure 4.7. Edit the category name. Then, click on CONFIRM button. The edited row had been saved into **category** table as shown in Figure 4.8.

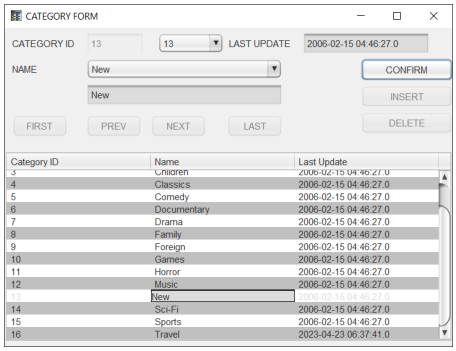


Figure 4.7 The category form is in editing state

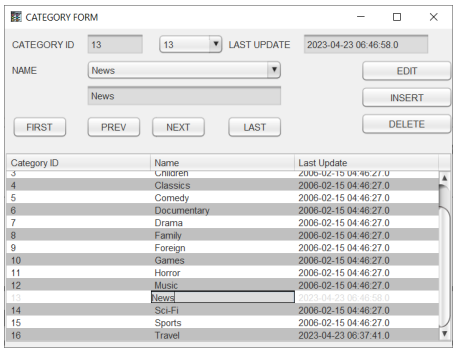


Figure 4.8 The edited row had been saved into database

UPDATING RECORD DIRECTLY ON JTABLE
UPDATING RECORD DIRECTLY ON JTABLE

Step 1

In **Category_Utility** class, define a new method **edit_database_from_jtable()**. This method handles updates to the component in the **CategoryForm**. It first checks if the event type is an UI indicating that a cell value has been changed. It then retrieves the updat from the model and uses it to update the corresponding row in the datal calling the **update_row_by_cat_id()** method.

If the update operation throws a **SQLIntegrityConstraintViolationExc SQLSyntaxErrorException**, or **SQLException**, it catches the excepti displays an error message to the user. Finally, it calls the **refresh_con** method to refresh all the objects on the form to display the updated data.

```
1     public static void edit_database_from_jtable(TableModelEvent
2 e, CategoryForm frm){
3         if (e.getType() == TableModelEvent.UPDATE) {
4             int row = e.getFirstRow();
5             TableModel model = (TableModel)e.getSource();
```

```
            int cat_id =
Integer.parseInt(String.valueOf(model.getValueAt(row, 0)));
            String name = (String) model.getValueAt(row, 1);

            try{
                update_row_by_cat_id(frm, cat_id, name);

                //Refreshes all objects on form
                refresh_controls(frm);

            } catch (SQLIntegrityConstraintViolationException
ex) {

Logger.getLogger(CategoryForm.class.getName()).log(Level.SEVERE,
"Duplicate entry", ex);
                JOptionPane.showMessageDialog(frm, "Error:
Duplicate entry\n" + ex.getMessage());
            } catch (SQLSyntaxErrorException ex) {

Logger.getLogger(CategoryForm.class.getName()).log(Level.SEVERE,
"Invalid SQL syntax", ex);
                JOptionPane.showMessageDialog(frm, "Error:
Invalid SQL syntax\n" + ex.getMessage());
            } catch (SQLException ex) {

Logger.getLogger(CategoryForm.class.getName()).log(Level.SEVERE,
"Database error", ex);
                JOptionPane.showMessageDialog(frm, "Error:
Database error\n" + ex.getMessage());
            }
        }
    }
```

Step 2

Create a new public class named **CategoryTableModelListener**. It impl the **TableModelListener** interface. The class has two instance varia **JTable** object named **jt** and a **CategoryForm** object named **frm**. The clas constructor that takes in a **JTable** object and a **CategoryForm** obje initializes the instance variables.

The **tableChanged()** method is an overridden method fror **TableModelListener** interface that gets called when there is a change table model. Inside this method, the **edit_database_from_jtable()** metho **Category_Utils** is called, passing in the **TableModelEvent** object a **CategoryForm** object. This method handles the editing of the database ba the changes made in the **JTable**.
Finally, the method checks if there is a cell editor currently active in the J and if so, it stops editing the cell by calling **stopCellEditing()** method.

```
package sakila;
```

```
import javax.swing.event.TableModelEvent;
import
javax.swing.event.TableModelListener;
import javax.swing.JTable;

public class CategoryTableModelListener
implements TableModelListener {
    private final JTable jt;
    private final CategoryForm frm;

    public
CategoryTableModelListener(JTable jt,
CategoryForm frm) {
        this.jt = jt;
        this.frm = frm;
    }

    @Override
    public void
tableChanged(TableModelEvent e) {

Category_Utils.edit_database_from_jtable(e,
frm);

        if (jt.getCellEditor() != null) {

jt.getCellEditor().stopCellEditing();
        }
    }
}
```

Step 3

Right click on **jtCategory**. Then, choose **Events > Mouse > mouseC** Define its event handler:

```
    private void jtCategoryMouseClicked(java.awt.event.MouseEvent ev
        // instantiate CategoryTableModelListener and add it as a li
to the table model
        CategoryTableModelListener tableModelListener = new
CategoryTableModelListener(this.getJTCategory(), this);

 this.getJTCategory().getModel().addTableModelListener(tableModelLis
;
    }
```

In this code, a mouse click event listener is attached to a **JTable** com called **jtCategory**. When the user clicks on the table, an insta **CategoryTableModelListener** is created and added as a listener to th model. This listener is responsible for handling any changes made to th data, such as updates made by the user.

	The **CategoryTableModelListener** constructor takes two argument **JTable** component that the listener is attached to and the **Categor** instance that contains the table. In the **tableChanged()** method of the li the **edit_database_from_jtable()** method from **Category_Utils** is calle the **TableModelEvent** object and the **CategoryForm** instance. This me responsible for updating the database with any changes made to the table After the **edit_database_from_jtable()** method is called, the **stopCellEd** method is called on any active cell editor for the table. This ensures th changes made by the user are committed before the listener updat database.
Step 4	Run the project. Click on any cell in second column in any row in **jtCatego** you want to edit. Then, change it. Then, click any where outsi corresponding cell. The edited data had been saved into database.

INSERTING NEW RECORD
INSERTING NEW RECORD

Step 1	In **Category_Utils** class, define a method named **insert_row()**. This method inserts a new row into the **category** table in the database. It first reads the input data from the form using the **read_inputs()** method, which returns a **HashMap** containing the values of the **name** field. It then constructs an SQL insert statement using a multi-line string literal (introduced in Java 15), which inserts a new row with the specified **name** value into the **category** table. Inside a try-with-resources block, the method establishes a database connection using the **getConnection()** method, prepares a statement using the constructed SQL insert statement, sets the **name** parameter of the statement using the **obj.getName()** method, creates a new **Category** object using a two-parameter constructor with the **name** and current timestamp values, and executes the insert statement using the **executeUpdate()** method. If a **SQLException** is caught during the execution of the method, it is logged using the **Logger** class and a message box is displayed to the user with an error message. 1 `//Inserts new row into category table` 2 `private static void insert_row(CategoryForm frm) throws` 3 `SQLException{`

```
        HashMap<String, String> input_data = read_inputs(frm);
        String name = input_data.get("name");

        // SQL insert statement
        String sql = """
            INSERT INTO category(name) VALUES(?)""";

        try(Connection conn = getConnection();
         PreparedStatement pstmt = conn.prepareStatement(sql)){

            //Creates a Category object two-params constructor
            Category obj = new Category(name, new
Timestamp(System.currentTimeMillis()));
            pstmt.setString(1,obj.getName());

            //Executes the sql insert statement
            pstmt.executeUpdate();
        } catch (SQLException ex) {

Logger.getLogger(CategoryForm.class.getName()).log(Level.SEVERE,
"Database error", ex);
            JOptionPane.showMessageDialog(frm, "Error: Database
error\n" + ex.getMessage());
        }
    }
```

Step 2

Still in **Category_Utils.java**, define **insert_actual()** and **insert_handler()** methods. The **insert_handler()** method is responsible for handling the "INSERT" button on the **CategoryForm**. When the button is clicked, the method checks the text of the button. If the text is "INSERT", it changes the text to "CONFIRM" and disables the **jbEdit** button, all the controls on the form, and the **jcbCategoryID** and **jcbName** combo boxes. It then clears the controls and enables the "CONFIRM" button.

If the text of the "INSERT" button is "CONFIRM", the method changes the text of the button back to "INSERT" and calls the **insert_actual()** method to actually insert the data into the database. It then enables the **jbEdit** button and re-enables all the controls on the form, as well as the **jcbCategoryID** and **jcbName** combo boxes.

```
    private static void
insert_actual(CategoryForm frm){
        try{
            insert_row(frm);

            //Refreshes table and comboboxes
            refresh_controls(frm);

```

```
        }catch(SQLException ex){
            JOptionPane.showMessageDialog(frm,
ex.getMessage(),

 "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    public static void
insert_handler(CategoryForm frm){

 if(frm.getJBInsert().getText().equals("INSERT")
){

frm.getJBInsert().setText("CONFIRM");

            //Disables jbEdit
            frm.getJBEdit().setEnabled(false);

            // Disables controls
            enable_controls(false, frm);

frm.getJCBCategoryID().setEnabled(false);
            frm.getJCBName().setEnabled(false);

            // Clears controls
            clear_controls(frm);

            // Enables
            frm.getJBInsert().setEnabled(true);
        }

        else {
            frm.getJBInsert().setText("INSERT");

            // Actual insertion
            insert_actual(frm);


```

```
            //Enables jbEdit

frm.getJBEdit().setEnabled(true);

            //Enables controls
            enable_controls(true, frm);

frm.getJCBCategoryID().setEnabled(true);

frm.getJCBName().setEnabled(true);
        }
    }
```

Step 3

In **CategoryForm.java**, double clic
to create its event listener:

```
    private void
jbInsertActionPerformed(java.aw
evt) {
        Category_Utils.insert_h
    }
```

Step 4

Run the project. Click on INSERT
the state of category form when ins
as shown in Figure 4.9.

Then, type a category name. Th
button to save the new record int
shown in Figure 4.10.

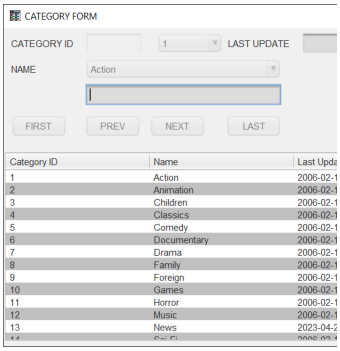


Figure 4.9 When user clicks on I
category form will be in stat

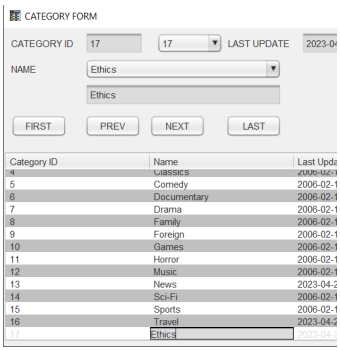


Figure 4.10 The new data had
category table

DELETING RECORD
DELETING RECORD

Step 1

Then in **Category_Utils** class, define **delete_handler()** method. This handles the deletion of a row in the **category** table. It first prompts th confirm the deletion and then proceeds to delete the row if the user conf category ID of the row to be deleted is obtained from the category ID com the **CategoryForm**. The SQL query to delete the row is constructed as a statement to avoid SQL injection attacks. The category ID is then se parameter value for the prepared statement and executed. If the exe successful, the controls on the form are refreshed using the **refresh_c** method. If there is a database error, an error message dialog is displayed.

```
1     public static void delete_handler(CategoryForm frm){
2         int dialogButton = JOptionPane.YES_NO_OPTION;
3         int cat_id =
4 Integer.parseInt(String.valueOf(frm.getJCBCategoryID().getSelectedI
5
6         String message = String.format("Are you sure you want to de
7 row Category ID: %d)", cat_id);
8         int answer = JOptionPane.showConfirmDialog(frm, message, "D
9 ROW OF DATA", dialogButton);
10
11        if(answer == JOptionPane.YES_OPTION){
12            String query = """
13                DELETE FROM category WHERE category_id = ?""";
14            try(Connection conn = getConnection();
15                PreparedStatement ps = conn.prepareStatement(query)
16                // Use PreparedStatement to avoid SQL injection att
17                ps.setInt(1, cat_id);
18                ps.executeUpdate();
19
20                // Refresh table and comboboxes
```

```
                refresh_controls(frm);

            } catch (SQLException ex){
                JOptionPane.showMessageDialog(frm, ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
            }
        }
    }
```

Step 2

In **CategoryForm.java**, double click on DELETE button to generate listener:

```
    private void
jbDeleteActionPerformed(java.awt.event.ActionEvent
evt) {
        Category_Utils.delete_handler(this);
    }
```

Step 3

Run the project. Choose **category_id** using **jcbCategoryID** or j combobox. Then, Click on DELETE button. The corresponding row of data deleted from database.

This is the full version of **Category_Utils.java**:

```
package sakila;
import java.util.logging.Level;
import java.util.logging.Logger;
import java.sql.*;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashMap;
import java.util.Objects;
import javax.swing.JComboBox;
import javax.swing.JOptionPane;
import javax.swing.event.TableModelEvent;
import javax.swing.table.DefaultTableModel;
import javax.swing.table.TableModel;

public class Category_Utils extends Utility{
    public static final int FIRST_INDEX = 0;
    public static final int INVALID_INDEX = -1;

    private static int currentIndex = FIRST_INDEX;
    private static final String SQL_ID = Query_Category.get_sql_id();
```

```
    //Creates category table
    public static void create_category_table() {
        try (Connection conn = getConnection()) {
            Statement stmt = conn.createStatement();
            stmt.addBatch(Query_Category.get_sql_category());
            stmt.executeBatch();

            String message = String.format("Successfully creates category
table");
            JOptionPane.showMessageDialog(null, message,
            "INFORMATION",JOptionPane.INFORMATION_MESSAGE);

        } catch (SQLException ex) {
            JOptionPane.showMessageDialog(null, ex.getMessage(),
                "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    //Populates category table with some rows of data
    public static void populate_category_table(){
        try(Connection conn = getConnection()){
            String sql = """
                INSERT INTO category(category_id, name, last_update)
                VALUES(?, ?, ?)""";

            //Creates a new Category class with default constructor
            PreparedStatement ps1 = conn.prepareStatement(sql);
            Category obj1 = new Category();
            ps1.setInt(1,obj1.getCategoryID());
            ps1.setString(2,obj1.getName());
            ps1.setTimestamp(3,obj1.getLastUpdate());

            // Creates a new Category class with four-params constructor
            PreparedStatement ps2 = conn.prepareStatement(sql);
            Category obj2 = new Category(2, "Ethnics", new
Timestamp(System.currentTimeMillis()));
            ps2.setInt(1,obj2.getCategoryID());
            ps2.setString(2,obj2.getName());
            ps2.setTimestamp(3,obj2.getLastUpdate());

            ps1.executeUpdate();
            ps2.executeUpdate();

        }catch(SQLException ex){
            JOptionPane.showMessageDialog(null, ex.getMessage(),
                "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    //Reads the content of category table
    public static void read_category_table(){
        try(Connection conn = getConnection()){
            Statement stmt = conn.createStatement();
```

```
            ResultSet rs = stmt.executeQuery("SELECT * FROM category");

            while(rs.next()){
                int cat_id = rs.getInt("category_id");
                String name = rs.getString("name");
                Timestamp lu = rs.getTimestamp("last_update");

                //Creates a Category object using three-params constructor
                Category obj = new Category(cat_id, name, lu);
                System.out.println(obj);
            }
            rs.close();
            stmt.close();

        }catch(SQLException ex){
            JOptionPane.showMessageDialog(null, ex.getMessage(),
                "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    private static ArrayList<Category> get_category_list(CategoryForm frm,
String sql, String item){
        ArrayList<Category> list = new ArrayList<>();
        Connection conv = null;

        try(Connection conn = getConnection();
            PreparedStatement ps = conn.prepareStatement(sql)){
            if (item.equalsIgnoreCase("none")==false) {
                 ps.setString(1,item);
            }
            ResultSet rs = ps.executeQuery();

            Category obj;
            while(rs.next()){
                //Using three-params constructor
                obj = new Category(rs.getInt("category_id"),
                    rs.getString("name"),
                    rs.getTimestamp("last_update"));

                list.add(obj);
            }
        }catch (SQLException ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
                "ERROR",JOptionPane.ERROR_MESSAGE);
        }
        return list;
    }

    private static void show_table_category(CategoryForm frm,
ArrayList<Category> list) throws SQLException{
        DefaultTableModel model = new DefaultTableModel(0,0);

        String header[] = {"Category ID", "Name", "Last Update"};
```

```
        model.setColumnIdentifiers(set_column_header(frm.getJTCategory(),
header));
        frm.getJTCategory().setModel(model);

        Object[] row = new Object[3];

        for(int i=0; i<list.size(); i++){
            row[0] = list.get(i).getCategoryID();
            row[1] = list.get(i).getName();
            row[2] = list.get(i).getLastUpdate();

            model.addRow(row);
        }
    }

    public static void refresh_controls(CategoryForm frm){
        frm.setLocationRelativeTo(null);
        frm.setTitle("CATEGORY FORM");

        //Shows the content of category table and populates combobox
        try{
            //Makes alternating color for table rows
            table_renderer(frm.getJTCategory());

            //Populates table
            ArrayList<Category> list = get_category_list(frm, "SELECT * FROM
category", "none");
            show_table_category(frm, list);

            //Populates jcbCategoryID
            String sql_id = "SELECT category_id FROM category ORDER BY
category_id";
            populate_combobox(sql_id, frm.getJCBCategoryID(), frm);


            //Populates getJCBName()
            String sql_name = "SELECT DISTINCT name FROM category ORDER BY
name";

            populate_combobox(sql_name, frm.getJCBName(), frm);

        }catch (SQLException ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
            "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    private static void clear_controls(CategoryForm frm){
        frm.getJTFCategoryID().setText("");
        frm.getJTFName().setText("");
        frm.getJTFLastUpdate().setText("");
    }
```

```
   //Displays category data result row by row
    private static <T> void display_category_data(CategoryForm frm, String sc
T item){
        try(Connection conn = getConnection()){
            PreparedStatement ps = conn.prepareStatement(sql);
            ps.setObject(1,item);
            ResultSet rs = ps.executeQuery();

            if (!rs.next()) {
                // no row found, clear the form fields
                clear_controls(frm);
                return;
            }

            do{

frm.getJTFCategoryID().setText(String.valueOf(rs.getInt("category_id")));
                frm.getJTFName().setText(rs.getString("name"));

frm.getJTFLastUpdate().setText(String.valueOf(rs.getTimestamp("last_update"))

                // Determines item selected from jcbCategoryID
                find_combo_value_selected(frm.getJCBCategoryID(),
rs.getInt("category_id"));

                // Determines item selected from jcbName
                find_combo_value_selected(frm.getJCBName(),
rs.getString("name"));
            }while(rs.next());

            rs.close();
            ps.close();
        }catch(SQLException ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    public static void jcbCategory_handler(CategoryForm frm, JComboBox<String
jcb) {
        Object item = jcb.getSelectedItem();
        String sql = "";
        if (jcb.equals(frm.getJCBCategoryID())) {
            sql = Query_Category.get_sql_id();
        } else if (jcb.equals(frm.getJCBName())) {
            sql = Query_Category.get_sql_name();
        }

        display_category_data(frm, sql, item);
    }

    public static void show_first_row(CategoryForm frm){
```

```
        String item =
String.valueOf(frm.getJCBCategoryID().getItemAt(FIRST_INDEX));
        display_category_data(frm, SQL_ID, item);
        currentIndex = FIRST_INDEX;
    }

    public static void show_last_row(CategoryForm frm){
        int endIndex = frm.getJCBCategoryID().getItemCount() - 1;
        String item =
String.valueOf(frm.getJCBCategoryID().getItemAt(endIndex));
        display_category_data(frm, SQL_ID, item);
        currentIndex = endIndex;
    }

    public static void show_prev_row(CategoryForm frm){
        currentIndex--;
        if(currentIndex < FIRST_INDEX){
            currentIndex = FIRST_INDEX;
            return;
        }
        String item =
String.valueOf(frm.getJCBCategoryID().getItemAt(currentIndex));
        display_category_data(frm, SQL_ID, item);
    }

    public static void show_next_row(CategoryForm frm){
        int endIndex = frm.getJCBCategoryID().getItemCount() - 1;
        currentIndex++;
        if(currentIndex > endIndex){
            currentIndex = endIndex;
            return;
        }
        String item =
String.valueOf(frm.getJCBCategoryID().getItemAt(currentIndex));
        display_category_data(frm, SQL_ID, item);
    }

    public static void mouse_pressed_handler(CategoryForm frm) {
        Objects.requireNonNull(frm, "frm must not be null");

        int selectedIndex = frm.getJTCategory().getSelectedRow();
        if (selectedIndex == -1) {
            JOptionPane.showMessageDialog(frm, "Please select a row to view 
data.",
                "No row selected", JOptionPane.INFORMATION_MESSAGE);
            return;
        }

        try (Connection conn = getConnection()) {
            String id =
String.valueOf(frm.getJTCategory().getModel().getValueAt(selectedIndex, 0));

            // Displays category data
```

```
            display_category_data(frm, Query_Category.get_sql_id(), id);

        } catch (SQLException ex) {
            Logger.getLogger(CategoryForm.class.getName()).log(Level.SEVERE,
"Error displaying category data", ex);
            String message = "Error displaying category data: " +
ex.getMessage();
            String stackTrace = Arrays.toString(ex.getStackTrace());
            JOptionPane.showMessageDialog(frm, message + "\n\n" + stackTrace,
"ERROR", JOptionPane.ERROR_MESSAGE);
        }

    }

    //Updates row of data in category tabel by category_id
    public static void update_row_by_cat_id(CategoryForm frm, int cat_id,
String name) throws SQLException{
        Connection conn = getConnection();
        ResultSet rs = null;
        String query_id = "SELECT category_id FROM category WHERE category_id
?";
        String update_query = """
            UPDATE category SET name = ? WHERE category_id = ?""";
        try(PreparedStatement idPs = conn.prepareStatement(query_id,

ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_READ_ONLY);
        PreparedStatement updatePS = conn.prepareStatement(update_query,
                ResultSet.TYPE_SCROLL_SENSITIVE, ResultSet.CONCUR_UPDATABLE))
        {
            idPs.setInt(1,cat_id);
            if(!idPs.execute()){
                String message = "Can't find category_id " + cat_id;

                JOptionPane.showMessageDialog(frm, message,
                    "ERROR",JOptionPane.ERROR_MESSAGE);
            } else{
                rs = idPs.getResultSet();
                rs.next();

                //Creates a Category object using three-params constructor
                Category obj = new Category(cat_id, name, new
Timestamp(System.currentTimeMillis()));
                updatePS.setString(1, obj.getName());
                updatePS.setInt(2, obj.getCategoryID());

                updatePS.executeUpdate();
                rs.close();
                updatePS.close();
                idPs.close();
                conn.close();
            }
        }catch(SQLException ex){
```

```
            Logger.getLogger(CategoryForm.class.getName()).log(Level.SEVERE,
"Error updating category data", ex);
            String message = "Error updating category data: " +
ex.getMessage();
            String stackTrace = Arrays.toString(ex.getStackTrace());
            JOptionPane.showMessageDialog(null, message + "\n\n" + stackTrace
"ERROR", JOptionPane.ERROR_MESSAGE);
        }catch(java.lang.NumberFormatException ex){
            Logger.getLogger(CategoryForm.class.getName()).log(Level.SEVERE,
"Invalid Input", ex);
            String message = "Invalid Input: " + ex.getMessage();
            String stackTrace = Arrays.toString(ex.getStackTrace());
            JOptionPane.showMessageDialog(null, message + "\n\n" + stackTrace
"ERROR", JOptionPane.ERROR_MESSAGE);
        }
    }

    private static HashMap<String, String> read_inputs(CategoryForm frm) {
        HashMap<String, String> input_data = new HashMap<>();
        String cat_id =
String.valueOf(frm.getJCBCategoryID().getSelectedItem());
        String name = frm.getJTFName().getText();

        // Validate user input
        int cat_id_int = 0;
        try {
            cat_id_int = Integer.parseInt(cat_id);
            if (cat_id_int <= 0) {
                throw new IllegalArgumentException("Category ID cannot be
negative or zero");
            }
        } catch (NumberFormatException ex) {
            JOptionPane.showMessageDialog(frm, "Invalid Category ID: " +
cat_id,
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        } catch (IllegalArgumentException ex) {
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        }

        if (name == null || name.isEmpty()) {
            JOptionPane.showMessageDialog(frm, "Name cannot be empty",
            "Error", JOptionPane.ERROR_MESSAGE);
            throw new IllegalArgumentException("Name cannot be empty");
        }

        input_data.put("cat_id", cat_id);
        input_data.put("name", name);

        return input_data;
    }
```

```
    private static void edit_actual(CategoryForm frm){
        try{
            HashMap<String, String> input_data = read_inputs(frm);
            int cat_id = Integer.parseInt(input_data.get("cat_id"));
            String name = input_data.get("name");

            update_row_by_cat_id(frm, cat_id, name);

            //Refreshes all objects on form
            refresh_controls(frm);

        }catch(SQLException ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    private static void enable_controls(boolean state, CategoryForm frm){
        frm.getJBFirst().setEnabled(state);
        frm.getJBPrev().setEnabled(state);
        frm.getJBNext().setEnabled(state);
        frm.getJBLast().setEnabled(state);
        frm.getJBInsert().setEnabled(state);
        frm.getJBDelete().setEnabled(state);
        frm.getJTFCategoryID().setEnabled(state);
    }

    public static void edit_handler(CategoryForm frm){
        if(frm.getJBEdit().getText().equals("EDIT")){
            frm.getJBEdit().setText("CONFIRM");

            // Disables controls
            enable_controls(false, frm);
        }

        else {
            frm.getJBEdit().setText("EDIT");

            // Actual editing
            edit_actual(frm);

            //Enables controls
            enable_controls(true, frm);
        }
    }

    public static void edit_database_from_jtable(TableModelEvent e,
CategoryForm frm){
        if (e.getType() == TableModelEvent.UPDATE) {
            int row = e.getFirstRow();
            TableModel model = (TableModel)e.getSource();
```

```
            int cat_id = Integer.parseInt(String.valueOf(model.getValueAt(row
0)));
            String name = (String) model.getValueAt(row, 1);

            try{
                update_row_by_cat_id(frm, cat_id, name);

                //Refreshes all objects on form
                refresh_controls(frm);

            } catch (SQLIntegrityConstraintViolationException ex) {

Logger.getLogger(CategoryForm.class.getName()).log(Level.SEVERE, "Duplicate
entry", ex);
                JOptionPane.showMessageDialog(frm, "Error: Duplicate entry\n'
ex.getMessage());
            } catch (SQLSyntaxErrorException ex) {

Logger.getLogger(CategoryForm.class.getName()).log(Level.SEVERE, "Invalid SQL
syntax", ex);
                JOptionPane.showMessageDialog(frm, "Error: Invalid SQL
syntax\n" + ex.getMessage());
            } catch (SQLException ex) {

Logger.getLogger(CategoryForm.class.getName()).log(Level.SEVERE, "Database
error", ex);
                JOptionPane.showMessageDialog(frm, "Error: Database error\n"
ex.getMessage());
            }
        }
    }

    //Inserts new row into category table
    private static void insert_row(CategoryForm frm) throws SQLException{
        HashMap<String, String> input_data = read_inputs(frm);
        String name = input_data.get("name");

        // SQL insert statement
        String sql = """
            INSERT INTO category(name) VALUES(?)""";

        try(Connection conn = getConnection();
         PreparedStatement pstmt = conn.prepareStatement(sql)){

            //Creates a Category object two-params constructor
            Category obj = new Category(name, new
Timestamp(System.currentTimeMillis()));
            pstmt.setString(1,obj.getName());

            //Executes the sql insert statement
            pstmt.executeUpdate();
        } catch (SQLException ex) {
            Logger.getLogger(CategoryForm.class.getName()).log(Level.SEVERE,
"Database error", ex);
```

```
            JOptionPane.showMessageDialog(frm, "Error: Database error\n" +
ex.getMessage());
        }
    }

    private static void insert_actual(CategoryForm frm){
        try{
            insert_row(frm);

            //Refreshes table and comboboxes
            refresh_controls(frm);

        }catch(SQLException ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    public static void insert_handler(CategoryForm frm){
        if(frm.getJBInsert().getText().equals("INSERT")){
            frm.getJBInsert().setText("CONFIRM");

            //Disables jbEdit
            frm.getJBEdit().setEnabled(false);

            // Disables controls
            enable_controls(false, frm);
            frm.getJCBCategoryID().setEnabled(false);
            frm.getJCBName().setEnabled(false);

            // Clears controls
            clear_controls(frm);

            // Enables
            frm.getJBInsert().setEnabled(true);
        }

        else {
            frm.getJBInsert().setText("INSERT");

            // Actual insertion
            insert_actual(frm);

            //Enables jbEdit
            frm.getJBEdit().setEnabled(true);

            //Enables controls
            enable_controls(true, frm);
            frm.getJCBCategoryID().setEnabled(true);
            frm.getJCBName().setEnabled(true);
        }
    }
```

```
    public static void delete_handler(CategoryForm frm){
        int dialogButton = JOptionPane.YES_NO_OPTION;
        int cat_id =
Integer.parseInt(String.valueOf(frm.getJCBCategoryID().getSelectedItem()));

        String message = String.format("Are you sure you want to delete the r
Category ID: %d)", cat_id);
        int answer = JOptionPane.showConfirmDialog(frm, message, "DELETING RO
OF DATA", dialogButton);

        if(answer == JOptionPane.YES_OPTION){
            String query = """
                DELETE FROM category WHERE category_id = ?""";
            try(Connection conn = getConnection();
                PreparedStatement ps = conn.prepareStatement(query)){
                // Use PreparedStatement to avoid SQL injection attacks
                ps.setInt(1, cat_id);
                ps.executeUpdate();

                // Refresh table and comboboxes
                refresh_controls(frm);

            } catch (SQLException ex){
                JOptionPane.showMessageDialog(frm, ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
            }
        }
    }
}
```

This is the full version of **CategoryForm.java**:

```
package sakila;
import java.awt.Toolkit;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.IOException;
import java.text.ParseException;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.JButton;
import javax.swing.JComboBox;
import javax.swing.JLabel;
import javax.swing.JTable;
import javax.swing.JTextField;

public class CategoryForm extends javax.swing.JFrame {
    public CategoryForm() {
        initComponents();
        Utility.setLookAndFeel(this);
        Category_Utils.refresh_controls(this);
```

```
        this.setIconImage(Toolkit.getDefaultToolkit().getImage(getClass().getResource("/ima
        this.setDefaultCloseOperation(this.HIDE_ON_CLOSE);
    }

    //Getter method for jtfCategoryID
    public JTextField getJTFCategoryID(){
        return this.jtfCategoryID;
    }

    //Getter method for jtfLastUpdate
    public JTextField getJTFLastUpdate(){
        return this.jtfLastUpdate;
    }

    //Getter method for jtfName
    public JTextField getJTFName(){
        return this.jtfName;
    }

    //Getter method for jtCategory
    public JTable getJTCategory(){
        return this.jtCategory;
    }

    //Getter method for jcbCategoryID
    public JComboBox getJCBCategoryID(){
        return this.jcbCategoryID;
    }

    //Getter method for jcbName
    public JComboBox getJCBName(){
        return this.jcbName;
    }

    //Getter method for jbEdit
    public JButton getJBEdit(){
        return this.jbEdit;
    }

    //Getter method for jbInsert
    public JButton getJBInsert(){
        return this.jbInsert;
    }

    //Getter method for jbDelete
    public JButton getJBDelete(){
        return this.jbDelete;
    }

    //Getter method for jbFirst
    public JButton getJBFirst(){
        return this.jbFirst;
    }
```

```java
    //Getter method for jbPrev
    public JButton getJBPrev(){
        return this.jbPrev;
    }

    //Getter method for jbNext
    public JButton getJBNext(){
        return this.jbNext;
    }

    //Getter method for jbLast
    public JButton getJBLast(){
        return this.jbLast;
    }

    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">
    private void initComponents() {
        //...
        pack();
    }// </editor-fold>

    private void jcbCategoryIDActionPerformed(java.awt.event.ActionEvent evt)
        Category_Utils.jcbCategory_handler(this, this.jcbCategoryID);
    }

    private void jcbNameActionPerformed(java.awt.event.ActionEvent evt) {
        Category_Utils.jcbCategory_handler(this, this.jcbName);
    }

    private void jbEditActionPerformed(java.awt.event.ActionEvent evt) {
        Category_Utils.edit_handler(this);
    }

    private void jbInsertActionPerformed(java.awt.event.ActionEvent evt) {
        Category_Utils.insert_handler(this);
    }

    private void jbDeleteActionPerformed(java.awt.event.ActionEvent evt) {
        Category_Utils.delete_handler(this);
    }

    private void jbLastActionPerformed(java.awt.event.ActionEvent evt) {
        Category_Utils.show_last_row(this);
    }

    private void jbNextActionPerformed(java.awt.event.ActionEvent evt) {
        Category_Utils.show_next_row(this);
    }

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

```
        Category_Utils.show_prev_row(this);
    }

    private void jbFirstActionPerformed(java.awt.event.ActionEvent evt) {
        Category_Utils.show_first_row(this);
    }

    private void jtCategoryMousePressed(java.awt.event.MouseEvent evt) {
        Category_Utils.mouse_pressed_handler(this);
    }

    private void jtCategoryMouseClicked(java.awt.event.MouseEvent evt) {
        // instantiate CategoryTableModelListener and add it as a listener to
        CategoryTableModelListener tableModelListener = new
CategoryTableModelListener(this.getJTCategory(), this);
        this.getJTCategory().getModel().addTableModelListener(tableModelListe
    }

    /**
     * @param args the command line arguments
     */
    public static void main(String args[]) {
        try {
            for (javax.swing.UIManager.LookAndFeelInfo info :
javax.swing.UIManager.getInstalledLookAndFeels()) {
                if ("Nimbus".equals(info.getName())) {
                    javax.swing.UIManager.setLookAndFeel(info.getClassName())
                    break;
                }
            }
        } catch (ClassNotFoundException ex) {

java.util.logging.Logger.getLogger(CategoryForm.class.getName()).log(java.ut
null, ex);
        } catch (InstantiationException ex) {

java.util.logging.Logger.getLogger(CategoryForm.class.getName()).log(java.ut
null, ex);
        } catch (IllegalAccessException ex) {

java.util.logging.Logger.getLogger(CategoryForm.class.getName()).log(java.ut
null, ex);
        } catch (javax.swing.UnsupportedLookAndFeelException ex) {

java.util.logging.Logger.getLogger(CategoryForm.class.getName()).log(java.ut
null, ex);
        }
        /* Create and display the form */
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new CategoryForm().setVisible(true);
            }
        });
```

```
    }

    // Variables declaration - do not modify
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JLabel jLabel4;
    private javax.swing.JScrollPane jScrollPane1;
    private javax.swing.JButton jbDelete;
    private javax.swing.JButton jbEdit;
    private javax.swing.JButton jbFirst;
    private javax.swing.JButton jbInsert;
    private javax.swing.JButton jbLast;
    private javax.swing.JButton jbNext;
    private javax.swing.JButton jbPrev;
    private javax.swing.JComboBox<String> jcbCategoryID;
    private javax.swing.JComboBox<String> jcbName;
    private javax.swing.JTable jtCategory;
    private javax.swing.JTextField jtfCategoryID;
    private javax.swing.JTextField jtfLastUpdate;
    private javax.swing.JTextField jtfName;
    // End of variables declaration
}
```

FILM FORM FILM FORM

CREATING AND POPULATING FILM TABLE

CREATING AND POPULATING FILM TABLE

Step 1	Create a new class named **Query_Film**. It contains several SQL queries as static strings, along with getter methods to access those queries. Here's a brief explanation of each query: • **sql_min**: A query that retrieves the minimum value of **film_id** from the **film** table. • **sql_max**: A query that retrieves the maximum value of **film_id** from the **film** table. • **sql_id**: A parameterized query that retrieves a row from the **film** table based on the **film_id** parameter. • **sql_title**: A parameterized query that retrieves a row from the **film** table based on the **title** parameter. • **sql_film_duration_dist**: A query that retrieves the distribution of film rental durations, along with the number of films having each duration value. • **sql_film_lang_dist**: A query that retrieves the distribution of films by language, along with the number of films in each language.

- **sql_film_rating_dist**: A query that retrieves the distribution of films by rating, along with the number of films having each rating value.
- **sql_film_year_dist**: A query that retrieves the distribution of films by release year, along with the number of films released in each year. The results are ordered by the number of films in descending order and limited to the top 10 years.
- **sql_film_joint**: A query that retrieves all columns from the **film** table, along with the **name** of the language for each film (obtained by joining the **film** and **language** tables).
- **sql_film**: A query that creates a new **film** table with various columns and constraints.

```
1  package sakila;
2
3  public class Query_Film {
4      private static final String
5  sql_min = "SELECT MIN(film_id) FROM
6  film";
7      private static final String
8  sql_max = "SELECT MAX(film_id) FROM
9  film";
10     private static final String
11 sql_id = "SELECT * FROM film WHERE
12 film_id  = ?";
13     private static final String
14 sql_title = "SELECT * FROM film
15 WHERE title = ?";
16
17     private static final String
18 sql_film_duration_dist = """
19         SELECT f.rental_duration AS
20 duration, COUNT(*) AS Number
21         FROM film f
22         JOIN language l1 ON
23 l1.language_id = f.language_id
24         GROUP BY duration
25         ORDER BY Count(*) DESC,
26 duration ASC;""";
27
28
29
```

```
    private static final String
sql_film_lang_dist = """
        SELECT l1.name, COUNT(*) AS
Number
        FROM film f
        JOIN language l1 ON
l1.language_id = f.language_id
        GROUP BY l1.name
        ORDER BY Count(*) DESC,
l1.name ASC;""";

    private static final String
sql_film_rating_dist = """
        SELECT rating, COUNT(*) AS
Number
        FROM film
        GROUP BY rating
        ORDER BY Count(*) DESC,
rating ASC""";

    private static final String
sql_film_year_dist = """
        SELECT YEAR(release_year)
AS year, Count(*) AS Number
        FROM film
        GROUP BY year
        ORDER BY Count(*) DESC,
year ASC
        LIMIT 10""";

    private static final String
sql_film_joint = """
        SELECT f.film_id, f.title,
f.description, f.release_year,
            f.language_id,
f.original_language_id,
f.rental_duration,
            f.rental_rate,
f.length, f.replacement_cost,
f.rating,
```

```
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
```

```
            f.special_features,
f.last_update,
            l1.name AS language_name
        FROM film f
        JOIN language l1 ON
l1.language_id = f.language_id;""";

    private static final String
sql_film = """
        CREATE TABLE film (
          film_id SMALLINT UNSIGNED NOT
NULL AUTO_INCREMENT,
          title VARCHAR(128) NOT NULL,
          description TEXT DEFAULT
NULL,
          release_year YEAR DEFAULT
NULL,
          language_id TINYINT UNSIGNED
NOT NULL,
          original_language_id TINYINT
UNSIGNED DEFAULT NULL,
          rental_duration TINYINT
UNSIGNED NOT NULL DEFAULT 3,
          rental_rate DECIMAL(4,2) NOT
NULL DEFAULT 4.99,
          length SMALLINT UNSIGNED
DEFAULT NULL,
          replacement_cost DECIMAL(5,2)
NOT NULL DEFAULT 19.99,
          rating ENUM('G','PG','PG-
13','R','NC-17') DEFAULT 'G',
          special_features
SET('Trailers','Commentaries','Deleted
Scenes','Behind the Scenes') DEFAULT
NULL,
          last_update TIMESTAMP NOT
NULL DEFAULT CURRENT_TIMESTAMP ON
UPDATE CURRENT_TIMESTAMP,
          PRIMARY KEY  (film_id),
          KEY idx_title (title),
          KEY idx_fk_language_id
(language_id),
          KEY
idx_fk_original_language_id
(original_language_id),
          CONSTRAINT fk_film_language
FOREIGN KEY (language_id) REFERENCES
language (language_id) ON DELETE
RESTRICT ON UPDATE CASCADE,
          CONSTRAINT
fk_film_language_original FOREIGN KEY
(original_language_id) REFERENCES
```

```
language (language_id) ON DELETE
RESTRICT ON UPDATE CASCADE
        ) ENGINE=InnoDB DEFAULT
CHARSET=utf8mb4;""";

    //Getter methods
    public static String get_sql_min()
{
        return sql_min;
    }

    public static String get_sql_max()
{
        return sql_max;
    }

    public static String get_sql_id() {
        return sql_id;
    }

    public static String
get_sql_title() {
        return sql_title;
    }

    public static String get_sql_film()
{
        return sql_film;
    }

    public static String
get_sql_film_joint() {
        return sql_film_joint;
    }

    public static String
get_sql_film_year_dist() {
        return sql_film_year_dist;
    }

    public static String
get_sql_film_rating_dist() {
        return sql_film_rating_dist;
    }

    public static String
get_sql_film_lang_dist() {
        return sql_film_lang_dist;
    }

    public static String
get_sql_film_duration_dist() {
        return sql_film_duration_dist;
```

```
    }
}
```

Step 2

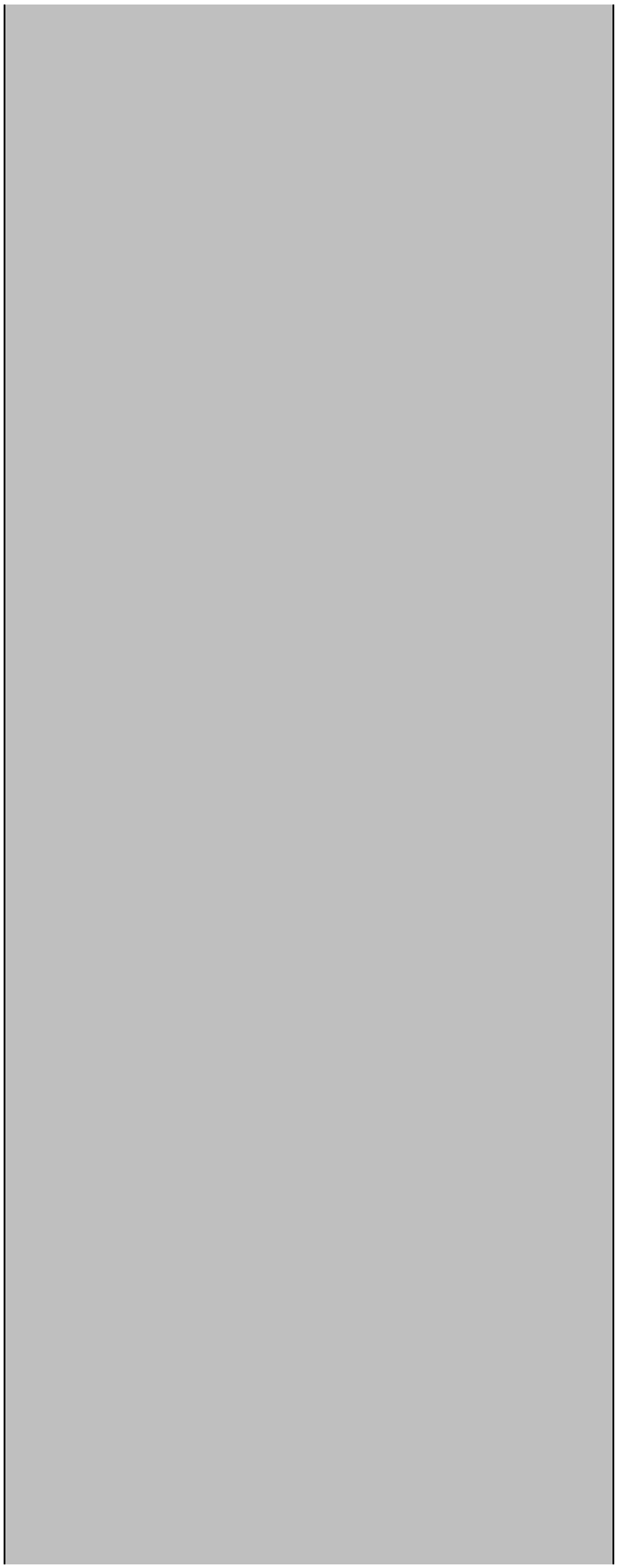
Then, create a public class named
variables and getter/setter metho
several constructors with different p
well as default values for some varia
is in the sakila package.

The instance variables are:

- **film_id**: an integer represen
- **title**: a string representing t
- **description**: a string rep
description
- **release_year**: an integer re
the film was released
- **language_id**: an integer re
the language the film is in
- **original_language_id**: an
the ID of the original langua
- **rental_duration**: an integ
rental duration of the film (d
- **rental_rate**: a double rep
rate of the film (default is 4.9
- **length**: an integer represen
film
- **replacement_cost**: a doub
replacement cost of the film
- **rating**: a string representi
film (default is "G")
- **special_features**: a string
special features of the film
- **last_update**: a **Timestamp**
the last time the film was up
- **lang_name**: a string repre
the language the film is in

Here's an explanation of each sette
class:

1. **setFilmID(int id)** - Sets the
If the id provided is less tha
IllegalArgumentException
2. **setTitle(String title)** - Sets
If the title provided is
IllegalArgumentException



length of the title is greater
an **IllegalArgumentExcept**

3. **setDescription(String des** description for the film.
4. **setReleaseYear(int year)** - for the film. The year provi year between 1900 and otherwise an **IllegalArgu** thrown.
5. **setLanguageID(int id)** - S for the film. If the id prov equal to zero, an **IllegalArg** thrown.
6. **setOriginalLanguageID(In** original language ID for provided is less than or eq null, an **IllegalArgumentEx**
7. **setRentalDuration(int du** rental duration for the fi provided is less than or equa rental duration of 3 is used.
8. **setRentalRate(double rat** rate for the film. If the rate or equal to zero, the default used.
9. **setLength(Integer length)** the film. If the length prov equal to zero and **IllegalArgumentException**
10. **setReplacementCost(doub** replacement cost for the film is less than or equal to replacement cost of 19.99 is
11. **setRating(String rating)** the film.
12. **setSpecialFeatures(String** special features for the film.
13. **setLastUpdate(Timestamp** update timestamp for the film
14. **setName(String name)** - name for the film.

The purpose of this code is to defi can be used to represent films application. The different construct in how the Film object can be cre depending on the needs of the appli

```
1  package sakila;
2  import java.util.Objects;
3  import java.sql.Timestamp;
4  import java.time.Year;
5  import java.util.Arrays;
6  import java.util.HashSet;
7  import java.util.List;
8  import java.util.Set;
9
10 public class Film {
11     //DEFAULT VALUES
12     private final int RENTA
13     private final double RE
14     private final double RE
15 19.99;
16     private final String RA
17
18     //14 Instance Variables
19     private int film_id;
20     private String title;
21     private String descript
22     private int release_yea
23     private int language_id
24     private int original_la
25     private int rental_dura
26     private double rental_r
27     private int length;
28     private double replacem
29     private String rating;
30     private String special_
31     private Timestamp last_
32     private String lang_nam
33
34     //Default constructor
35     Film(){
36         this(1, "NO TITLE",
37 3,
38             4.99, 100, 19.9
39             new
40 Timestamp(System.currentTim
41 "English");
42     }
43
44     //Twelve-params constru
45     Film(String title, Stri
46 year, int lang_id, int ori_
47             int duration, d
48 int length, double cost, St
49             String features
50         setTitle(title);
51         setDescription(desc
52         setReleaseYear(year
```

```
53          setLanguageID(lang_
54          setOriginalLanguage
55          setRentalDuration(d
56          setRentalRate(renta
57          setLength(length);
58          setReplacementCost(
59          setRating(rating);
60          setSpecialFeatures(
61          setLastUpdate(lu);
62      }
63
64      //Thirteen-params const
65      Film(int film_id, Strin
66  description, int year, int
67  ori_lang_id,
68              int duration, d
69  int length, double cost, St
70              String features
71          this(title, descrip
72  lang_id, ori_lang_id,
73              duration, renta
74  cost, rating,
75              features, lu);
76          setFilmID(film_id);
77      }
78
79      //Fourteen-params const
80      Film(int film_id, Strin
81  description, int year, int
82  ori_lang_id,
83              int duration, d
84  int length, double cost, St
85              String features
86  String name){
87          this(film_id, title
88  year, lang_id, ori_lang_id,
89              duration, renta
90  cost, rating,
91              features, lu);
92          setName(name);
93      }
94
95      // Getter methods
96      public int getFilmID()
97      public String getTitle(
98      public String getDescri
99  description;}
100     public int getReleaseYe
101 release_year;}
102     public int getLanguageI
103 language_id;}
104     public int getOriginalL
105 original_language_id;}
106
```

```
107     public int getRentalDu
108 rental_duration;}
109     public double getRental
110 rental_rate;}
111     public int getLength()
112     public double getReplac
113 replacement_cost;}
114     public String getRating
115     public String getSpecia
116 special_features;}
117     public Timestamp getLas
118 last_update;}
119     public String getName()
120 lang_name;}
121 
122     //Setter methods
123     public void setFilmID(i
124         if (id <= 0) {
125             throw new
126127 IllegalArgumentException("F
128 greater than zero.");
129         }
130         this.film_id = id;
131     }
132 
133     public void setTitle(St
134         if (title == null |
135 title.trim().isEmpty()) {
136             throw new
137 IllegalArgumentException("T
138 or empty");
139         }
140         if (title.length()
141             throw new
142 IllegalArgumentException("T
143 longer than 128 characters"
144         }
144         this.title = title;
145     }
146 
147     public void setDescript
148 description) {
149         this.description =
150     }
151 
152     public void setReleaseY
153         Year releaseYear =
154         if
155 (releaseYear.isBefore(Year.
156 releaseYear.isAfter(Year.no
157             throw new
158 IllegalArgumentException("I
159 year.");
160 
```

```
        }
        this.release_year =
releaseYear.getValue();
    }

    public void setLanguage
        if (id <= 0) {
            throw new
IllegalArgumentException("l
greater than zero.");
        }
        this.language_id =
    }

    public void setOriginal
id) {
        if (id != null && i
            throw new
IllegalArgumentException("O
must be greater than zero."
        }
        this.original_langu
    }
    public void setRentalDu
duration) {
        if (duration <= 0)
            this.rental_dur
RENTAL_DURATION;
        }
        this.rental_duratio
    }

    public void setRentalRa
        if (rate <= 0) {
            this.rental_rat
        }
        this.rental_rate =
    }

    public void setLength(
        if (length != null
            throw new
IllegalArgumentException("l
greater than zero.");
        }
        this.length = lengt
    }

    public void setReplacem
cost) {
        if (cost <= 0) {
            this.replacemen
REPLACEMENT_COST;
```

```
215         }
216         this.replacement_co
217     }
218
219     public void setRating(S
220         List<String> validR
221 Arrays.asList("G", "PG", "R
222 17");
223         if
224 (!validRatings.contains(rat
225226 {
227             this.rating = R
228         }
229         this.rating = ratin
230     }
231
232     public void setSpecialF
233 features) {
234         this.special_featur
235     }
236
237     public void setLastUpda
238         if (date == null) {
239         throw new
240 IllegalArgumentException("D
241 null");
242         }
243         this.last_update =
244     }
245
246     public void setName(Str
247         if (name == null ||
248 name.trim().isEmpty()) {
249             throw new
250 IllegalArgumentException("L
251 be null or empty");
252         }
253         if (name.length() >
254             throw new
255 IllegalArgumentException("L
256 be longer than 45 character
         }
         this.lang_name = na
     }

     // Override the hashCod
     @Override
     public int hashCode() {
         return Objects.hash
 description, release_year,
                 language_id
 original_language_id, renta
```

```
                rental_rate
replacement_cost, rating,
                special_fea
lang_name);
    }

    // Override the equals(
    @Override
    public boolean equals((
        if (this == o) retu
        if (o == null || ge
o.getClass()) return false;
        Film fm = (Film) o;
        return film_id == f
                language_id
&&
                original_lan
fm.original_language_id &&
                Objects.equa
&&
                Objects.equa
fm.description) &&
                Objects.equa
fm.release_year) &&
                Objects.equa
fm.rental_duration) &&
                Objects.equa
fm.rental_rate) &&
                Objects.equa
fm.length) &&

Objects.equals(replacement_
fm.replacement_cost) &&
                Objects.equa
fm.rating) &&

Objects.equals(special_feat
fm.special_features) &&
                Objects.equa
fm.lang_name) &&
                Objects.equa
fm.last_update);
    }

    @Override
    public String toString(
        return "\nFilm ID
this.getFilmID() +
                "\nTitle
this.getTitle() +
                "\nDescript
this.getDescription() +
```

```
                "\nRelease
this.getReleaseYear() +
                "\nLanguage
this.getLanguageID() +
                "\nLanguage
this.getName() +
                "\nOriginal
this.getOriginalLanguageID(
                "\nRental D
this.getRentalDuration()+
                "\nRental R
this.getRentalRate()+
                "\nLength
this.getLength()+
                "\nReplacem
this.getReplacementCost()+
                "\nRating
this.getRating()+
                "\nSpecial
this.getSpecialFeatures()+
                "\nLast Upd
this.getLastUpdate();
    }
}
```

Step 3

Create a new public class named **Fi** methods for creating, populating, a in a database. Here is a brief method:

- **create_film_table()**: This film table in the database us It gets a connection to the **getConnection()** method fr **Utility**. It then executes a create the table and displa box to indicate if the table is or if there is an error.
- **populate_film_table()**: Th some rows of data into th database using SQL stat connection to the dat **getConnection()** method fr **Utility**. It then creates two objects with SQL statement the table. The first object in default constructor of the second object inserts data

parameter constructor of th
the values of each para
methods of the **Film** class.
the **executeUpdate()** meth
statement object to insert th

- **read_film_table()**: This
content of the film table in
SQL statements. It gets a
database using the **getCo**
from the parent class **Utili**
Statement object to execut
select all rows from the
ResultSet object to retriev
table and creates a **Film** ob
parameter constructor for
sets the values of each p
ResultSet object's getXXX
XXX is the data type of the
prints the **Film** object to the

```
1  package sakila;
2  import java.util.logging.Level
3  import java.util.logging.Logge
4  import java.sql.*;
5  import java.util.logging.Level
6  import java.util.logging.Logge
7  import java.sql.*;
8  import java.util.ArrayList;
9  import java.util.Arrays;
10 import java.util.HashMap;
11 import java.util.Objects;
12 import javax.swing.JComboBox;
13 import javax.swing.JOptionPane
14 import javax.swing.event.Table
15 import javax.swing.table.Defau
16 import javax.swing.table.Table
17
18 public class Film_Utils extend
19     public static final int FI
20     public static final int IN
21
22     private static int current
23 FIRST_INDEX;
24     private static final Strin
25 Query_Film.get_sql_id();
26
27     //Creates film table
28     public static void create_
29         try (Connection conn =
30 {
31
```

```
32              Statement stmt =
33  conn.createStatement();
34
35  stmt.addBatch(Query_Film.get_s
36              stmt.executeBatch(
37
38              String message =
39  String.format("Successfully cr
40  table");
41              JOptionPane.showMe
42  message,
43
44   "INFORMATION",JOptionPane.INF
45  ;
46
47          } catch (SQLException
48              JOptionPane.showMe
49  ex.getMessage(),
50
51   "ERROR",JOptionPane.ERROR_MES
52          }
53      }
54
55      //Populates film table wit
56  data
57      public static void populat
58          try(Connection conn =
59              String sql = """
60                      INSERT INTO
61  title, description, release_ye
62  original_language_id, rental_d
63  rental_rate, length,
64                              repla
65  rating, special_features, last
66                      VALUES(?, ?,
67  ?, ?, ?, ?, ?, ?)""";
68
69              PreparedStatement
70  conn.prepareStatement(sql);
71              //Creates a new Fi
72  default constructor
73              Film obj1 = new Fi
74              ps1.setInt(1,obj1.
75              ps1.setString(2, o
76              ps1.setString(3,
77  obj1.getDescription());
78              ps1.setInt(4,
79  obj1.getReleaseYear());
80              ps1.setInt(5,
81  obj1.getLanguageID());
82              ps1.setInt(6,
83  obj1.getOriginalLanguageID());
84
85
```

```
            ps1.setInt(7,
obj1.getRentalDuration());
            ps1.setDouble(8,
obj1.getRentalRate());
            ps1.setInt(9, obj1
            ps1.setDouble(10,
obj1.getReplacementCost());
            ps1.setString(11,
obj1.getRating());
            ps1.setString(12,
obj1.getSpecialFeatures());
            ps1.setTimestamp(1
obj1.getLastUpdate());

            PreparedStatement
conn.prepareStatement(sql);
            //Creates a new Fi
fourteen-params constructor
            Film obj2 = new Fi
Ethnic Movie", null, 2023, 1,
            2.99, 150, 13.99,
"Trailers,Commentaries",
            new
Timestamp(System.currentTimeMi
"English");
            ps2.setInt(1,obj2.
            ps2.setString(2, o
            ps2.setString(3,
obj2.getDescription());
            ps2.setInt(4,
obj2.getReleaseYear());
            ps2.setInt(5,
obj2.getLanguageID());
            ps2.setInt(6,
obj2.getOriginalLanguageID());
            ps2.setInt(7,
obj2.getRentalDuration());
            ps2.setDouble(8,
obj2.getRentalRate());
            ps2.setInt(9, obj2
            ps2.setDouble(10,
obj2.getReplacementCost());
            ps2.setString(11,
obj2.getRating());
            ps2.setString(12,
obj2.getSpecialFeatures());
            ps2.setTimestamp(1
obj2.getLastUpdate());

            ps1.executeUpdate(
            ps2.executeUpdate(

        }catch(SQLException ex
```

```
            JOptionPane.showMe
ex.getMessage(),

 "ERROR",JOptionPane.ERROR_MES
        }
    }

    //Reads the content of fil
    public static void read_fi
        try(Connection conn =
            Statement stmt =
conn.createStatement();
            String sql = "SELE
            ResultSet rs =
stmt.executeQuery(sql);
            while(rs.next()){
                int id = rs.ge
                String title =
rs.getString("title");
                String descrip
rs.getString("description");
                int year =
rs.getInt("release_year");
                int lang_id =
rs.getInt("language_id");
                int ori_lang_i
rs.getInt("original_language_i
                int duration =
rs.getInt("rental_duration");
                double rate =
rs.getDouble("rental_rate");
                int length =
rs.getInt("length");
                double cost =
rs.getDouble("replacement_cost
                String rating
rs.getString("rating");
                String feature
rs.getString("special_features
                Timestamp lu =
rs.getTimestamp("last_update")

                //Creates an F
thirteen-params constructor
                Film obj = new
description, year, lang_id, or
duration, rate, length, cost,
lu);
                System.out.pri
            }
            rs.close();
            stmt.close();
```

```
        }catch(SQLException ex
            JOptionPane.showMe
ex.getMessage(),

 "ERROR",JOptionPane.ERROR_MES
        }
    }
}
```

Step 4

In the driver class, **Sa**
create_film_table(), **populate_**
read_film_table() as shown in line 2

```
1  package sakila;
2
3  public class Sakila {
4      public static void main(Str
5  args) {
6  //        Utility.testConnectic
7  //
8  Actor_Utils.create_actor_table(
9  //
10 Actor_Utils.populate_actor_tabl
11 //        Actor_Utils.read_acto
12 //        ActorForm frm = new
13 ActorForm();
14 //        frm.setVisible(true);
15
16 //
17 Language_Utils.create_language_
18 //
19 Language_Utils.populate_languag
20 //
21 Language_Utils.read_language_ta
22 //        LanguageForm frm = ne
23 LanguageForm();
24 //        frm.setVisible(true);
25
26 //
27 Category_Utils.create_category_
28 //
   Category_Utils.populate_categor
   //
   Category_Utils.read_category_ta
   //        CategoryForm frm = ne
   CategoryForm();
   //        frm.setVisible(true);

           Film_Utils.create_film_
           Film_Utils.populate_fil
```

```
            Film_Utils.read_film_ta
        }
    }
```

Run project to see the result in cons

```
Film ID               : 1
Title                 : NO TITLE
Description           : null
Release Year          : 2023
Language ID           : 1
Language Name         : null
Original Language ID  : 1
Rental Duration       : 3
Rental Rate           : 4.99
Length                : 100
Replacement Cose      : 19.99
Rating                : G
Special Features      : null
Last Update           : 2023-04-23 1

Film ID               : 2
Title                 : Batak Ethnic
Description           : null
Release Year          : 2023
Language ID           : 1
Language Name         : null
Original Language ID  : 1
Rental Duration       : 3
Rental Rate           : 2.99
Length                : 150
Replacement Cose      : 13.99
Rating                : G
Special Features      : Trailers,Com
Last Update           : 2023-04-23 1
```

DESIGNING GUI
DESIGNING GUI

Step 1	In the project, create a new **JFrame Form** and name it as **FilmForm.java**. In the Design tab, add thirteen **JLabel** to the form and set their corresponding **text** properties as FILM ID, TITLE, DESCRIPTION, RELEASE YEAR, RENTAL RATE, RENTAL DURATION, LENGTH, REPLACEMENT COST, RATING, LANGUAGE,

	ORIGINAL LANGUAGE, SPECIAL FEATURES, and LAST UPDATE.
Step 2	Then, add twelve **JTextField** to the form and set set their corresponding **Variable Name** as **jtfFilmID**, **jtfTitle**, **jtfDescription**, **jtfReleaseYear**, **jtfRentalRate**, **jtfLength**, **jtfRentalDuration**, **jtfReplacementCost**, **jtfLanguageID**, **jtfOriginalLanguageID**, **jtfSpecialFeatures**, and **jtfLastUpdate**.
Step 3	Then, add eight **JButton** to the form and set their corresponding **Variable Name** as **jbFirst**, **jbPrev**, **jbNext**, **jbLast**, **jbEdit**, **jbInsert**, **jbDelete**, and **jbChart**. Set their corresponding **text** properties as FIRST, PREV, NEXT, LAST, EDIT, INSERT, DELETE, and CHART.
Step 4	Then, add seven **JComboBox** to the form and set set their corresponding **Variable Name** as **jcbFilmID**, **jcbTitle**, **jcbRating**, **jcbReleaseYear**, **jcbLanguage**, **jcbOriginalLanguage**, and **jcbSpecialFeatures**.
Step 5	Lastly, add a new **JTable** to the form set set its **Variable Name** as **jtFilm**. Then, right-click on it, then choose **Table Contents...** and set the number of columns to 14 and the number of rows to 50.
Step 6	In the driver class, **Sakila.java**, create a new object of **FilmForm** class using its default constructor as shown in 27 – 28:

```
1  package sakila;
2
3  public class Sakila {
4      public static void main(String[]
5  args) {
6  //         Utility.testConnection();
7  //
8  Actor_Utils.create_actor_table();
9  //
10 Actor_Utils.populate_actor_table();
11 //         Actor_Utils.read_actor_table();
12
```

```
13  //        ActorForm frm = new
14  ActorForm();
15  //        frm.setVisible(true);
16
17  //
18  Language_Utils.create_language_table();
19  //
20  Language_Utils.populate_language_table();
21  //
22  Language_Utils.read_language_table();
23  //        LanguageForm frm = new
24  LanguageForm();
25  //        frm.setVisible(true);
26
27  //
28  Category_Utils.create_category_table();
29  //
30  Category_Utils.populate_category_table();
    //
    Category_Utils.read_category_table();
    //        CategoryForm frm = new
    CategoryForm();
    //        frm.setVisible(true);

    //        Film_Utils.create_film_table();
    //
    Film_Utils.populate_film_table();
    //        Film_Utils.read_film_table();
            FilmForm frm = new FilmForm();
            frm.setVisible(true);
        }
    }
```

Step 8

In **FilmForm**'s constructor, invoke **setLookAndFeel()** to set the look and feel of the form as shown in line 17.

```
1
2
3
4
5
6
7
8
9
10
11
```

```
12  package sakila;
13
14  import java.awt.Toolkit;
15  import java.awt.event.ActionEvent;
16  import
17  java.awt.event.ActionListener;
18  import javax.swing.JButton;
19  import javax.swing.JComboBox;
20  import javax.swing.JMenuItem;
    import javax.swing.JOptionPane;
    import javax.swing.JPopupMenu;
    import javax.swing.JTable;
    import javax.swing.JTextField;

    public class FilmForm extends
    javax.swing.JFrame {
        public FilmForm() {
            initComponents();

    Utility.setLookAndFeel(this);
        }
        //...
    }
```

Run the project to see the film form as shown in Figure 5.1.

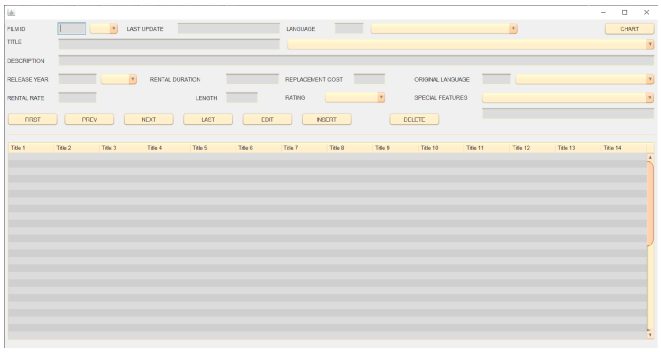


Figure 5.1 The layout of film form

Step 9

In **FilmForm.java**, define **ge** method for every object in the for place them outside and beneath its c constructor:

```
1      //Getter methods for JText
2  variable instances
3      public JTextField
4  getJTFFilmID(){return
5  this.jtfFilmID;}
```

```
    public JTextField getJTFTi
{return this.jtfTitle;}
    public JTextField
getJTFDescription(){return
this.jtfDescription;}
    public JTextField
getJTFLastUpdate(){return
this.jtfLastUpdate;}
    public JTextField
getJTFLanguageID(){return
this.jtfLanguageID;}
    public JTextField
getJTFReleaseYear(){return
this.jtfReleaseYear;}
    public JTextField
getJTFRentalDuration(){return
this.jtfRentalDuration;}
    public JTextField
getJTFOriginalLanguageID(){ret
this.jtfOriginalLanguageID;}
    public JTextField
getJTFRentalRate(){return
this.jtfRentalRate;}
    public JTextField
getJTFLength(){return
this.jtfLength;}
    public JTextField
getJTFReplacementCost(){return
this.jtfReplacementCost;}
    public JTextField
getJTFSpecialFeatures(){return
this.jtfSpecialFeatures;}

    //Getter methods for JComb
variable instances
    public JComboBox getJCBFil
{return this.jcbFilmID;}
    public JComboBox
getJCBLanguage(){return
this.jcbLanguage;}
    public JComboBox getJCBTit
{return this.jcbTitle;}
    public JComboBox
getJCBReleaseYear(){return
this.jcbReleaseYear;}
    public JComboBox
getJCBOriginalLanguage(){retur
this.jcbOriginalLanguage;}
    public JComboBox getJCBRat
{return this.jcbRating;}
    public JComboBox
getJCBSpecialFeatures(){return
this.jcbSpecialFeatures;}
```

```
    //Getter methods for JTabl
variable instance
    public JTable getJTFilm()
{return this.jtFilm;}

    //Getter methods for JButt
variable instances
    public JButton getJBEdit()
{return this.jbEdit;}
    public JButton getJBInsert
{return this.jbInsert;}
    public JButton getJBDelete
{return this.jbDelete;}
    public JButton getJBChart(
{return this.jbChart;}
    public JButton getJBFirst(
{return this.jbFirst;}
    public JButton getJBPrev()
{return this.jbPrev;}
    public JButton getJBNext()
{return this.jbNext;}
    public JButton getJBLast()
{return this.jbLast;}
```

POPULATING TABLE AND COMBOBOXES
POPULATING TABLE AND COMBOBOXES

Step 1	In **Film_Utils.java**, add two new methods: **get_film_list()** and **show_tab** on a **FilmForm** object and an **ArrayList<Film>**. The **get_film_list()** method takes three arguments: • **frm**: a **FilmForm** object • **sql**: a SQL query string • **item**: a string representing a search item The method returns an **ArrayList<Film>** object that contains **Film** obj data from a **ResultSet** obtained from executing a SQL query specified by t The method first creates a new **ArrayList** object to hold the **Film** objec database connection using the **getConnection()** method and creates a **Pr Connection** object, using the sql query string. If the item argument is n parameter of the **PreparedStatement** object to the value of item.

The method then executes the SQL query and iterates over the **ResultS** each row and adding it to the **ArrayList**.

If an **SQLException** occurs, the method displays an **JOptionPane.showMessageDialog()** method and returns an empty **Arra**

The **show_table_film()** method takes two arguments:

- **frm**: a **FilmForm** object
- **list**: an **ArrayList<Film>** object

The method displays the **Film** data in a **JTable** contained within the **F** **DefaultTableModel** object and sets its column identifiers to the result o method with the **JTextField** object contained within the **FilmForm** object column headers.

It then sets the **JTable's** model to the new **DefaultTableModel**. **ArrayList<Film>** object, creating an **Object** array of length 14 to hold each element of the array to the corresponding data value of the **Film** obje

Finally, it adds the **Object** array as a new row to the **DefaultTableModel**.

```
1     private static ArrayList<Film> get_film_list(FilmForm
2  frm, String sql, String item){
3          ArrayList<Film> list = new ArrayList<>();
4          Connection conv = null;
5
6          try(Connection conn = getConnection();
7              PreparedStatement ps =
8  conn.prepareStatement(sql)){
9              if (item.equalsIgnoreCase("none")==false) {
10                  ps.setString(1,item);
11             }
12             ResultSet rs = ps.executeQuery();
13
14             Film obj;
15             while(rs.next()){
16                 //Using fourteen-params constructor
17                 obj = new Film(rs.getInt("film_id"),
18 rs.getString("title"),
19                     rs.getString("description"),
20 rs.getInt("release_year"),
21                     rs.getInt("language_id"),
22 rs.getInt("original_language_id"),
23                     rs.getInt("rental_duration"),
24 rs.getDouble("rental_rate"),
25                     rs.getInt("length"),
26 rs.getDouble("replacement_cost"),
27                     rs.getString("rating"),
28 rs.getString("special_features"),
29                     rs.getTimestamp("last_update"),
30                     rs.getString("language_name"));
```

```
31
32                  list.add(obj);
33              }
34          }catch (SQLException ex){
35              JOptionPane.showMessageDialog(frm,
36  ex.getMessage(),
37                  "ERROR",JOptionPane.ERROR_MESSAGE);
38          }
39          return list;
40      }
41
42      private static void show_table_film(FilmForm frm,
43  ArrayList<Film> list) throws SQLException{
44          DefaultTableModel model = new DefaultTableModel(0,0);
45
46          String header[] = {"Film ID", "Title", "Description",
47  "Release Year", "Language ID", "Original Language", "Rental
48  Duration", "Rental Rate", "Length",
49              "Replacement Cost", "Rating", "Special Features",
50  "Language Name", "Last Update"};
51
52
53  model.setColumnIdentifiers(set_column_header(frm.getJTFilm(),
54  header));
55          frm.getJTFilm().setModel(model);
56
57          Object[] row = new Object[14];
58
59          for(int i=0; i<list.size(); i++){
60              row[0] = list.get(i).getFilmID();
61              row[1] = list.get(i).getTitle();
62              row[2] = list.get(i).getDescription();
63              row[3] = list.get(i).getReleaseYear();
64              row[4] = list.get(i).getLanguageID();
65              row[5] = list.get(i).getOriginalLanguageID();
66              row[6] = list.get(i).getRentalDuration();
                row[7] = list.get(i).getRentalRate();
                row[8] = list.get(i).getLength();
                row[9] = list.get(i).getReplacementCost();
                row[10] = list.get(i).getRating();
                row[11] = list.get(i).getSpecialFeatures();
                row[12] = list.get(i).getName();
                row[13] = list.get(i).getLastUpdate();

                model.addRow(row);
            }
        }
```

Step 2	In **Film_Utils.java**, define **refresh_controls()** method. This method is a the contents of the **FilmForm** user interface. It performs the following tas

1. Sets the location and title of the **FilmForm** window.
2. Sets alternating row colors for the **JTable** component in the **FilmF**
3. Populates the **JTable** with the list of films retrieved using the **get_f**
4. Populates several **JComboBox** components in the **FilmForm** use from specific columns of the film table in the database. Thes **release_year**, **rating**, **special_features**, **name** (language name),
5. Handles any **SQLExceptions** that might be thrown during the pro dialog box with the message.

```
1     public static void
2  refresh_controls(FilmForm frm){
3
4  frm.setLocationRelativeTo(null);
5          frm.setTitle("FILM FORM");
6
7          //Shows the content of film
8  table and populates combobox
9          try{
10             //Makes alternating
11 color for table rows
12
13 table_renderer(frm.getJTFilm());
14
15             //Populates table
16             ArrayList<Film> list =
17 get_film_list(frm,
18 Query_Film.get_sql_film_joint(),
19 "none");
20             show_table_film(frm,
21 list);
22
23             //Populates jcbFilmID
24             String sql_id = "SELECT
25 film_id FROM film ORDER BY
26 film_id";
27
28 populate_combobox(sql_id,
29 frm.getJCBFilmID(), frm);
30
31             //Populates
32 getJCBTitle()
33             String sql_title =
34  "SELECT DISTINCT title FROM film
35 ORDER BY title";
36
37 populate_combobox(sql_title,
38 frm.getJCBTitle(), frm);
39
40             //Populates
41 getJCBReleaseYear()
42             String sql_year =
43  "SELECT DISTINCT
```

```
44  YEAR(release_year) FROM film ORDER
45  BY release_year";
46
47  populate_combobox(sql_year,
    frm.getJCBReleaseYear(), frm);

                //Populates
    getJCBRating()
                String sql_rating =
     "SELECT DISTINCT rating FROM film
    ORDER BY rating";

    populate_combobox(sql_rating,
    frm.getJCBRating(), frm);

                //Populates
    getJCBSpecialFeatures()
                String sql_feat =
     "SELECT DISTINCT special_features
    FROM film ORDER BY
    special_features";

    populate_combobox(sql_feat,
    frm.getJCBSpecialFeatures(), frm);

                //Populates
    getJCBLanguage() and
    getJCBOriginalLanguage()
                String sql_lang =
     "SELECT name FROM language ORDER
    BY name";

    populate_combobox(sql_lang,
    frm.getJCBLanguage(), frm);

    populate_combobox(sql_lang,
    frm.getJCBOriginalLanguage(), frm);

            }catch (SQLException ex){

    JOptionPane.showMessageDialog(frm,
    ex.getMessage(),

     "ERROR",JOptionPane.ERROR_MESSAGE)
    ;
            }
        }
```

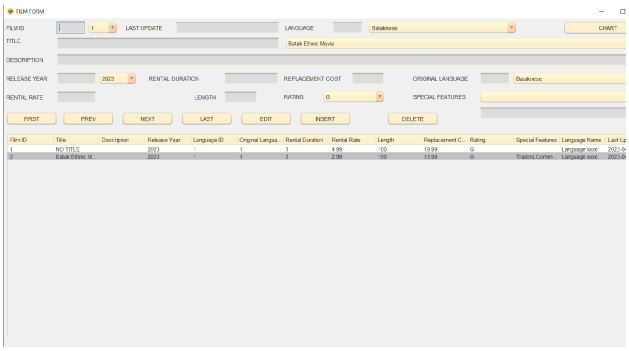


Figure 5.2 The content of **film** table displayed

Step 3

In **FilmForm**'s default constructor, the **Film_Utils.refresh_controls(thi** the controls in the **FilmForm** with data from a database using the **ref Film_Utils** class.

The **this.setIconImage()** method sets the icon of the **F this.setDefaultCloseOperation(this.HIDE_ON_CLOSE)** method sets th to hide the form instead of exiting the application when the close button is

```
1  package sakila;
2  import java.awt.Toolkit;
3  import java.awt.event.ActionEvent;
4  import java.awt.event.ActionListener;
5  import java.io.IOException;
6  import java.text.ParseException;
7  import java.util.logging.Level;
8  import java.util.logging.Logger;
9  import javax.swing.JButton;
10 import javax.swing.JComboBox;
11 import javax.swing.JLabel;
12 import javax.swing.JTable;
13 import javax.swing.JTextField;
14
15 public class FilmForm extends javax.swing.JFrame {
16     public FilmForm() {
17         initComponents();
18         Utility.setLookAndFeel(this);
19         Film_Utils.refresh_controls(this);
20
21  this.setIconImage(Toolkit.getDefaultToolkit().getImage(getClass().
22 ;
23         this.setDefaultCloseOperation(this.HIDE_ON_CLOSE);
24     }
25     //...
26 }
```

Step

Run the project to see the content of **film** table displayed in **jtFilm** as shov

4

If you use the data from **Sakila** MySQL database available in the internet, table displayed in **jtFilm** as shown in Figure 5.3.

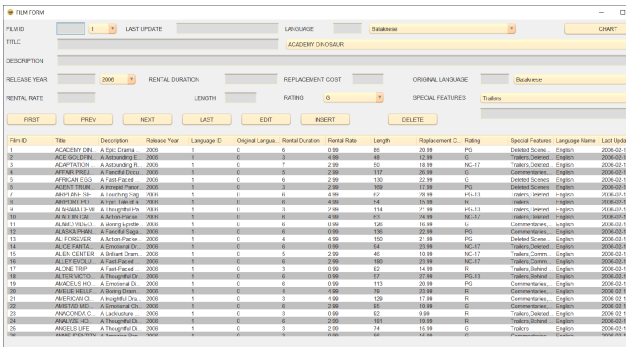


Figure 5.3 The the content of **film** table in original **Sakila** database ava
jtFilm

DISPLAYING AND NAVIGATING DATA ROW BY ROW

Step 1

In **Film_Utils** class, define four new methods. These are event handlers f components in the **FilmForm** form. Each handler is triggered when the u the corresponding **JComboBox** component.

jcbLanguage_handler() and **jcbOriginalLanguage_handler()** upd **JTextField** components with the ID of the selected language from the database.

jcbReleaseYear_handler() updates the **JTextField** component for the selected year.

jcbSpecialFeatures_handler() updates the **JTextField** component for t the selected feature.

```
1     //If user chooses one of items in jcbLanguage
2     public static void jcbLanguage_handler(FilmForm frm){
3         String item =
4 String.valueOf(frm.getJCBLanguage().getSelectedItem());
5         Object lang_id = get_val_from_database("language",
6 "language_id", "name", item);
7         frm.getJTFLanguageID().setText(String.valueOf(lang_id));
8     }
9
10    //If user chooses one of items in jcbOriginalLanguage
11    public static void jcbOriginalLanguage_handler(FilmForm
12 frm){
13        String item =
14 String.valueOf(frm.getJCBOriginalLanguage().getSelectedItem());
15
```

```
        Object lang_id = get_val_from_database("language",
"language_id", "name", item);

frm.getJTFOriginalLanguageID().setText(String.valueOf(lang_id));
    }

    //If user chooses one of items in jcbReleaseYear
    public static void jcbReleaseYear_handler(FilmForm frm){
        String item =
String.valueOf(frm.getJCBReleaseYear().getSelectedItem());
        frm.getJTFReleaseYear().setText(item);
    }

    //If user chooses one of items in jcbSpecialFeatures
    public static void jcbSpecialFeatures_handler(FilmForm frm)
{
        String item =
String.valueOf(frm.getJCBSpecialFeatures().getSelectedItem());
        frm.getJTFSpecialFeatures().setText(item);
    }
```

Step 2

Double click on each **jcbLanguage**, **jcbOriginalLanguage**, **jcbSpecialFeatures** combobox to define its event listener on **FilmForm**:

```
    private void
jcbLanguageActionPerformed(java.awt.event.ActionEvent
evt) {
        Film_Utils.jcbLanguage_handler(this);
    }

    private void
jcbOriginalLanguageActionPerformed(java.awt.event.ActionEvent evt) {
        Film_Utils.jcbOriginalLanguage_handler(this);
    }

    private void
jcbSpecialFeaturesActionPerformed(java.awt.event.ActionEvent evt) {
        Film_Utils.jcbSpecialFeatures_handler(this);
    }

    private void
jcbReleaseYearActionPerformed(java.awt.event.ActionEvent
evt) {
        Film_Utils.jcbReleaseYear_handler(this);
    }
```

These are event handler methods for the corresponding **JComboBo** Whenever the user selects an item from one of the **JComboBoxes**, an **A** and the appropriate event handler method is called. These methods

methods in the **Film_Utils** class to update the relevant text fields in th values.

Step 3

Run the project. Choose one of items on each **jcbLanguage**, **jcbReleaseYear**, and **jcbSpecialFeatures** combobox to see the result as s

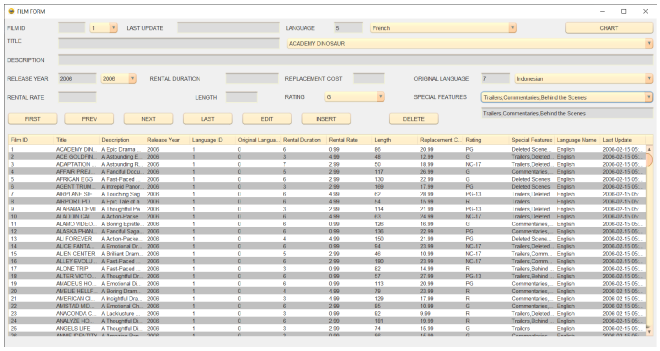


Figure 5.4 User can choose one of items on each **jcbLanguage**, **jcb jcbReleaseYear**, and **jcbSpecialFeatures** combob

Step 4

In **Film_Utils** class, define two new methods named **clear_controls()** and

- **clear_controls()**: This method is used to clear all the text fields in takes the **FilmForm** object as an argument and uses its various g the necessary text fields, then sets their text to empty strings. This the form when there are no results found for a particular query.
- **display_film_data()**: This method is used to display the data retur the **FilmForm** object. It takes three arguments: the **FilmForm** o execute, and an item to be used in the query (e.g. a film ID or establishes a connection to the database using the **getConne** creates a **PreparedStatement** object and sets its first parameter argument. The method then executes the query using **executeQ ResultSet**. If the **ResultSet** is empty (i.e. no rows return **clear_controls()** to clear the form fields. Otherwise, it loops thro a do-while loop and sets the values of the appropriate text fields using its getter methods. It also calls **find_combo_value_selecte** which item in the combo box should be selected based on the query. Finally, the method closes the **ResultSet** and **PreparedSta SQLException** that may occur and displays an error message.

```
1     private static void clear_controls(FilmForm frm){
2         frm.getJTFFilmID().setText("");
3         frm.getJTFTitle().setText("");
4         frm.getJTFDescription().setText("");
5         frm.getJTFReleaseYear().setText("");
6         frm.getJTFLanguageID().setText("");
7         frm.getJTFOriginalLanguageID().setText("");
8         frm.getJTFRentalDuration().setText("");
9         frm.getJTFRentalRate().setText("");
10        frm.getJTFLength().setText("");
```

```
11          frm.getJTFReplacementCost().setText("");
12          frm.getJTFSpecialFeatures().setText("");
13          frm.getJTFLastUpdate().setText("");
14      }
15
16     //Displays film data result row by row
17      private static <T> void display_film_data(FilmForm frm, String
18          try(Connection conn = getConnection()){
19              PreparedStatement ps = conn.prepareStatement(sql);
20              ps.setObject(1,item);
21              ResultSet rs = ps.executeQuery();
22
23              if (!rs.next()) {
24                  // no row found, clear the form fields
25                  clear_controls(frm);
26                  return;
27              }
28
29              do{
30                  frm.getJTFFilmID().setText(String.valueOf(rs.getInt
31                  frm.getJTFTitle().setText(rs.getString("title"));
32                  frm.getJTFDescription().setText(rs.getString("descr
33
34  frm.getJTFReleaseYear().setText(String.valueOf(rs.getInt("release_y
35                  frm.getJTFLanguageID().setText(String.valueOf(rs.ge
36
37  frm.getJTFOriginalLanguageID().setText(String.valueOf(rs.getInt("or
38
39  frm.getJTFRentalDuration().setText(String.valueOf(rs.getInt("rental
40
41  frm.getJTFRentalRate().setText(String.valueOf(rs.getDouble("rental_
42                  frm.getJTFLength().setText(String.valueOf(rs.getInt
43
44  frm.getJTFReplacementCost().setText(String.valueOf(rs.getDouble("re
45                  frm.getJTFSpecialFeatures().setText(rs.getString("s
46                  frm.getJTFLastUpdate().setText(String.valueOf(rs.ge
47
48                  // Determines item selected from jcbFilmID
49                  find_combo_value_selected(frm.getJCBFilmID(), rs.ge
50
51                  // Determines item selected from jcbTitle
52                  find_combo_value_selected(frm.getJCBTitle(), rs.get
53
54                  // Determines item selected from jcbRating
55                  find_combo_value_selected(frm.getJCBRating(), rs.ge
56
57                  // Determines item selected from jcbLanguage
58                  int lang_id = rs.getInt("language_id");
59                  if(lang_id != 0) {
60                      Object lang_name = get_val_from_database("langu
61  "language_id", lang_id);
62                      find_combo_value_selected(frm.getJCBLanguage(),
63                  }
64
```

```
65
66                 // Determines item selected from jcbOriginalLanguage
67                  int ori_lang_id = rs.getInt("original_language_id")
68                  if(ori_lang_id != 0) {
69                      Object lang_name = get_val_from_database("langu
70 "language_id", ori_lang_id);
71                      find_combo_value_selected(frm.getJCBOriginalLan
72                  }
73
74             }while(rs.next());
75
76             rs.close();
77             ps.close();
78         }catch(SQLException ex){
79             JOptionPane.showMessageDialog(frm, ex.getMessage(),
80                     "ERROR",JOptionPane.ERROR_MESSAGE);
81         }
82     }
83
84
85
86
87
88
89
90
91
```

Step 5

In the same class, define another method named **jcbFilm_handler()**. This the **JComboBoxes** (**jcb**) that allow the user to select a film by either its arguments, the FilmForm object and the **JComboBox** object that triggered

The first line of the method gets the selected item from the **getSelectedItem()** method. The selected item could be either the fi depending on which **JComboBox** was used.
Next, the method determines which **JComboBox** is triggered the ev **JComboBox** is equal to the one for film ID or film title. If it is the one fo SQL query to retrieve a film by ID using the **get_sql_id()** method of the the one for film title, it retrieves the SQL query to retrieve a film by title method of the **Query_Film** class.

Finally, the method calls the **display_film_data()** method passing in th SQL query, and the selected item. This method retrieves the film data bas displays it on the form.

```
1     public static void
2 jcbFilm_handler(FilmForm frm,
3 JComboBox<String> jcb) {
4
```

```
        Object item =
jcb.getSelectedItem();
        String sql = "";
        if
(jcb.equals(frm.getJCBFilmID())) {
            sql =
Query_Film.get_sql_id();
        } else if
(jcb.equals(frm.getJCBTitle())) {
            sql =
Query_Film.get_sql_title();
        }

        display_film_data(frm, sql,
item);
    }
```

Step 6

In **FilmForm**, double click on **jcbFilmID** and **jcbTitle** comboboxes and de event handler as follows:

```
    private void
jcbFilmIDActionPerformed(java.awt.event.ActionEvent
evt) {
        Film_Utils.jcbFilm_handler(this,
this.jcbFilmID);
    }

    private void
jcbTitleActionPerformed(java.awt.event.ActionEvent
evt) {
        Film_Utils.jcbFilm_handler(this,
this.jcbTitle);
    }
```

These two methods are event handlers for the **JComboBox** components ir selects an item from the **jcbFilmID** or **jcbTitle** JComboBox, the corres triggered and the associated event handler method is called.

In this case, both of the event handler methods call the same utility me from the **Film_Utils** class, passing in the **FilmForm** object and the **JCo** triggered the event. The **jcbFilm_handler()** method then determines th based on the **JComboBox** component that triggered the event, and calls method to display the film data in the form based on the selected item.

Step 7

Run the project. Choose one of items in **jcbFilmID** and/or **jcbTitle** combo the content of **film** table as shown in Figure 5.5.

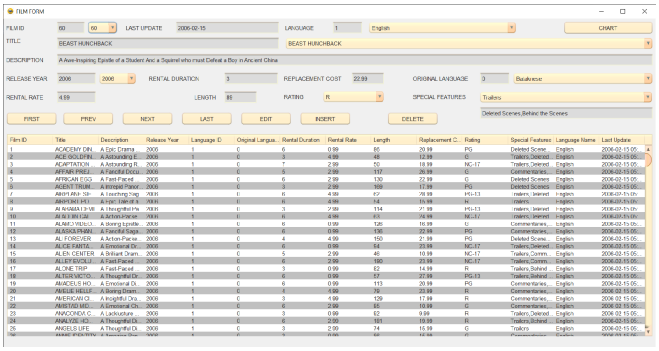


Figure 5.5 Displaying row by row the content of **film**

Step 8

Define four navigating methods in **Film_Utils** class. These are methods t film data rows displayed on the form. Here is a brief description of each m

- **show_first_row()**: Displays the first row of film data. It retriev **jcbFilmID** combo box and calls **display_film_data()** method to d data.
- **show_last_row()**: Displays the last row of film data. It retriev **jcbFilmID** combo box and calls **display_film_data()** method to d data.
- **show_prev_row()**: Displays the previous row of film data. It decr variable by 1, checks if it is within the valid range, retrieves the c **jcbFilmID** combo box and calls **display_film_data()** method to d data.
- **show_next_row()**: Displays the next row of film data. It incre variable by 1, checks if it is within the valid range, retrieves the c **jcbFilmID** combo box and calls **display_film_data()** method to d data.

All these methods call **display_film_data()** method to display the film **currentIndex** variable to keep track of the current row displayed on the f

```
1     public static void show_first_row(FilmForm frm){
2         String item =
3 String.valueOf(frm.getJCBFilmID().getItemAt(FIRST_INDEX));
4         display_film_data(frm, SQL_ID, item);
5         currentIndex = FIRST_INDEX;
6     }
7
8     public static void show_last_row(FilmForm frm){
9         int endIndex = frm.getJCBFilmID().getItemCount() -
10 1;
11        String item =
12 String.valueOf(frm.getJCBFilmID().getItemAt(endIndex));
13        display_film_data(frm, SQL_ID, item);
14        currentIndex = endIndex;
15    }
16
17    public static void show_prev_row(FilmForm frm){
```

```
        currentIndex--;
        if(currentIndex < FIRST_INDEX){
            currentIndex = FIRST_INDEX;
            return;
        }
        String item =
String.valueOf(frm.getJCBFilmID().getItemAt(currentIndex));
        display_film_data(frm, SQL_ID, item);
    }

    public static void show_next_row(FilmForm frm){
        int endIndex = frm.getJCBFilmID().getItemCount() -
1;
        currentIndex++;
        if(currentIndex > endIndex){
            currentIndex = endIndex;
            return;
        }
        String item =
String.valueOf(frm.getJCBFilmID().getItemAt(currentIndex));
        display_film_data(frm, SQL_ID, item);
    }
```

Step 9

Then in **FilmForm**, double click on each navigation buttons to define th handler:

```
    private void
jbFirstActionPerformed(java.awt.event.ActionEvent
evt) {
       Film_Utils.show_first_row(this);
    }

    private void
jbLastActionPerformed(java.awt.event.ActionEvent
evt) {
        Film_Utils.show_last_row(this);
    }

    private void
jbPrevActionPerformed(java.awt.event.ActionEvent
evt) {
        Film_Utils.show_prev_row(this);
    }

    private void
jbNextActionPerformed(java.awt.event.ActionEvent
evt) {
        Film_Utils.show_next_row(this);
    }
```

These methods are event handlers for the four buttons in the user interfa navigate through the list of films.

- **jbFirstActionPerformed()**: When the user clicks this button method from the **Film_Utils** class is called, which displays the fi the user interface.
- **jbLastActionPerformed()**: When the user clicks this button, the **s** from the **Film_Utils** class is called, which displays the last row o interface.
- **jbPrevActionPerformed()**: When the user clicks this button method from the **Film_Utils** class is called, which displays the pre in the user interface.
- **jbNextActionPerformed()**: When the user clicks this button method from the **Film_Utils** class is called, which displays the ne the user interface.

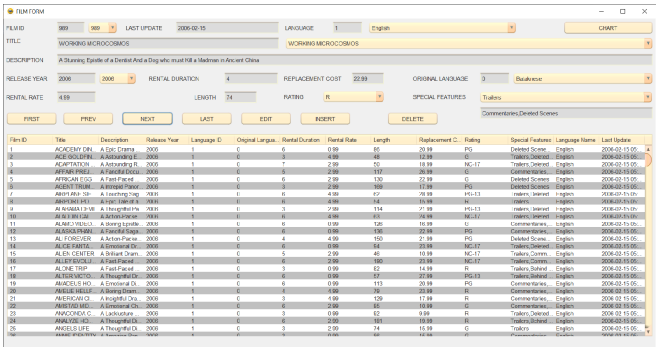


Figure 5.6 User clicks on one or more navigation buttons o

Step 10	Run the project. Click on one or more navigation buttons to see the result
Step 11	Define **mouse_pressed_handler()** method in **Film_Utils** class. This me press event on the **JTable** component in the **FilmForm**. It first checks if not, it displays an information dialog prompting the user to select a row retrieves the film ID from the selected row and displays the film data **display_film_data()** method with the **get_sql_id query**.

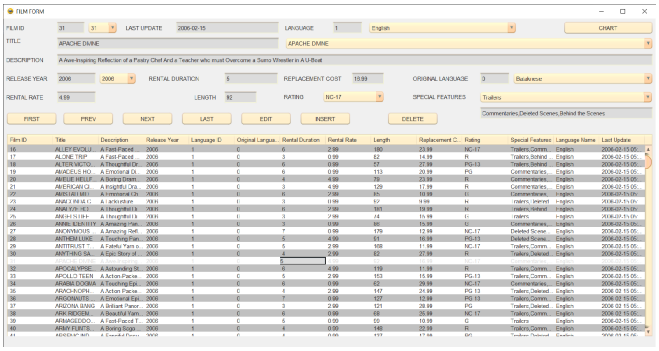


Figure 5.7 User double-clicks on any row in **jtFilr**

If there is an error retrieving or displaying the film data, it logs the err dialog with the error message and stack trace.

```
    public static void mouse_pressed_handler(FilmForm frm) {
        Objects.requireNonNull(frm, "frm must not be null");

        int selectedIndex = frm.getJTFilm().getSelectedRow();
        if (selectedIndex == -1) {
            JOptionPane.showMessageDialog(frm, "Please select a row to view its
data.",
                "No row selected",
JOptionPane.INFORMATION_MESSAGE);
            return;
        }

        try (Connection conn = getConnection()) {
            String id =
String.valueOf(frm.getJTFilm().getModel().getValueAt(selectedIndex,
0));

            // Displays film data
            display_film_data(frm, Query_Film.get_sql_id(), id);

        } catch (SQLException ex) {

Logger.getLogger(FilmForm.class.getName()).log(Level.SEVERE, "Error
displaying film data", ex);
            String message = "Error displaying film data: " +
ex.getMessage();
            String stackTrace =
Arrays.toString(ex.getStackTrace());
            JOptionPane.showMessageDialog(frm, message + "\n\n" +
stackTrace, "ERROR", JOptionPane.ERROR_MESSAGE);
        }
    }
```

Step	Right click on **jtFilm**. Then, choose **Events > Mouse > mousePressed**. D

12	 1 private void jtFilmMousePressed(java.awt.event.MouseEvent evt) { 2 Film_Utils.mouse_pressed_handler(this); 3 }
Step 13	Run the project. Double click on any row in **jtFilm** table. You will see t **film** table displayed in textfields and comboboxes as shown in Figure 5.7.

UPDATING RECORD
UPDATING RECORD

Step 1	In **Film_Utils** class, define a new method named **update_row_by_film_id()**. The method is designed to update a row of data in the **film** table of a database based on the film ID. The method takes several parameters including a **FilmForm** object that allows users to input the data needed to update a row in the table. The other parameters include the film ID, title, description, release year, language ID, original language ID, rental duration, rental rate, length, replacement cost, rating, and special features. These parameters are used to construct a new **Film** object, which will contain the updated data. The method begins by establishing a connection to the database using the **getConnection()** method. This method returns a **Connection** object, which represents a connection to the database. The **ResultSet** object is initialized to null, and two SQL queries are defined: one to retrieve the film ID of the row to be updated, and another to update the row with the new data. Both queries are defined using a multi-line string syntax,

which makes it easier to write SQL queries that span multiple lines.

The try-with-resources statement is used to declare two **PreparedStatement** objects, which are used to execute the SQL queries. The first **PreparedStatement** object is used to retrieve the film ID of the row to be updated. It uses the **query_id** string, which includes a placeholder for the film ID. The **PreparedStatement** object is then initialized with the film ID using the **setInt()** method. The **execute()** method is called to execute the query, and if it returns false (i.e., there is no result set), an error message is displayed using the **JOptionPane** class. Otherwise, the **ResultSet** is retrieved using the **getResultSet()** method, and the **next()** method is called to move the cursor to the first row of the **ResultSet**.

Next, a new **Film** object is created using the thirteen-parameter constructor. This constructor takes the film ID and all the other parameters, and initializes the corresponding fields of the Film object. The **PreparedStatement** object for the update query is then initialized with the new data using various **setXXX()** methods. The corresponding values from the **Film** object are used to set the parameters for the **PreparedStatement** object. Finally, the **executeUpdate()** method is called to execute the update query, which updates the row in the database with the new data.

Finally, all the database resources are closed using the **close()** method. If there is an **SQLException** or **NumberFormatException**, a log is created using the **Logger** class, and an error message with a stack trace is displayed using the **JOptionPane** class.

	1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 55 56 57 58

```
59
60
61
```

```
62      //Updates row of data in film tabel by film_id
63      public static void update_row_by_film_id(FilmForm frm,
64  int film_id, String title, String desc,
65          int year, int lang_id, int ori_lang_id, int
66  duration, double rate, int length, double cost,
67          String rating, String features) throws SQLException{
68          Connection conn = getConnection();
69          ResultSet rs = null;
70          String query_id = "SELECT film_id FROM film WHERE
71  film_id = ?";
72          String update_query = """
73              UPDATE film SET title = ?, description = ?,
                    release_year = ?, language_id = ?,
  original_language_id = ?,
                    rental_duration = ?, rental_rate = ?, length
  = ?,
                    replacement_cost = ?,
                    rating = ?, special_features = ? WHERE
  film_id = ?""";
          try(PreparedStatement idPs =
  conn.prepareStatement(query_id,

  ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_READ_ONLY);
          PreparedStatement updatePS =
  conn.prepareStatement(update_query,
                    ResultSet.TYPE_SCROLL_SENSITIVE,
  ResultSet.CONCUR_UPDATABLE))
          {
              idPs.setInt(1,film_id);
              if(!idPs.execute()){
                  String message = "Can't find film_id " +
  film_id;

                  JOptionPane.showMessageDialog(frm, message,
                      "ERROR",JOptionPane.ERROR_MESSAGE);
              } else{
                  rs = idPs.getResultSet();
                  rs.next();

                  //Creates an Film object using thirteen-
  params constructor
                  Film obj = new Film(film_id, title, desc,
  year, lang_id, ori_lang_id, duration, rate, length, cost,
  rating, features, new
  Timestamp(System.currentTimeMillis()));
                  updatePS.setString(1, obj.getTitle());
                  updatePS.setString(2, obj.getDescription());
                  updatePS.setInt(3, obj.getReleaseYear());
                  updatePS.setInt(4, obj.getLanguageID());
                  updatePS.setInt(5,
  obj.getOriginalLanguageID());
                  updatePS.setInt(6, obj.getRentalDuration());
                  updatePS.setDouble(7, obj.getRentalRate());
```

```
                updatePS.setInt(8, obj.getLength());
                updatePS.setDouble(9,
obj.getReplacementCost());
                updatePS.setString(10, obj.getRating());
                updatePS.setString(11,
obj.getSpecialFeatures());
                updatePS.setInt(12, obj.getFilmID());

                updatePS.executeUpdate();
                rs.close();
                updatePS.close();
                idPs.close();
                conn.close();
            }
        }catch(SQLException ex){

Logger.getLogger(FilmForm.class.getName()).log(Level.SEVERE,
"Error updating film data", ex);
            String message = "Error updating film data: " +
ex.getMessage();
            String stackTrace =
Arrays.toString(ex.getStackTrace());
            JOptionPane.showMessageDialog(null, message +
"\n\n" + stackTrace, "ERROR", JOptionPane.ERROR_MESSAGE);
        }catch(java.lang.NumberFormatException ex){

Logger.getLogger(FilmForm.class.getName()).log(Level.SEVERE,
"Invalid Input", ex);
            String message = "Invalid Input: " +
ex.getMessage();
            String stackTrace =
Arrays.toString(ex.getStackTrace());
            JOptionPane.showMessageDialog(null, message +
"\n\n" + stackTrace, "ERROR", JOptionPane.ERROR_MESSAGE);
        }
    }
```

Step 2	Then in **read_inpu** and return this metho object and Inside the data. The various co **jcbFilmID** individual **S**

Next, the
inputs usi
checks wh
than zer
IllegalArg
error mes
displayed t

If all input
HashMap

Finally, the
returned.

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
```

```
38
39 "Invali
40
41
42
43
44 ex.getM
45
46
47
48
49
50
55
56
57
58 IllegalA
59
60 negativ
61
62
63
64 "Invali
65
66
67
68
69 ex.getM
70
71
72
73
74
75
76 cannot
   cannot
```

```
    }
```

Step 3

Still in th
edit_actua
database b

First, the
the user ir
returns a
values. The
and assigne

The meth
method w
method up
on the film

After upd
refresh_co
FilmForm,
an exceptio
the error m

```
1     pri
2  frm){
3
4
5  read_ir
6
7  Intege
8
9  input_
10
11 input_
12
13 Intege
14
15 Intege
16
17
18 Intege
19
20 Intege
21
22 Double.
```

```
23
24 Integer
25
26 Double.
27
28
29 input_c
30
   input_c


   title,

   duratic




   ex.getM

    "ERROR

      }
```

Step 4

Lastly, defi
and **edit_f**
editing of
method is
the **FilmFo**

If the text
text to "CO
form, so th

If the text c
the text ba
which upda
changes m
all controls

If there is
method dis

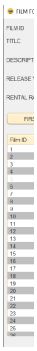
```
1     pri
2 enable_
3 {
```

```
4
5
6
7
8
9
10
11
12  frm.get
13      }
14
15      pub
16  frm){
17
18   if(frm
19  ){
20
21  frm.get
22
23
24
25
26
27
28
29
        }
```

Step 5

Run the
jcbTitle c
jtFilm (in
button as s

Edit the an
language).
had been s



Fi

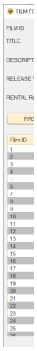


Figure 5

UPDATING RECORD DIRECTLY ON JTABLE
UPDATING RECORD DIRECTLY ON JTABLE

Step 1	In **Film_Utility** class, define a new method nam **edit_database_from_jtable()**. It handles the **TableModelEvent** that occ when a user updates a row in the **JTable** in the **FilmForm**. It extracts updated data from the table model, converts it to the appropriate data typ and then calls the **update_row_by_film_id()** method to update corresponding row in the database. If there is a **SQLIntegrityConstraintViolationExcepti SQLSyntaxErrorException**, or **SQLException**, it catches the exception a displays a **JOptionPane** message to the user with the appropriate er message. Finally, it refreshes all objects on the form by calling the **refresh_control** method.

```
1     public static void
2 edit_database_from_jtable(TableModelEvent e, FilmForm frm){
3         if (e.getType() == TableModelEvent.UPDATE) {
4             int row = e.getFirstRow();
5             TableModel model = (TableModel)e.getSource();
6
7
```

```
            int film_id =
Integer.parseInt(String.valueOf(model.getValueAt(row, 0)));
            String title =
String.valueOf(model.getValueAt(row, 1));
            String desc =
String.valueOf(model.getValueAt(row, 2));
            int year =
Integer.parseInt(String.valueOf(model.getValueAt(row, 3)));
            int lang_id =
Integer.parseInt(String.valueOf(model.getValueAt(row, 4)));
            int ori_lang_id =
Integer.parseInt(String.valueOf(model.getValueAt(row, 5)));
            int duration =
Integer.parseInt(String.valueOf(model.getValueAt(row, 6)));
            double rate =
Double.parseDouble(String.valueOf(model.getValueAt(row, 7)));
            int length =
Integer.parseInt(String.valueOf(model.getValueAt(row, 8)));
            double cost =
Double.parseDouble(String.valueOf(model.getValueAt(row, 9)));
            String rating =
String.valueOf(model.getValueAt(row, 10));
            String features =
String.valueOf(model.getValueAt(row, 11));

            try{
                update_row_by_film_id(frm, film_id, title,
desc,
                    year, lang_id, ori_lang_id, duration,
rate, length, cost,
                    rating, features);

                //Refreshes all objects on form
                refresh_controls(frm);

            } catch
(SQLIntegrityConstraintViolationException ex) {

Logger.getLogger(FilmForm.class.getName()).log(Level.SEVERE,
"Duplicate entry", ex);
                JOptionPane.showMessageDialog(frm, "Error:
Duplicate entry\n" + ex.getMessage());
            } catch (SQLSyntaxErrorException ex) {

Logger.getLogger(FilmForm.class.getName()).log(Level.SEVERE,
"Invalid SQL syntax", ex);
                JOptionPane.showMessageDialog(frm, "Error:
Invalid SQL syntax\n" + ex.getMessage());
            } catch (SQLException ex) {

Logger.getLogger(FilmForm.class.getName()).log(Level.SEVERE,
"Database error", ex);
```

```
                JOptionPane.showMessageDialog(frm, "Error:
Database error\n" + ex.getMessage());
            }
        }
    }
```

Step 2

Create a new public class named **FilmTableModelListener**. It defines custom event listener for the **JTable** on the **FilmForm** form. Whenever user modifies a cell in the table, the **tableChanged()** method of the listen is called.

The **tableChanged()** method calls the **edit_database_from_jtabl** method of the **Film_Utils** class, which handles the database update based the edited cell. Then, it stops the cell editor to commit the change.

The constructor of the **FilmTableModelListener** takes two parameters: **JTable** component and the **FilmForm** form, which are stored in instan variables. This allows the listener to access these components when neede

```
1  package sakila;
2  import
3  javax.swing.event.TableModelEvent;
4  import
5  javax.swing.event.TableModelListener;
6  import javax.swing.JTable;
7
8  public class FilmTableModelListener
9  implements TableModelListener {
10     private final JTable jt;
11     private final FilmForm frm;
12
13     public
14 FilmTableModelListener(JTable jt,
15 FilmForm frm) {
16         this.jt = jt;
17         this.frm = frm;
18     }
19
20     @Override
21     public void
22 tableChanged(TableModelEvent e) {
23
   Film_Utils.edit_database_from_jtable(e,
   frm);

         if (jt.getCellEditor() != null)
   {

   jt.getCellEditor().stopCellEditing();
```

```
        }
    }
}
```

Step 3

Right click on **jtFilm**. Then, choose **Events > Mouse > mouseClick** Define its event handler:

```
    private void jtFilmMouseClicked(java.awt.event.MouseEvent evt) {
        // instantiate FilmTableModelListener and add it as a listen
to the table model
        FilmTableModelListener tableModelListener = new
FilmTableModelListener(this.getJTFilm(), this);

 this.getJTFilm().getModel().addTableModelListener(tableModelListene
;
    }
```

This method is an event handler for when the user clicks on a cell in **JTable** displaying the film data. When the user clicks on a cell, a n **FilmTableModelListener** object is created and added as a listener to table model.

The **FilmTableModelListener** class is responsible for listening to chan made in the table, such as when a user edits a cell, and then updating database accordingly. By adding this listener to the table model, the progr can automatically detect when changes are made and update the datab without requiring the user to manually click a "save" button or perform a other actions.

Step 4

Run the project. Click on any cell in any column (except first colum) in a row in **jtFilm** that you want to edit. Then, change it. Then, click any wh outside the corresponding cell. The edited data had been saved i database.

INSERTING NEW RECORD
INSERTING NEW RECORD

Step 1

In **Film_Utils** class, define a method named **insert_row()**. This method is responsible for inserting a new row into the **film** table in the database. It first reads the input data from the form, which is passed in as a parameter (**frm**), and stores it in a **HashMap**. The input data includes the film title, description, release year, language ID, original language ID, rental duration, rental rate, length, replacement cost, rating, and special features.

Next, it creates a SQL INSERT statement as a string, with placeholders for each of the values to be inserted. It then uses a try-with-resources block to create a database connection and a prepared statement. The prepared statement is created from the SQL string, and each of the placeholders is replaced with the corresponding value from the **Film** object created using the input data.

Finally, the prepared statement is executed using the **executeUpdate()** method, which inserts the new row into the **film** table.

If an exception is caught while executing the SQL statement, it is logged and an error message is displayed in a dialog box on the form.

```
1     //Inserts new row into film table
2     private static void insert_row(FilmForm frm) throws
3  SQLException{
4          HashMap<String, String> input_data =
5  read_inputs(frm);
6          String title = input_data.get("title");
7          String desc = input_data.get("desc");
8          int year = Integer.parseInt(input_data.get("year"));
9          int lang_id =
10 Integer.parseInt(input_data.get("lang_id"));
11
12         int ori_lang_id =
13 Integer.parseInt(input_data.get("ori_lang_id"));
14         int duration =
15 Integer.parseInt(input_data.get("duration"));
16         double rate =
17 Double.parseDouble(input_data.get("rate"));
18         int length =
19 Integer.parseInt(input_data.get("length"));
20         double cost =
21 Double.parseDouble(input_data.get("cost"));
22
23         String rating = input_data.get("rating");
24         String features = input_data.get("features");
25
26         // SQL insert statement
27         String sql = """
28             INSERT INTO film(title, description,
29 release_year, language_id,
30                 original_language_id, rental_duration,
31 rental_rate, length,
32                 replacement_cost, rating, special_features)
33                 VALUES(?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)""";
34
```

```
        try(Connection conn = getConnection();
         PreparedStatement pstmt =
conn.prepareStatement(sql)){
            //Creates an Film object using twelve-params
constructor
            Film obj = new Film(title, desc, year, lang_id,
ori_lang_id,
                duration, rate, length, cost, rating,
features, new Timestamp(System.currentTimeMillis()));
            pstmt.setString(1, obj.getTitle());
            pstmt.setString(2, obj.getDescription());
            pstmt.setInt(3, obj.getReleaseYear());
            pstmt.setInt(4, obj.getLanguageID());
            pstmt.setInt(5, obj.getOriginalLanguageID());
            pstmt.setInt(6, obj.getRentalDuration());
            pstmt.setDouble(7, obj.getRentalRate());
            pstmt.setInt(8, obj.getLength());
            pstmt.setDouble(9, obj.getReplacementCost());
            pstmt.setString(10, obj.getRating());
            pstmt.setString(11, obj.getSpecialFeatures());

            //Executes the sql insert statement
            pstmt.executeUpdate();
        } catch (SQLException ex) {

Logger.getLogger(FilmForm.class.getName()).log(Level.SEVERE,
"Database error", ex);
            JOptionPane.showMessageDialog(frm, "Error:
Database error\n" + ex.getMessage());
        }
    }
```

Step 2

Still in **Film_Utils.java**, define **insert_actual()** and **insert_handler()** methods. The **insert_handler()** method is responsible for handling the logic when the user clicks the "Insert" button on the GUI form. When the button is clicked for the first time, the method changes the button's label to "CONFIRM", disables the "Edit" button, disables all controls on the form except the "Insert" button, and clears all input fields.

Once the user has filled in the input fields, they can click the "CONFIRM" button to insert the new row into the film table in the database using the **insert_actual()** method. If the insertion is successful, the method refreshes the table and combo boxes, enables the "Edit" button, and re-enables all controls.

If an error occurs during the insertion process, the method displays an error message dialog box.

```
    private static void insert_actual(FilmForm
frm){
        try{
            insert_row(frm);

            //Refreshes table and comboboxes
            refresh_controls(frm);

        }catch(SQLException ex){
            JOptionPane.showMessageDialog(frm,
ex.getMessage(),

 "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    public static void insert_handler(FilmForm
frm){

 if(frm.getJBInsert().getText().equals("INSERT")
){

frm.getJBInsert().setText("CONFIRM");

            //Disables jbEdit
            frm.getJBEdit().setEnabled(false);

            // Disables controls
            enable_controls(false, frm);

frm.getJCBFilmID().setEnabled(false);
            frm.getJCBTitle().setEnabled(false);

            // Clears controls
            clear_controls(frm);

            // Enables
            frm.getJBInsert().setEnabled(true);
        }

        else {
            frm.getJBInsert().setText("INSERT");

            // Actual insertion
            insert_actual(frm);

            //Enables jbEdit
            frm.getJBEdit().setEnabled(true);

            //Enables controls
            enable_controls(true, frm);
            frm.getJCBFilmID().setEnabled(true);
            frm.getJCBTitle().setEnabled(true);
```

```
        }
    }
```

Step 3

In **FilmForm.java**, double click on INSERT button to create its event listener:

```
    private void jbInsertActionPerformed(java.awt.event.ActionEvent evt) {
        Film_Utils.insert_handler(this);
    }
```

Step 4

Run the project. Click on INSERT button. You will see the state of film form when insertion is in progress as shown in Figure 5.10.

Then, fill in all fields. Then, click CONFIRM button to save the new record into **film** table as shown in Figure 5.11.

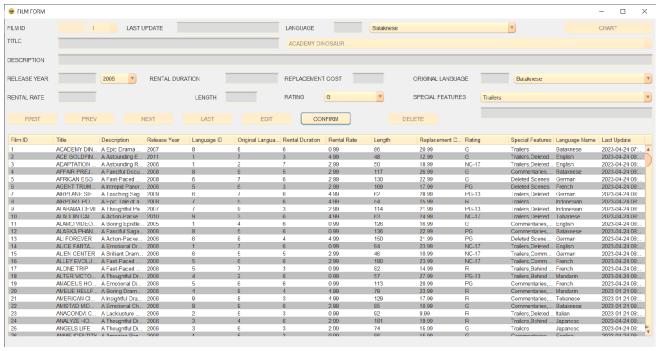


Figure 5.10 When user clicks on INSERT button, the film form will be in state of insertion

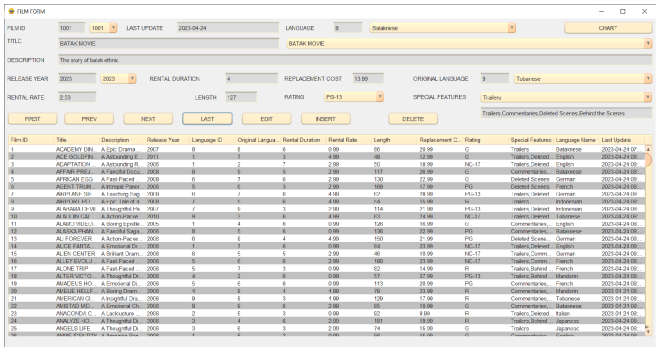


Figure 5.11 The new data had been saved into **film** table

DELETING RECORD
DELETING RECORD

Step 1

Then in **Film_Utils** class, define **delete_handler()** method. It is respor for handling the delete operation of a selected row from the **film** table. It gets the selected film ID from the **JComboBox** component in the **FilmI** object. Then it shows a confirmation dialog box asking the user whether want to delete the selected row or not. If the user confirms the deleti prepares a SQL DELETE statement to delete the row from the database **PreparedStatement** is used to avoid SQL injection attacks, and the **film_** set as a parameter in the statement. If the deletion is successful, **refresh_controls()** method is called to update the **JTable** and **JCombo** components in the **FilmForm**. If there is an exception during the del process, an error message is shown to the user using a **JOptionPane**.

```
1     public static void delete_handler(FilmForm frm){
2         int dialogButton = JOptionPane.YES_NO_OPTION;
3         int film_id =
4 Integer.parseInt(String.valueOf(frm.getJCBFilmID().getSelectedItem(
5 
6         String message = String.format("Are you sure you want to de
7 the row Film ID: %d)", film_id);
8         int answer = JOptionPane.showConfirmDialog(frm, message,
9 "DELETING ROW OF DATA", dialogButton);
10 
11        if(answer == JOptionPane.YES_OPTION){
12            String query = """
13                DELETE FROM film WHERE film_id = ?""";
14            try(Connection conn = getConnection();
15                PreparedStatement ps = conn.prepareStatement(query)
16                // Use PreparedStatement to avoid SQL injection att
17                ps.setInt(1, film_id);
18                ps.executeUpdate();
19 
20                // Refresh table and comboboxes
21                refresh_controls(frm);
22 
23            } catch (SQLException ex){
24                JOptionPane.showMessageDialog(frm, ex.getMessage(),
25                    "ERROR",JOptionPane.ERROR_MESSAGE);
26            }
27        }
28    }
```

Step 2

In **FilmForm.java**, double click on DELETE button to generate its listener:

```
    private void
jbDeleteActionPerformed(java.awt.event.ActionEvent
evt) {
        Film_Utils.delete_handler(this);
    }
```

Step 3	Run the project. Choose **film_id** using **jcbFilmID** or **jcbFilm** combobox. T Click on DELETE button. The corresponding row of data had been deleted database.

PLOTTING CHART
PLOTTING CHART

Step 1	Create a new **JFrame** and save it as **Charts_Film.java**.
Step 2	In **Charts_Film.java**, add six **JPanels** and set their corresponding **Variable Name** as **jPanel1**, **jPanel2**, **jPanel3**, **jPanel4**, **jPanel5**, and **jPanel6**. Then, add getter method for each object as follows:

```
    //Getter method for jPanel1
    public JPanel getJPanel1(){
        return this.jPanel1;
    }

    //Getter method for jPanel2
    public JPanel getJPanel2(){
        return this.jPanel2;
    }

    //Getter method for jPanel3
    public JPanel getJPanel3(){
        return this.jPanel3;
    }

    //Getter method for jPanel4
    public JPanel getJPanel4(){
        return this.jPanel4;
    }

    //Getter method for jPanel5
    public JPanel getJPanel5(){
        return this.jPanel5;
    }

    //Getter method for jPanel6
```

```
27      public JPanel getJPanel6(){
28          return this.jPanel6;
29      }
```

Step 3

In **Film_Utils** class, define four new methods. These are four methods used to draw different types of charts for film distribution data. Here is an explanation of each method:

1. **draw_pie_chart_film_by_year(Charts_Film frm, JPanel jp)** - This method creates a pie chart to show the distribution of films by release year. It first creates a **DefaultPieDataset** object by calling the **create_pie_dataset()** method, passing in a SQL query and the column names for the data and labels. Then, it calls the **draw_piechart_with_dataset()** method, passing in the **frm** and **jp** objects, the dataset, and a title for the chart.
2. **draw_bar_chart_film_by_rating(Charts_Film frm, JPanel jp)** - This method creates a bar chart to show the distribution of films by rating. It first creates a **DefaultCategoryDataset** object by calling the **create_bar_dataset()** method, passing in a SQL query and the column names for the data and labels. Then, it calls the **draw_barchart_with_dataset()** method, passing in the **frm** and **jp** objects, the dataset, a title for the chart, and labels for the x and y axes.
3. **draw_pie_chart_film_by_duration(Charts_Film frm, JPanel jp)** - This method creates a pie chart to show the distribution of films by rental duration. It first creates a **DefaultPieDataset** object by calling the **create_pie_dataset()** method, passing in a SQL query and the column names for the data and labels. Then, it calls the **draw_piechart_with_dataset()** method, passing in the **frm** and **jp** objects, the dataset, and a title for the chart.
4. **draw_bar_chart_film_by_language(Charts_Film frm, JPanel jp)** - This method creates a bar chart to show the distribution of films by language. It first creates a **DefaultCategoryDataset** object by calling the **create_bar_dataset()** method, passing in a SQL query and the column names for the data and labels. Then, it calls the **draw_barchart_with_dataset()** method, passing in the **frm** and **jp** objects, the dataset, a title for the chart, and labels for the x and y axes.

```
1     private static void
2  draw_pie_chart_film_by_year(Charts_Film frm, JPanel jp){
3
```

```
        jp.setPreferredSize(new Dimension(jp.getWidth(),
jp.getHeight()));
        DefaultPieDataset dataset =
create_pie_dataset(Query_Film.get_sql_film_year_dist(),
"Number", "year");

        //Draws piechart film distribution by release year
        draw_piechart_with_dataset(frm, jp, dataset, "TOP
10 FILM DISTRIBUTION BY RELEASE YEAR");
    }

    private static void
draw_bar_chart_film_by_rating(Charts_Film frm, JPanel jp){
        jp.setPreferredSize(new Dimension(jp.getWidth(),
jp.getHeight()));

        DefaultCategoryDataset dataset =
create_bar_dataset(Query_Film.get_sql_film_rating_dist(),
"Number", "rating");

        //Draws barchart film distribution by rating
        draw_barchart_with_dataset(frm, jp, dataset, "TOP
10 FILM DISTRIBUTION BY RATING", "RATING", "NUMBER");
    }

    private static void
draw_pie_chart_film_by_duration(Charts_Film frm, JPanel jp)
{
        jp.setPreferredSize(new Dimension(jp.getWidth(),
jp.getHeight()));
        DefaultPieDataset dataset =
create_pie_dataset(Query_Film.get_sql_film_duration_dist(),
"Number", "duration");

        //Draws piechart film distribution by rental
duration
        draw_piechart_with_dataset(frm, jp, dataset, "TOP
10 FILM DISTRIBUTION BY RENTAL DURATION");
    }

    private static void
draw_bar_chart_film_by_language(Charts_Film frm, JPanel jp)
{
        jp.setPreferredSize(new Dimension(jp.getWidth(),
jp.getHeight()));

        DefaultCategoryDataset dataset =
create_bar_dataset(Query_Film.get_sql_film_lang_dist(),
"Number", "l1.name");

        //Draws barchart film distribution by language
        draw_barchart_with_dataset(frm, jp, dataset, "TOP
10 FILM DISTRIBUTION BY LANGUAGE", "LANGUAGE", "NUMBER");
```

```
    }
```

Step 4

In **Film_Utils** class, define another two new methods. These two methods are used to create a categorized pie chart to show the distribution of rental rates in films. Here is an explanation of each method:

1. **categorized_rental_rate_piedataset(Charts_Film frm, JPanel jp, String table)** - This method creates a categorized pie dataset from the rental rates in a given table. It first calls the **get_col_val_from_database()** method to retrieve a list of rental rates from the specified table. Then, it creates a **Map<String, Double>** object to map the rental rate ranges to their corresponding count. Next, it calls the **categorize_values()** method to categorize the rental rates into the specified ranges and count the number of films in each range. Finally, it creates a **DefaultPieDataset** object from the categorized values using the **create_pie_dataset_from_categorized_vals()** method and calls the **draw_piechart_with_dataset()** method to draw the chart.

2. **draw_pie_chart_categorized_rental_rate(Charts_Film frm, JPanel jp)** - This method is called to draw the categorized rental rate pie chart. It sets the preferred size of the **JPanel** object and then calls the **categorized_rental_rate_piedataset()** method, passing in the **frm** and **jp** objects and the name of the table containing the rental rate data.

```
1
```

```
    private static void
categorized_rental_rate_piedataset(Charts_Film frm,
JPanel jp, String table) {
        try {
            List<Number> rates =
get_col_val_from_database("rental_rate", table);

            Map<String, Double> range_vals = new
LinkedHashMap<>();
            range_vals.put("< $1", 1.0);
            range_vals.put("$1 < x < $2",2.0);
            range_vals.put("$2 < x < $3", 3.0);
            range_vals.put("$3 < x < $5", 5.0);
            range_vals.put("x > $5",
Double.MAX_VALUE);

            Map<String, Double> rate_counts =
categorize_values(rates, range_vals);

            DefaultPieDataset dataset =
create_pie_dataset_from_categorized_vals(rate_counts);
            draw_piechart_with_dataset(frm, jp,
dataset,"CATEGORIZED RENTAL RATE DISTRIBUTION IN
FILM");
        } catch (SQLException ex) {
            JOptionPane.showMessageDialog(null,
ex.getMessage(),
                    "ERROR",
JOptionPane.ERROR_MESSAGE);
        } catch (ClassNotFoundException ex) {
            JOptionPane.showMessageDialog(null,
ex.getMessage(),
                    "ERROR",
JOptionPane.ERROR_MESSAGE);
        }
    }

    public static void
draw_pie_chart_categorized_rental_rate(Charts_Film
frm, JPanel jp){
        jp.setPreferredSize(new
Dimension(jp.getWidth(), jp.getHeight()));
        categorized_rental_rate_piedataset(frm, jp,
 "film");
    }
```

Step 5

Still in **Film_Util**
categorized_len
length values of f
chart to display t

takes three argu
class which is u
JPanel instance
table is the nam
stored.
The method fir
method with the
retrieve a list o
creates a **Linked**
range of lengths
bound of that ran
for films with ler
with length betw
with length betw
with length betw
length greater th

Next, the metho
the list of length
Map object **leng**
each range. Th
length values in
counts the numbe

Then, th
create_pie_data
the **length_coun**
dataset that ca
create_pie_data
converts the **len**
object, which is
method to draw t

Finally, the meth
method with the
is used to s
draw_piechart_v
with the given da
any SQL exceptio
the execution of
the **JOptionPane**

The **draw_pie_ch**
method that ca
distribution pie
which are the s
categorized_len

	argument is fixe
	draw the categor
	data.
	
	1 private s
	2 categorized_l
	3 jp, String ta
	4 try {
	5 L
	6 get_col_val_f
	7
	8 M
	9 LinkedHashMap
	10 r
	11 r
	12 r
	13 r
	14 r
	15
	16 M
	17 categorize_va
	18
	19 D
	20 create_pie_da
	21 d
	22 dataset,"CATE
	23 } cat
	24 J
	25 ex.getMessage
	26
	27 } cat
	28 J
	29 ex.getMessage
	30
	31 }
	32 }
	33
	
	public st
	draw_pie_char
	JPanel jp){
	jp.se
	jp.getHeight(
	categ
	}

Step 6	In **Film_Utils** **jbchart_handler** responsible for passing the neces

Charts_Film for
on six different pa

The method f
passing the **Cha**
parameters. This
distribution of filr

Next, the meth
passing the **Cha**
parameters. This
distribution of filr

Then, the metho
passing the **Cha**
parameters. This
distribution of filr
After tha
draw_bar_chart_
Charts_Film obj
method draws a
language.

Next,
draw_pie_chart_
Charts_Film obj
method draws a
categorized renta

Finally,
draw_pie_chart_
Charts_Film obj
method draws a
categorized lengt

Overall, the **jb**
drawing of multi
specific chart-dr
parameters to the

```
1
2
3
4
5
6
7
8
```

```
    public st
jbchart_handl
        //Dra
by release ye
        draw_
frm.getJPanel
```

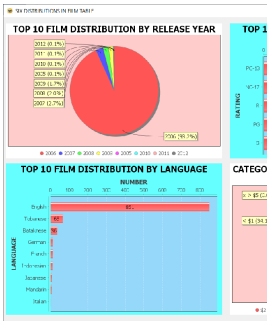
```
9           //Dra
10 by rating
11          draw_
12 frm.getJPanel
13
14          //Dra
15 by rental du
16
17 draw_pie_cha
18 frm.getJPanel
19
            //Dra
   by language

   draw_bar_cha
   frm.getJPanel

            //Dra
   by categorize

   draw_pie_cha
   frm.getJPanel

            //Dra
   by categorize

   draw_pie_cha
   frm.getJPanel
       }
```

Step 7

In **FilmForm**, d
event listener:

```
1     private vo
2  jbChartActionP
3  evt) {
4           Charts
5           frm1.s
6           frm1.s
7  TABLE");
           frm1.s
           Film_l
       }
```

This is an event
button is clicked,
class, sets its
jbchart_handler
different charts
database.

Step 8	Run the project. see the six chart 5.12.  **Figure 5.12** Th 10 film distributi duration, top 10 by categorized re
Step 9	Create a new **JFr**

Step 10	In **Charts_Film** corresponding **Va** method for the ob
	`1 //Getter`
	`2 public J`
	`3 retu`
	`4 }`

Step 11	In **Film_Utils** cl **categorized_rep** static method th **Charts_Film2**, a attempts to retr from the **get_col_val_fron LinkedHashMap** range values, and values to the **cat** the costs into **DefaultPieDatas create_pie_datas**

Finally, the m
draw_piechart_v

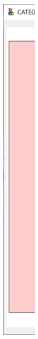
The
draw_pie_chart_
static method th
Charts_Film2 ar
JPanel, and
categorized_rep
Charts_Film2 ob

```
    private s
categorized_
frm, JPanel
        try
            L
get_col_val_t

            M
LinkedHashMap
            r
            r
            r
            r
            r
            r

            M
categorize_va

            D
create_pie_da
            d
dataset,"CATE
FILM");
        } cat
            J
ex.getMessage

        } cat
            J
ex.getMessage

        }
    }

    public st
draw_pie_char
frm, JPanel
        jp.se
jp.getHeight(
```

	`categ`
	`"film");`
	`}`

Step 12	In **Film_Utils** `jbchart_handler` **Film_Utils** class argument.
	
	The method **draw_pie_chart_ Charts_Film2** ob method draws a data based on cat
	
	`public sta`
	`jbchart_handle`
	`//Draw`
	`categorized re`
	
	`draw_pie_chart`
	`frm.getJPanel`
	`}`

Step 13	In **FilmForm**, d event listener by
	
	`private v`
	`jbChartAction`
	`evt) {`
	`Chart`
	`frm1.`
	`frm1.`
	`TABLE");`
	`frm1.`
	`Film_`
	
	`Chart`
	`frm2.`
	`frm2.`
	`DISTRIBUTIONS IN`
	`frm2.`
	`Film_`
	`}`

	The code creates sets its location, **jbchart_handler** passing the insta **jbchart_handler** **draw_pie_chart_** which retrieves categorizes the v creates a pie cha The pie chart **Charts_Film2** ins
Step 14	Run the project. see the film dis displayed on **Cha**  **Figure 5.13** The

This is the full version of **Film_Utils.java**:

```
package sakila;
import java.awt.Dimension;
import java.util.logging.Level;
import java.util.logging.Logger;
import java.sql.*;
import java.util.logging.Level;
import java.util.logging.Logger;
import java.sql.*;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashMap;
import java.util.LinkedHashMap;
import java.util.List;
import java.util.Map;
import java.util.Objects;
```

```java
import javax.swing.JComboBox;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.event.TableModelEvent;
import javax.swing.table.DefaultTableModel;
import javax.swing.table.TableModel;
import org.jfree.data.category.DefaultCategoryDataset;
import org.jfree.data.general.DefaultPieDataset;

public class Film_Utils extends Utility{
    public static final int FIRST_INDEX = 0;
    public static final int INVALID_INDEX = -1;

    private static int currentIndex = FIRST_INDEX;
    private static final String SQL_ID = Query_Film.get_sql_id();

    //Creates film table
    public static void create_film_table() {
        try (Connection conn = getConnection()) {
            Statement stmt = conn.createStatement();
            stmt.addBatch(Query_Film.get_sql_film());
            stmt.executeBatch();

            String message = String.format("Successfully creates film table");
            JOptionPane.showMessageDialog(null, message,
            "INFORMATION",JOptionPane.INFORMATION_MESSAGE);

        } catch (SQLException ex) {
            JOptionPane.showMessageDialog(null, ex.getMessage(),
                "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    //Populates film table with some rows of data
    public static void populate_film_table(){
        try(Connection conn = getConnection()){
            String sql = """
                  INSERT INTO film(film_id, title, description, release_year,
                         original_language_id, rental_duration, rental_rate,
                         replacement_cost, rating, special_features, last_upd
                  VALUES(?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)""";

            PreparedStatement ps1 = conn.prepareStatement(sql);
            //Creates a new Film class with default constructor
            Film obj1 = new Film();
            ps1.setInt(1,obj1.getFilmID());
            ps1.setString(2, obj1.getTitle());
            ps1.setString(3, obj1.getDescription());
            ps1.setInt(4, obj1.getReleaseYear());
            ps1.setInt(5, obj1.getLanguageID());
            ps1.setInt(6, obj1.getOriginalLanguageID());
            ps1.setInt(7, obj1.getRentalDuration());
            ps1.setDouble(8, obj1.getRentalRate());
```

```
            ps1.setInt(9, obj1.getLength());
            ps1.setDouble(10, obj1.getReplacementCost());
            ps1.setString(11, obj1.getRating());
            ps1.setString(12, obj1.getSpecialFeatures());
            ps1.setTimestamp(13, obj1.getLastUpdate());

            PreparedStatement ps2 = conn.prepareStatement(sql);
            //Creates a new Film class with fourteen-params constructor
            Film obj2 = new Film(2, "Batak Ethnic Movie", null, 2023, 1, 1, 3
            2.99, 150, 13.99, "G", "Trailers,Commentaries",
            new Timestamp(System.currentTimeMillis()), "English");
            ps2.setInt(1,obj2.getFilmID());
            ps2.setString(2, obj2.getTitle());
            ps2.setString(3, obj2.getDescription());
            ps2.setInt(4, obj2.getReleaseYear());
            ps2.setInt(5, obj2.getLanguageID());
            ps2.setInt(6, obj2.getOriginalLanguageID());
            ps2.setInt(7, obj2.getRentalDuration());
            ps2.setDouble(8, obj2.getRentalRate());
            ps2.setInt(9, obj2.getLength());
            ps2.setDouble(10, obj2.getReplacementCost());
            ps2.setString(11, obj2.getRating());
            ps2.setString(12, obj2.getSpecialFeatures());
            ps2.setTimestamp(13, obj2.getLastUpdate());

            ps1.executeUpdate();
            ps2.executeUpdate();

        }catch(SQLException ex){
            JOptionPane.showMessageDialog(null, ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    //Reads the content of film table
    public static void read_film_table(){
        try(Connection conn = getConnection()){
            Statement stmt = conn.createStatement();
            String sql = "SELECT * FROM film";
            ResultSet rs = stmt.executeQuery(sql);
            while(rs.next()){
                int id = rs.getInt("film_id");
                String title = rs.getString("title");
                String description = rs.getString("description");
                int year = rs.getInt("release_year");
                int lang_id = rs.getInt("language_id");
                int ori_lang_id = rs.getInt("original_language_id");
                int duration = rs.getInt("rental_duration");
                double rate = rs.getDouble("rental_rate");
                int length = rs.getInt("length");
                double cost = rs.getDouble("replacement_cost");
                String rating = rs.getString("rating");
                String features = rs.getString("special_features");
```

```
                Timestamp lu = rs.getTimestamp("last_update");

                //Creates an Film object using thirteen-params constructor
                Film obj = new Film(id, title, description, year, lang_id, or
                    duration, rate, length, cost, rating, features, lu);
                System.out.println(obj);
            }
            rs.close();
            stmt.close();

        }catch(SQLException ex){
            JOptionPane.showMessageDialog(null, ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    private static ArrayList<Film> get_film_list(FilmForm frm, String sql, St
        ArrayList<Film> list = new ArrayList<>();
        Connection conv = null;

        try(Connection conn = getConnection();
            PreparedStatement ps = conn.prepareStatement(sql)){
            if (item.equalsIgnoreCase("none")==false) {
                 ps.setString(1,item);
            }
            ResultSet rs = ps.executeQuery();

            Film obj;
            while(rs.next()){
                //Using fourteen-params constructor
                obj = new Film(rs.getInt("film_id"), rs.getString("title"),
                    rs.getString("description"), rs.getInt("release_year"),
                    rs.getInt("language_id"), rs.getInt("original_language_id
                    rs.getInt("rental_duration"), rs.getDouble("rental_rate")
                    rs.getInt("length"), rs.getDouble("replacement_cost"),
                    rs.getString("rating"), rs.getString("special_features"),
                    rs.getTimestamp("last_update"), rs.getString("language_na

                list.add(obj);
            }
        }catch (SQLException ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
                "ERROR",JOptionPane.ERROR_MESSAGE);
        }
        return list;
    }

    private static void show_table_film(FilmForm frm, ArrayList<Film> list) t
SQLException{
        DefaultTableModel model = new DefaultTableModel(0,0);

        String header[] = {"Film ID", "Title", "Description", "Release Year",
ID",
```

```
            "Original Language", "Rental Duration", "Rental Rate", "Length",
            "Replacement Cost", "Rating", "Special Features", "Language Name"
Update"};

        model.setColumnIdentifiers(set_column_header(frm.getJTFilm(), header)
        frm.getJTFilm().setModel(model);

        Object[] row = new Object[14];

        for(int i=0; i<list.size(); i++){
            row[0] = list.get(i).getFilmID();
            row[1] = list.get(i).getTitle();
            row[2] = list.get(i).getDescription();
            row[3] = list.get(i).getReleaseYear();
            row[4] = list.get(i).getLanguageID();
            row[5] = list.get(i).getOriginalLanguageID();
            row[6] = list.get(i).getRentalDuration();
            row[7] = list.get(i).getRentalRate();
            row[8] = list.get(i).getLength();
            row[9] = list.get(i).getReplacementCost();
            row[10] = list.get(i).getRating();
            row[11] = list.get(i).getSpecialFeatures();
            row[12] = list.get(i).getName();
            row[13] = list.get(i).getLastUpdate();

            model.addRow(row);
        }
    }

    public static void refresh_controls(FilmForm frm){
        frm.setLocationRelativeTo(null);
        frm.setTitle("FILM FORM");

        //Shows the content of film table and populates combobox
        try{
            //Makes alternating color for table rows
            table_renderer(frm.getJTFilm());

            //Populates table
            ArrayList<Film> list = get_film_list(frm, Query_Film.get_sql_filr
ORDER BY f.film_id", "none");
            show_table_film(frm, list);

            //Populates jcbFilmID
            String sql_id = "SELECT film_id FROM film ORDER BY film_id";
            populate_combobox(sql_id, frm.getJCBFilmID(), frm);

            //Populates getJCBTitle()
            String sql_title = "SELECT DISTINCT title FROM film ORDER BY titl
            populate_combobox(sql_title, frm.getJCBTitle(), frm);

            //Populates getJCBReleaseYear()
```

```
            String sql_year = "SELECT DISTINCT YEAR(release_year) FROM film (
release_year";
            populate_combobox(sql_year, frm.getJCBReleaseYear(), frm);

            //Populates getJCBRating()
            String sql_rating = "SELECT DISTINCT rating FROM film ORDER BY ra
            populate_combobox(sql_rating, frm.getJCBRating(), frm);

            //Populates getJCBSpecialFeatures()
            String sql_feat = "SELECT DISTINCT special_features FROM film ORD
special_features";
            populate_combobox(sql_feat, frm.getJCBSpecialFeatures(), frm);

            //Populates getJCBLanguage() and getJCBOriginalLanguage()
            String sql_lang = "SELECT name FROM language ORDER BY name";
            populate_combobox(sql_lang, frm.getJCBLanguage(), frm);
            populate_combobox(sql_lang, frm.getJCBOriginalLanguage(), frm);

        }catch (SQLException ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
            "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    //If user chooses one of items in jcbLanguage
    public static void jcbLanguage_handler(FilmForm frm){
        String item = String.valueOf(frm.getJCBLanguage().getSelectedItem());
        Object lang_id = get_val_from_database("language", "language_id", "na
        frm.getJTFLanguageID().setText(String.valueOf(lang_id));
    }

    //If user chooses one of items in jcbOriginalLanguage
    public static void jcbOriginalLanguage_handler(FilmForm frm){
        String item = String.valueOf(frm.getJCBOriginalLanguage().getSelected
        Object lang_id = get_val_from_database("language", "language_id", "na
        frm.getJTFOriginalLanguageID().setText(String.valueOf(lang_id));
    }

    //If user chooses one of items in jcbReleaseYear
    public static void jcbReleaseYear_handler(FilmForm frm){
        String item = String.valueOf(frm.getJCBReleaseYear().getSelectedItem(
        frm.getJTFReleaseYear().setText(item);
    }

    //If user chooses one of items in jcbSpecialFeatures
    public static void jcbSpecialFeatures_handler(FilmForm frm){
        String item = String.valueOf(frm.getJCBSpecialFeatures().getSelectedI
        frm.getJTFSpecialFeatures().setText(item);
    }

    private static void clear_controls(FilmForm frm){
        frm.getJTFFilmID().setText("");
        frm.getJTFTitle().setText("");
```

```
        frm.getJTFDescription().setText("");
        frm.getJTFReleaseYear().setText("");
        frm.getJTFLanguageID().setText("");
        frm.getJTFOriginalLanguageID().setText("");
        frm.getJTFRentalDuration().setText("");
        frm.getJTFRentalRate().setText("");
        frm.getJTFLength().setText("");
        frm.getJTFReplacementCost().setText("");
        frm.getJTFSpecialFeatures().setText("");
        frm.getJTFLastUpdate().setText("");
    }

   //Displays film data result row by row
    private static <T> void display_film_data(FilmForm frm, String sql, T ite
        try(Connection conn = getConnection()){
            PreparedStatement ps = conn.prepareStatement(sql);
            ps.setObject(1,item);
            ResultSet rs = ps.executeQuery();

            if (!rs.next()) {
                // no row found, clear the form fields
                clear_controls(frm);
                return;
            }

            do{
                frm.getJTFFilmID().setText(String.valueOf(rs.getInt("film_id'
                frm.getJTFTitle().setText(rs.getString("title"));
                frm.getJTFDescription().setText(rs.getString("description"));

frm.getJTFReleaseYear().setText(String.valueOf(rs.getInt("release_year")));
                frm.getJTFLanguageID().setText(String.valueOf(rs.getInt("lang

frm.getJTFOriginalLanguageID().setText(String.valueOf(rs.getInt("original_lan

frm.getJTFRentalDuration().setText(String.valueOf(rs.getInt("rental_duration'

frm.getJTFRentalRate().setText(String.valueOf(rs.getDouble("rental_rate")));
                frm.getJTFLength().setText(String.valueOf(rs.getInt("length"

frm.getJTFReplacementCost().setText(String.valueOf(rs.getDouble("replacement_
                frm.getJTFSpecialFeatures().setText(rs.getString("special_fea
                frm.getJTFLastUpdate().setText(String.valueOf(rs.getDate("las

                // Determines item selected from jcbFilmID
                find_combo_value_selected(frm.getJCBFilmID(), rs.getInt("filn

                // Determines item selected from jcbTitle
                find_combo_value_selected(frm.getJCBTitle(), rs.getString("ti

                // Determines item selected from jcbRating
                find_combo_value_selected(frm.getJCBRating(), rs.getString("r
```

```
                // Determines item selected from jcbReleaseYear
                find_combo_value_selected(frm.getJCBReleaseYear(),
rs.getInt("release_year"));

                // Determines item selected from jcbLanguage
                int lang_id = rs.getInt("language_id");
                if(lang_id != 0) {
                    Object lang_name = get_val_from_database("language", "nam
"language_id", lang_id);
                    find_combo_value_selected(frm.getJCBLanguage(), lang_name
                }

               // Determines item selected from jcbOriginalLanguage
                int ori_lang_id = rs.getInt("original_language_id");
                if(ori_lang_id != 0) {
                    Object lang_name = get_val_from_database("language", "nam
"language_id", ori_lang_id);
                    find_combo_value_selected(frm.getJCBOriginalLanguage(), l
                }

            }while(rs.next());

            rs.close();
            ps.close();
        }catch(SQLException ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    public static void jcbFilm_handler(FilmForm frm, JComboBox<String> jcb) {
        Object item = jcb.getSelectedItem();
        String sql = "";
        if (jcb.equals(frm.getJCBFilmID())) {
            sql = Query_Film.get_sql_id();
        } else if (jcb.equals(frm.getJCBTitle())) {
            sql = Query_Film.get_sql_title();
        }

        display_film_data(frm, sql, item);
    }

    public static void show_first_row(FilmForm frm){
        String item = String.valueOf(frm.getJCBFilmID().getItemAt(FIRST_INDEX
        display_film_data(frm, SQL_ID, item);
        currentIndex = FIRST_INDEX;
    }

    public static void show_last_row(FilmForm frm){
        int endIndex = frm.getJCBFilmID().getItemCount() - 1;
        String item = String.valueOf(frm.getJCBFilmID().getItemAt(endIndex));
        display_film_data(frm, SQL_ID, item);
        currentIndex = endIndex;
```

```
    }

    public static void show_prev_row(FilmForm frm){
        currentIndex--;
        if(currentIndex < FIRST_INDEX){
            currentIndex = FIRST_INDEX;
            return;
        }
        String item = String.valueOf(frm.getJCBFilmID().getItemAt(currentInde
        display_film_data(frm, SQL_ID, item);
    }

    public static void show_next_row(FilmForm frm){
        int endIndex = frm.getJCBFilmID().getItemCount() - 1;
        currentIndex++;
        if(currentIndex > endIndex){
            currentIndex = endIndex;
            return;
        }
        String item = String.valueOf(frm.getJCBFilmID().getItemAt(currentInde
        display_film_data(frm, SQL_ID, item);
    }

    public static void mouse_pressed_handler(FilmForm frm) {
        Objects.requireNonNull(frm, "frm must not be null");

        int selectedIndex = frm.getJTFilm().getSelectedRow();
        if (selectedIndex == -1) {
            JOptionPane.showMessageDialog(frm, "Please select a row to view :
                "No row selected", JOptionPane.INFORMATION_MESSAGE);
            return;
        }

        try (Connection conn = getConnection()) {
            String id =
String.valueOf(frm.getJTFilm().getModel().getValueAt(selectedIndex, 0));

            // Displays film data
            display_film_data(frm, Query_Film.get_sql_id(), id);

        } catch (SQLException ex) {
            Logger.getLogger(FilmForm.class.getName()).log(Level.SEVERE, "Err
film data", ex);
            String message = "Error displaying film data: " + ex.getMessage()
            String stackTrace = Arrays.toString(ex.getStackTrace());
            JOptionPane.showMessageDialog(frm, message + "\n\n" + stackTrace,
JOptionPane.ERROR_MESSAGE);
        }
    }

    //Updates row of data in film tabel by film_id
    public static void update_row_by_film_id(FilmForm frm, int film_id, Strin
String desc,
```

```
        int year, int lang_id, int ori_lang_id, int duration, double rate, in
double cost,
        String rating, String features) throws SQLException{
        Connection conn = getConnection();
        ResultSet rs = null;
        String query_id = "SELECT film_id FROM film WHERE film_id = ?";
        String update_query = """
            UPDATE film SET title = ?, description = ?,
                release_year = ?, language_id = ?, original_language_id = ?,
                rental_duration = ?, rental_rate = ?, length = ?, replacement
                rating = ?, special_features = ? WHERE film_id = ?""";
        try(PreparedStatement idPs = conn.prepareStatement(query_id,
                    ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_READ_ONL
        PreparedStatement updatePS = conn.prepareStatement(update_query,
                ResultSet.TYPE_SCROLL_SENSITIVE, ResultSet.CONCUR_UPDATABLE))
        {
            idPs.setInt(1,film_id);
            if(!idPs.execute()){
                String message = "Can't find film_id " + film_id;

                JOptionPane.showMessageDialog(frm, message,
                    "ERROR",JOptionPane.ERROR_MESSAGE);
            } else{
                rs = idPs.getResultSet();
                rs.next();

                //Creates an Film object using thirteen-params constructor
                Film obj = new Film(film_id, title, desc, year, lang_id, ori_
                        duration, rate, length, cost, rating, features, new
Timestamp(System.currentTimeMillis()));
                updatePS.setString(1, obj.getTitle());
                updatePS.setString(2, obj.getDescription());
                updatePS.setInt(3, obj.getReleaseYear());
                updatePS.setInt(4, obj.getLanguageID());
                updatePS.setInt(5, obj.getOriginalLanguageID());
                updatePS.setInt(6, obj.getRentalDuration());
                updatePS.setDouble(7, obj.getRentalRate());
                updatePS.setInt(8, obj.getLength());
                updatePS.setDouble(9, obj.getReplacementCost());
                updatePS.setString(10, obj.getRating());
                updatePS.setString(11, obj.getSpecialFeatures());
                updatePS.setInt(12, obj.getFilmID());

                updatePS.executeUpdate();
                rs.close();
                updatePS.close();
                idPs.close();
                conn.close();
            }
        }catch(SQLException ex){
            Logger.getLogger(FilmForm.class.getName()).log(Level.SEVERE, "Err
film data", ex);
            String message = "Error updating film data: " + ex.getMessage();
            String stackTrace = Arrays.toString(ex.getStackTrace());
```

```
            JOptionPane.showMessageDialog(null, message + "\n\n" + stackTrace
JOptionPane.ERROR_MESSAGE);
        }catch(java.lang.NumberFormatException ex){
            Logger.getLogger(FilmForm.class.getName()).log(Level.SEVERE, "Inv
ex);
            String message = "Invalid Input: " + ex.getMessage();
            String stackTrace = Arrays.toString(ex.getStackTrace());
            JOptionPane.showMessageDialog(null, message + "\n\n" + stackTrace
JOptionPane.ERROR_MESSAGE);
        }
    }

    private static HashMap<String, String> read_inputs(FilmForm frm) {
        HashMap<String, String> input_data = new HashMap<>();
        String film_id = String.valueOf(frm.getJCBFilmID().getSelectedItem())
        String title = frm.getJTFTitle().getText();
        String desc = frm.getJTFDescription().getText();
        String year =  frm.getJTFReleaseYear().getText();
        String lang_id =  frm.getJTFLanguageID().getText();
        String ori_lang_id =  frm.getJTFOriginalLanguageID().getText();
        String duration =  frm.getJTFRentalDuration().getText();
        String rate =  frm.getJTFRentalRate().getText();
        String length =  frm.getJTFLength().getText();
        String cost =  frm.getJTFReplacementCost().getText();
        String rating = String.valueOf(frm.getJCBRating().getSelectedItem());
        String features = frm.getJTFSpecialFeatures().getText();

        // Validate user input
        int film_id_int = 0;
        try {
            film_id_int = Integer.parseInt(film_id);
            if (film_id_int <= 0) {
                throw new IllegalArgumentException("Film ID cannot be negativ
            }
        } catch (NumberFormatException ex) {
            JOptionPane.showMessageDialog(frm, "Invalid Film ID: " + film_id,
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        } catch (IllegalArgumentException ex) {
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        }

        int lang_id_int = 0;
        try {
            lang_id_int = Integer.parseInt(lang_id);
            if (lang_id_int <= 0) {
                throw new IllegalArgumentException("Language ID cannot be neg
zero");
            }
        } catch (NumberFormatException ex) {
            JOptionPane.showMessageDialog(frm, "Invalid Language ID: " + lang
            "Error", JOptionPane.ERROR_MESSAGE);
```

```
            throw ex;
        } catch (IllegalArgumentException ex) {
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        }

        if (title == null || title.isEmpty()) {
            JOptionPane.showMessageDialog(frm, "Title cannot be empty",
            "Error", JOptionPane.ERROR_MESSAGE);
            throw new IllegalArgumentException("Title cannot be empty");
        }

        input_data.put("film_id", film_id);
        input_data.put("title", title);
        input_data.put("desc", desc);
        input_data.put("year", year);
        input_data.put("lang_id", lang_id);

        input_data.put("ori_lang_id", ori_lang_id);
        input_data.put("duration", duration);
        input_data.put("rate", rate);
        input_data.put("length", length);
        input_data.put("cost", cost);

        input_data.put("rating", rating);
        input_data.put("features", features);

        return input_data;
    }

    private static void edit_actual(FilmForm frm){
        try{
            HashMap<String, String> input_data = read_inputs(frm);
            int film_id = Integer.parseInt(input_data.get("film_id"));
            String title = input_data.get("title");
            String desc = input_data.get("desc");
            int year = Integer.parseInt(input_data.get("year"));
            int lang_id = Integer.parseInt(input_data.get("lang_id"));

            int ori_lang_id = Integer.parseInt(input_data.get("ori_lang_id"))
            int duration = Integer.parseInt(input_data.get("duration"));
            double rate = Double.parseDouble(input_data.get("rate"));
            int length = Integer.parseInt(input_data.get("length"));
            double cost = Double.parseDouble(input_data.get("cost"));

            String rating = input_data.get("rating");
            String features = input_data.get("features");

            update_row_by_film_id(frm, film_id, title, desc,
                    year, lang_id, ori_lang_id, duration, rate, length, cost,
                    rating, features);
```

```
            //Refreshes all objects on form
            refresh_controls(frm);

        }catch(SQLException ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    private static void enable_controls(boolean state, FilmForm frm){
        frm.getJBFirst().setEnabled(state);
        frm.getJBPrev().setEnabled(state);
        frm.getJBNext().setEnabled(state);
        frm.getJBLast().setEnabled(state);
        frm.getJBInsert().setEnabled(state);
        frm.getJBDelete().setEnabled(state);
        frm.getJBChart().setEnabled(state);
        frm.getJTFFilmID().setEnabled(state);
    }

    public static void edit_handler(FilmForm frm){
        if(frm.getJBEdit().getText().equals("EDIT")){
            frm.getJBEdit().setText("CONFIRM");

            // Disables controls
            enable_controls(false, frm);
        }

        else {
            frm.getJBEdit().setText("EDIT");

            // Actual editing
            edit_actual(frm);

            //Enables controls
            enable_controls(true, frm);
        }
    }

    public static void edit_database_from_jtable(TableModelEvent e, FilmForm
        if (e.getType() == TableModelEvent.UPDATE) {
            int row = e.getFirstRow();
            TableModel model = (TableModel)e.getSource();
            int film_id = Integer.parseInt(String.valueOf(model.getValueAt(r
            String title = String.valueOf(model.getValueAt(row, 1));
            String desc = String.valueOf(model.getValueAt(row, 2));
            int year = Integer.parseInt(String.valueOf(model.getValueAt(row,
            int lang_id = Integer.parseInt(String.valueOf(model.getValueAt(r
            int ori_lang_id = Integer.parseInt(String.valueOf(model.getValueA
            int duration = Integer.parseInt(String.valueOf(model.getValueAt(
            double rate = Double.parseDouble(String.valueOf(model.getValueAt
            int length = Integer.parseInt(String.valueOf(model.getValueAt(ro
            double cost = Double.parseDouble(String.valueOf(model.getValueAt
            String rating = String.valueOf(model.getValueAt(row, 10));
```

```
            String features = String.valueOf(model.getValueAt(row, 11));

            try{
                update_row_by_film_id(frm, film_id, title, desc,
                    year, lang_id, ori_lang_id, duration, rate, length, cost,
                    rating, features);

                //Refreshes all objects on form
                refresh_controls(frm);

            } catch (SQLIntegrityConstraintViolationException ex) {
                Logger.getLogger(FilmForm.class.getName()).log(Level.SEVERE,
entry", ex);
                JOptionPane.showMessageDialog(frm, "Error: Duplicate entry\n'
ex.getMessage());
            } catch (SQLSyntaxErrorException ex) {
                Logger.getLogger(FilmForm.class.getName()).log(Level.SEVERE,
syntax", ex);
                JOptionPane.showMessageDialog(frm, "Error: Invalid SQL syntax
ex.getMessage());
            } catch (SQLException ex) {
                Logger.getLogger(FilmForm.class.getName()).log(Level.SEVERE,
error", ex);
                JOptionPane.showMessageDialog(frm, "Error: Database error\n"
ex.getMessage());
            }
        }
    }

    //Inserts new row into film table
    private static void insert_row(FilmForm frm) throws SQLException{
        HashMap<String, String> input_data = read_inputs(frm);
        String title = input_data.get("title");
        String desc = input_data.get("desc");
        int year = Integer.parseInt(input_data.get("year"));
        int lang_id = Integer.parseInt(input_data.get("lang_id"));

        int ori_lang_id = Integer.parseInt(input_data.get("ori_lang_id"));
        int duration = Integer.parseInt(input_data.get("duration"));
        double rate = Double.parseDouble(input_data.get("rate"));
        int length = Integer.parseInt(input_data.get("length"));
        double cost = Double.parseDouble(input_data.get("cost"));

        String rating = input_data.get("rating");
        String features = input_data.get("features");

        // SQL insert statement
        String sql = """
            INSERT INTO film(title, description, release_year, language_id,
```

```
                original_language_id, rental_duration, rental_rate,
length,
                replacement_cost, rating, special_features)
                VALUES(?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)""";

        try(Connection conn = getConnection();
         PreparedStatement pstmt = conn.prepareStatement(sql)){
            //Creates an Film object using twelve-params constructor
            Film obj = new Film(title, desc, year, lang_id,
ori_lang_id,
                duration, rate, length, cost, rating, features, new
Timestamp(System.currentTimeMillis()));
            pstmt.setString(1, obj.getTitle());
            pstmt.setString(2, obj.getDescription());
            pstmt.setInt(3, obj.getReleaseYear());
            pstmt.setInt(4, obj.getLanguageID());
            pstmt.setInt(5, obj.getOriginalLanguageID());
            pstmt.setInt(6, obj.getRentalDuration());
            pstmt.setDouble(7, obj.getRentalRate());
            pstmt.setInt(8, obj.getLength());
            pstmt.setDouble(9, obj.getReplacementCost());
            pstmt.setString(10, obj.getRating());
            pstmt.setString(11, obj.getSpecialFeatures());

            //Executes the sql insert statement
            pstmt.executeUpdate();
        } catch (SQLException ex) {

Logger.getLogger(FilmForm.class.getName()).log(Level.SEVERE, "Database
error", ex);
            JOptionPane.showMessageDialog(frm, "Error: Database
error\n" + ex.getMessage());
        }
    }

    private static void insert_actual(FilmForm frm){
        try{
            insert_row(frm);

            //Refreshes table and comboboxes
            refresh_controls(frm);

        }catch(SQLException ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    public static void insert_handler(FilmForm frm){
        if(frm.getJBInsert().getText().equals("INSERT")){
            frm.getJBInsert().setText("CONFIRM");

            //Disables jbEdit
```

```
            frm.getJBEdit().setEnabled(false);

            // Disables controls
            enable_controls(false, frm);
            frm.getJCBFilmID().setEnabled(false);
            frm.getJCBTitle().setEnabled(false);

            // Clears controls
            clear_controls(frm);

            // Enables
            frm.getJBInsert().setEnabled(true);
        }

        else {
            frm.getJBInsert().setText("INSERT");

            // Actual insertion
            insert_actual(frm);

            //Enables jbEdit
            frm.getJBEdit().setEnabled(true);

            //Enables controls
            enable_controls(true, frm);
            frm.getJCBFilmID().setEnabled(true);
            frm.getJCBTitle().setEnabled(true);
        }
    }

    public static void delete_handler(FilmForm frm){
        int dialogButton = JOptionPane.YES_NO_OPTION;
        int film_id =
Integer.parseInt(String.valueOf(frm.getJCBFilmID().getSelectedItem()));

        String message = String.format("Are you sure you want to delete
the row Film ID: %d)", film_id);
        int answer = JOptionPane.showConfirmDialog(frm, message,
"DELETING ROW OF DATA", dialogButton);

        if(answer == JOptionPane.YES_OPTION){
            String query = """
                DELETE FROM film WHERE film_id = ?""";
            try(Connection conn = getConnection();
                PreparedStatement ps = conn.prepareStatement(query)){
                // Use PreparedStatement to avoid SQL injection attacks
                ps.setInt(1, film_id);
                ps.executeUpdate();

                // Refresh table and comboboxes
                refresh_controls(frm);

            } catch (SQLException ex){
```

```
                JOptionPane.showMessageDialog(frm, ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
            }
        }
    }

    private static void draw_pie_chart_film_by_year(Charts_Film frm,
JPanel jp){
        jp.setPreferredSize(new Dimension(jp.getWidth(),
jp.getHeight()));
        DefaultPieDataset dataset =
create_pie_dataset(Query_Film.get_sql_film_year_dist(), "Number",
"year");

        //Draws piechart film distribution by release year
        draw_piechart_with_dataset(frm, jp, dataset, "TOP 10 FILM
DISTRIBUTION BY RELEASE YEAR");
    }

    private static void draw_bar_chart_film_by_rating(Charts_Film frm,
JPanel jp){
        jp.setPreferredSize(new Dimension(jp.getWidth(),
jp.getHeight()));

        DefaultCategoryDataset dataset =
create_bar_dataset(Query_Film.get_sql_film_rating_dist(), "Number",
"rating");

        //Draws barchart film distribution by rating
        draw_barchart_with_dataset(frm, jp, dataset, "TOP 10 FILM
DISTRIBUTION BY RATING", "RATING", "NUMBER");
    }

    private static void draw_pie_chart_film_by_duration(Charts_Film
frm, JPanel jp){
        jp.setPreferredSize(new Dimension(jp.getWidth(),
jp.getHeight()));
        DefaultPieDataset dataset =
create_pie_dataset(Query_Film.get_sql_film_duration_dist(), "Number",
"duration");

        //Draws piechart film distribution by rental duration
        draw_piechart_with_dataset(frm, jp, dataset, "TOP 10 FILM
DISTRIBUTION BY RENTAL DURATION");
    }

    private static void draw_bar_chart_film_by_language(Charts_Film
frm, JPanel jp){
        jp.setPreferredSize(new Dimension(jp.getWidth(),
jp.getHeight()));

        DefaultCategoryDataset dataset =
create_bar_dataset(Query_Film.get_sql_film_lang_dist(), "Number",
```

```
"l1.name");

        //Draws barchart film distribution by language
        draw_barchart_with_dataset(frm, jp, dataset, "TOP 10 FILM
DISTRIBUTION BY LANGUAGE", "LANGUAGE", "NUMBER");
    }

    private static void categorized_rental_rate_piedataset(Charts_Film
frm, JPanel jp, String table) {
        try {
            List<Number> rates =
get_col_val_from_database("rental_rate", table);

            Map<String, Double> range_vals = new LinkedHashMap<>();
            range_vals.put("< $1", 1.0);
            range_vals.put("$1 < x < $2",2.0);
            range_vals.put("$2 < x < $3", 3.0);
            range_vals.put("$3 < x < $5", 5.0);
            range_vals.put("x > $5", Double.MAX_VALUE);

            Map<String, Double> rate_counts = categorize_values(rates,
range_vals);

            DefaultPieDataset dataset =
create_pie_dataset_from_categorized_vals(rate_counts);
            draw_piechart_with_dataset(frm, jp, dataset,"CATEGORIZED
RENTAL RATE DISTRIBUTION IN FILM");
        } catch (SQLException ex) {
            JOptionPane.showMessageDialog(null, ex.getMessage(),
                    "ERROR", JOptionPane.ERROR_MESSAGE);
        } catch (ClassNotFoundException ex) {
            JOptionPane.showMessageDialog(null, ex.getMessage(),
                    "ERROR", JOptionPane.ERROR_MESSAGE);
        }
    }

    public static void
draw_pie_chart_categorized_rental_rate(Charts_Film frm, JPanel jp){
        jp.setPreferredSize(new Dimension(jp.getWidth(),
jp.getHeight()));
        categorized_rental_rate_piedataset(frm, jp, "film");
    }

    private static void categorized_length_piedataset(Charts_Film frm,
JPanel jp, String table) {
        try {
            List<Number> lengths = get_col_val_from_database("length",
table);

            Map<String, Double> range_vals = new LinkedHashMap<>();
            range_vals.put("< 100", 100.0);
            range_vals.put("100 < x < 125",125.0);
            range_vals.put("125 < x < 150", 150.0);
```

```
            range_vals.put("150 < x < 200", 200.0);
            range_vals.put("x > 200", Double.MAX_VALUE);

            Map<String, Double> length_counts =
categorize_values(lengths, range_vals);

            DefaultPieDataset dataset =
create_pie_dataset_from_categorized_vals(length_counts);
            draw_piechart_with_dataset(frm, jp, dataset,"CATEGORIZED
LENGTH DISTRIBUTION IN FILM");
        } catch (SQLException ex) {
            JOptionPane.showMessageDialog(null, ex.getMessage(),
                    "ERROR", JOptionPane.ERROR_MESSAGE);
        } catch (ClassNotFoundException ex) {
            JOptionPane.showMessageDialog(null, ex.getMessage(),
                    "ERROR", JOptionPane.ERROR_MESSAGE);
        }
    }

    public static void draw_pie_chart_categorized_length(Charts_Film
frm, JPanel jp){
        jp.setPreferredSize(new Dimension(jp.getWidth(),
jp.getHeight()));
        categorized_length_piedataset(frm, jp, "film");
    }

    public static void jbchart_handler(Charts_Film frm){
        //Draws piechart film distribution by release year
        draw_pie_chart_film_by_year(frm, frm.getJPanel1());

        //Draws barchart film distribution by rating
        draw_bar_chart_film_by_rating(frm, frm.getJPanel2());

        //Draws piechart film distribution by rental duration
        draw_pie_chart_film_by_duration(frm, frm.getJPanel3());

        //Draws barchart film distribution by language
        draw_bar_chart_film_by_language(frm, frm.getJPanel4());

        //Draws piechart film distribution by categorized rental rate
        draw_pie_chart_categorized_rental_rate(frm, frm.getJPanel5());

        //Draws piechart film distribution by categorized length
        draw_pie_chart_categorized_length(frm, frm.getJPanel6());
    }

    private static void
categorized_replacement_cost_piedataset(Charts_Film2 frm, JPanel jp,
String table) {
        try {
            List<Number> costs =
get_col_val_from_database("replacement_cost", table);
```

```
            Map<String, Double> range_vals = new LinkedHashMap<>();
            range_vals.put("< 10$", 10.0);
            range_vals.put("10$ < x < 15$",15.0);
            range_vals.put("15$ < x < 20$", 20.0);
            range_vals.put("20$ < x < 30$", 30.0);
            range_vals.put("30$ < x < 40$", 40.0);
            range_vals.put("x > 40$", Double.MAX_VALUE);

            Map<String, Double> cost_counts = categorize_values(costs,
range_vals);

            DefaultPieDataset dataset =
create_pie_dataset_from_categorized_vals(cost_counts);
            draw_piechart_with_dataset(frm, jp, dataset,"CATEGORIZED
REPLACEMENT COST DISTRIBUTION IN FILM");
        } catch (SQLException ex) {
            JOptionPane.showMessageDialog(null, ex.getMessage(),
                    "ERROR", JOptionPane.ERROR_MESSAGE);
        } catch (ClassNotFoundException ex) {
            JOptionPane.showMessageDialog(null, ex.getMessage(),
                    "ERROR", JOptionPane.ERROR_MESSAGE);
        }
    }

    public static void
draw_pie_chart_categorized_replacement_cost(Charts_Film2 frm, JPanel
jp){
        jp.setPreferredSize(new Dimension(jp.getWidth(),
jp.getHeight()));
        categorized_replacement_cost_piedataset(frm, jp, "film");
    }

    public static void jbchart_handler2(Charts_Film2 frm){
        //Draws piechart film distribution by categorized replacement
cost
        draw_pie_chart_categorized_replacement_cost(frm,
frm.getJPanel1());
    }
}
```

This is the full version of **FilmForm.java**:

```
package sakila;
import java.awt.Toolkit;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.*;
import javax.swing.event.TableModelEvent;
```

```
import javax.swing.event.TableModelListener;

public class FilmForm extends javax.swing.JFrame {

    public FilmForm() {
        initComponents();
        Utility.setLookAndFeel(this);
        Film_Utils.refresh_controls(this);

 this.setIconImage(Toolkit.getDefaultToolkit().getImage(getClass().getResourc
;
        this.setDefaultCloseOperation(this.HIDE_ON_CLOSE);
    }

    //Getter methods for JTextField variable instances
    public JTextField getJTFFilmID(){return this.jtfFilmID;}
    public JTextField getJTFTitle(){return this.jtfTitle;}
    public JTextField getJTFDescription(){return this.jtfDescription;}
    public JTextField getJTFLastUpdate(){return this.jtfLastUpdate;}
    public JTextField getJTFLanguageID(){return this.jtfLanguageID;}
    public JTextField getJTFReleaseYear(){return this.jtfReleaseYear;}
    public JTextField getJTFRentalDuration(){return this.jtfRentalDuration;}
    public JTextField getJTFOriginalLanguageID(){return this.jtfOriginalLangu
    public JTextField getJTFRentalRate(){return this.jtfRentalRate;}
    public JTextField getJTFLength(){return this.jtfLength;}
    public JTextField getJTFReplacementCost(){return this.jtfReplacementCost
    public JTextField getJTFSpecialFeatures(){return this.jtfSpecialFeatures

    //Getter methods for JComboBox variable instances
    public JComboBox getJCBFilmID(){return this.jcbFilmID;}
    public JComboBox getJCBLanguage(){return this.jcbLanguage;}
    public JComboBox getJCBTitle(){return this.jcbTitle;}
    public JComboBox getJCBReleaseYear(){return this.jcbReleaseYear;}
    public JComboBox getJCBOriginalLanguage(){return this.jcbOriginalLanguage
    public JComboBox getJCBRating(){return this.jcbRating;}
    public JComboBox getJCBSpecialFeatures(){return this.jcbSpecialFeatures;}

    //Getter methods for JTable variable instance
    public JTable getJTFilm(){return this.jtFilm;}

    //Getter methods for JButton variable instances
    public JButton getJBEdit(){return this.jbEdit;}
    public JButton getJBInsert(){return this.jbInsert;}
    public JButton getJBDelete(){return this.jbDelete;}
    public JButton getJBChart(){return this.jbChart;}
    public JButton getJBFirst(){return this.jbFirst;}
    public JButton getJBPrev(){return this.jbPrev;}
    public JButton getJBNext(){return this.jbNext;}
    public JButton getJBLast(){return this.jbLast;}

    @SuppressWarnings("unchecked")
    private void initComponents() {
        //...
```

```
        pack();
    }// </editor-fold>

    private void jcbLanguageActionPerformed(java.awt.event.ActionEvent evt) {
        Film_Utils.jcbLanguage_handler(this);
    }

    private void jcbOriginalLanguageActionPerformed(java.awt.event.ActionEver
        Film_Utils.jcbOriginalLanguage_handler(this);
    }

    private void jbFirstActionPerformed(java.awt.event.ActionEvent evt) {
       Film_Utils.show_first_row(this);
    }

    private void jbLastActionPerformed(java.awt.event.ActionEvent evt) {
        Film_Utils.show_last_row(this);
    }

    private void jbPrevActionPerformed(java.awt.event.ActionEvent evt) {
        Film_Utils.show_prev_row(this);
    }

    private void jbNextActionPerformed(java.awt.event.ActionEvent evt) {
        Film_Utils.show_next_row(this);
    }

    private void jcbFilmIDActionPerformed(java.awt.event.ActionEvent evt) {
        Film_Utils.jcbFilm_handler(this, this.jcbFilmID);
    }

    private void jcbTitleActionPerformed(java.awt.event.ActionEvent evt) {
        Film_Utils.jcbFilm_handler(this, this.jcbTitle);
    }

    private void jtFilmMousePressed(java.awt.event.MouseEvent evt) {
        Film_Utils.mouse_pressed_handler(this);
    }

    private void jbChartActionPerformed(java.awt.event.ActionEvent evt) {
        Charts_Film frm1 = new Charts_Film();
        frm1.setLocationRelativeTo(null);
        frm1.setTitle("SIX DISTRIBUTIONS IN FILM TABLE");
        frm1.setVisible(true);
        Film_Utils.jbchart_handler(frm1);

        Charts_Film2 frm2 = new Charts_Film2();
        frm2.setLocationRelativeTo(null);
        frm2.setTitle("CATEGORIZIED REPLACEMENT COST DISTRIBUTIONS IN FILM TA
        frm2.setVisible(true);
        Film_Utils.jbchart_handler2(frm2);
    }
```

```
    private void jbDeleteActionPerformed(java.awt.event.ActionEvent evt) {
        Film_Utils.delete_handler(this);
    }

    private void jbInsertActionPerformed(java.awt.event.ActionEvent evt) {
        Film_Utils.insert_handler(this);
    }

    private void jbEditActionPerformed(java.awt.event.ActionEvent evt) {
        Film_Utils.edit_handler(this);
    }

    private void jcbSpecialFeaturesActionPerformed(java.awt.event.ActionEvent
        Film_Utils.jcbSpecialFeatures_handler(this);
    }

    private void jcbReleaseYearActionPerformed(java.awt.event.ActionEvent evt
        Film_Utils.jcbReleaseYear_handler(this);
    }

    private void jtFilmMouseClicked(java.awt.event.MouseEvent evt) {
        // instantiate FilmTableModelListener and add it as a listener to the
        FilmTableModelListener tableModelListener = new FilmTableModelListene
this);
        this.getJTFilm().getModel().addTableModelListener(tableModelListener)
    }

    public static void main(String args[]) {
        try {
            for (javax.swing.UIManager.LookAndFeelInfo info :
javax.swing.UIManager.getInstalledLookAndFeels()) {
                if ("Nimbus".equals(info.getName())) {
                    javax.swing.UIManager.setLookAndFeel(info.getClassName())
                    break;
                }
            }
        } catch (ClassNotFoundException ex) {

java.util.logging.Logger.getLogger(FilmForm.class.getName()).log(java.util.lo
null, ex);
        } catch (InstantiationException ex) {

java.util.logging.Logger.getLogger(FilmForm.class.getName()).log(java.util.lo
null, ex);
        } catch (IllegalAccessException ex) {

java.util.logging.Logger.getLogger(FilmForm.class.getName()).log(java.util.lo
null, ex);
        } catch (javax.swing.UnsupportedLookAndFeelException ex) {

java.util.logging.Logger.getLogger(FilmForm.class.getName()).log(java.util.lo
null, ex);
```

```
        }

        /* Create and display the form */
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new FilmForm().setVisible(true);
            }
        });
    }

    // Variables declaration - do not modify
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel10;
    private javax.swing.JLabel jLabel11;
    private javax.swing.JLabel jLabel12;
    private javax.swing.JLabel jLabel13;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JLabel jLabel3;
    private javax.swing.JLabel jLabel4;
    private javax.swing.JLabel jLabel5;
    private javax.swing.JLabel jLabel6;
    private javax.swing.JLabel jLabel7;
    private javax.swing.JLabel jLabel8;
    private javax.swing.JLabel jLabel9;
    private javax.swing.JScrollPane jScrollPane1;
    private javax.swing.JButton jbChart;
    private javax.swing.JButton jbDelete;
    private javax.swing.JButton jbEdit;
    private javax.swing.JButton jbFirst;
    private javax.swing.JButton jbInsert;
    private javax.swing.JButton jbLast;
    private javax.swing.JButton jbNext;
    private javax.swing.JButton jbPrev;
    private javax.swing.JComboBox<String> jcbFilmID;
    private javax.swing.JComboBox<String> jcbLanguage;
    private javax.swing.JComboBox<String> jcbOriginalLanguage;
    private javax.swing.JComboBox<String> jcbRating;
    private javax.swing.JComboBox<String> jcbReleaseYear;
    private javax.swing.JComboBox<String> jcbSpecialFeatures;
    private javax.swing.JComboBox<String> jcbTitle;
    private javax.swing.JTable jtFilm;
    private javax.swing.JTextField jtfDescription;
    private javax.swing.JTextField jtfFilmID;
    private javax.swing.JTextField jtfLanguageID;
    private javax.swing.JTextField jtfLastUpdate;
    private javax.swing.JTextField jtfLength;
    private javax.swing.JTextField jtfOriginalLanguageID;
    private javax.swing.JTextField jtfReleaseYear;
    private javax.swing.JTextField jtfRentalDuration;
    private javax.swing.JTextField jtfRentalRate;
    private javax.swing.JTextField jtfReplacementCost;
    private javax.swing.JTextField jtfSpecialFeatures;
    private javax.swing.JTextField jtfTitle;
```

```
    // End of variables declaration
}
```

FILM ACTOR FORM

FILM ACTOR FORM

CREATING AND POPULATING FILM_ACTOR TABLE

CREATING AND POPULATING FILM_ACTOR TABLE

Step 1	Create a new class named **Query_FilmActor**. It contains several static SQL queries. Here is a brief explanation of each query: 1. **sql_min**: This query selects the minimum value of **film_id** from the **film** table. 2. **sql_max**: This query selects the maximum value of **film_id** from the **film** table. 3. **sql_id**: This query selects all columns from the **film** table where the **film_id** matches a specified parameter. 4. **sql_title**: This query selects all columns from the **film** table where the **title** matches a specified parameter. 5. **sql_film_rating_dist**: This query calculates the average rental rate, replacement cost, and

length for each film rating in the film table, and groups the results by rating.

6. **sql_film_actor_dist**: This query calculates the average rental rate, replacement cost, and length for each actor in the film_actor table, and groups the results by actor. It also counts the number of films each actor has acted in and orders the results by the count in descending order, then by actor name in ascending order. It only returns the top 10 results.
7. **sql_film_actor_joint**: This query joins the **film_actor**, **film**, **actor**, and **language** tables and selects several columns from each table, including the film_id, actor_id, and last_update columns from the film_actor table. It also selects the title, description, release_year, language_id, original_language_id, rental_duration, rental_rate, length, replacement_cost, rating, and special_features columns from the film table, the name column from the language table, and the concatenated first_name and last_name columns from the actor table. It does not have a WHERE clause and returns all results.
8. **sql_film_actor**: This query creates a new table called **film_actor** with three columns: **actor_id**, **film_id**, and **last_update**. It also defines primary and foreign key constraints and sets the engine and charset for the table.

The class also contains several getter methods that return the SQL queries as strings.

```
1  package sakila;
2
3  public class Query_FilmActor {
4      private static final String
5  sql_min = "SELECT MIN(film_id) FROM
6  film";
7      private static final String
8  sql_max = "SELECT MAX(film_id) FROM
9  film";
10     private static final String
11 sql_id = "SELECT * FROM film WHERE
12 film_id  = ?";
13     private static final String
14 sql_title = "SELECT * FROM film
15 WHERE title = ?";
```

```
16
17      private static final String
18  sql_film_rating_dist = """
19          SELECT rating,
20  AVG(rental_rate) AS
21  avg_rental_rate,
22          AVG(replacement_cost) AS
23  avg_replacement_cost,
24          AVG(length) AS avg_length,
25          COUNT(*) AS Number
26          FROM film
27          GROUP BY rating""";
28
29      private static final String
30  sql_film_actor_dist = """
31          SELECT
32  CONCAT(a.first_name,'
33  ',a.last_name) AS actor_name,
34          AVG(f.rental_rate) AS
35  avg_rental_rate,
36          AVG(f.replacement_cost) AS
37  avg_replacement_cost,
38          AVG(f.length) AS
39  avg_length,
40          COUNT(*) AS Number
41          FROM film_actor fa
42          JOIN film f ON f.film_id =
43  fa.film_id
44          JOIN actor a ON a.actor_id
45  = fa.actor_id
46          GROUP BY actor_name
47          ORDER BY Count(*) DESC,
48  actor_name ASC
49          LIMIT 10""";
50
51      private static final String
52  sql_film_actor_joint = """
53          SELECT fa.film_id,
54  fa.actor_id, fa.last_update,
55              f.title, f.description,
56  f.release_year,
57              f.language_id,
58  f.original_language_id,
59  f.rental_duration,
60              f.rental_rate,
61  f.length, f.replacement_cost,
62  f.rating,
63              f.special_features,
64  l.name AS language_name,
65              CONCAT(a.first_name,'
66  ',a.last_name) AS actor_name
67          FROM film_actor fa
68
69
```

```
        JOIN film f ON f.film_id =
fa.film_id
        JOIN actor a ON a.actor_id
= fa.actor_id
        JOIN language l ON
l.language_id = f.language_id""";

    private static final String
sql_film_actor = """
        CREATE TABLE film_actor (
            actor_id SMALLINT
UNSIGNED NOT NULL,
            film_id SMALLINT
UNSIGNED NOT NULL,
            last_update TIMESTAMP
NOT NULL DEFAULT CURRENT_TIMESTAMP
ON UPDATE CURRENT_TIMESTAMP,
            PRIMARY KEY
(actor_id,film_id),
            KEY idx_fk_film_id
(`film_id`),
            CONSTRAINT
fk_film_actor_actor FOREIGN KEY
(actor_id) REFERENCES actor
(actor_id) ON DELETE RESTRICT ON
UPDATE CASCADE,
            CONSTRAINT
fk_film_actor_film FOREIGN KEY
(film_id) REFERENCES film (film_id)
ON DELETE RESTRICT ON UPDATE
CASCADE
        ) ENGINE=InnoDB DEFAULT
CHARSET=utf8mb4;""";

    //Getter methods
    public static String
get_sql_min() {
        return sql_min;
    }

    public static String
get_sql_max() {
        return sql_max;
    }

    public static String
get_sql_id() {
        return sql_id;
    }

    public static String
get_sql_title() {
        return sql_title;
```

```
    }

    public static String
get_sql_film_actor() {
        return sql_film_actor;
    }

    public static String
get_sql_film_actor_joint() {
        return
sql_film_actor_joint;
    }

    public static String
get_sql_film_actor_dist() {
        return sql_film_actor_dist;
    }

    public static String
get_sql_film_rating_dist() {
        return
sql_film_rating_dist;
    }
}
```

Step 2

Then, create a public class named **FilmActor** with instance variables and getter/setter methods. The class has 16 instance variables, including integer and double values, strings, and a timestamp. It also has a default constructor, a three-parameter constructor, and a sixteen-parameter constructor, each of which initializes some or all of the instance variables.

The class also has getter and setter methods for each of the instance variables, as well as an overridden **hashCode()** method and an overridden **equals()** method for object comparison.

Finally, the class has a **toString()** method that returns a formatted string containing the values of all instance variables. This method can be used to print a summary of the **FilmActor** object.

```
1  package sakila;
2  import java.util.Objects;
3  import java.sql.Timestamp;
4
5  public class FilmActor {
6      //16 Instance Variables
```

```
7       private int actor_id;
8       private int film_id;
9       private Timestamp last_update;
10
11      private String title;
12      private String description;
13      private int release_year;
14      private int language_id;
15      private int original_language_id;
16      private int rental_duration;
17      private double rental_rate;
18      private int length;
19      private double replacement_cost;
20      private String rating;
21      private String special_features;
22
23      private String lang_name;
24      private String actor_name;
25
26      //Default constructor
27      FilmActor(){
28          this(1, 1, new
29  Timestamp(System.currentTimeMillis()));
30      }
31
32      //Three-params constructor
33      FilmActor(int act_id, int film_id,
34  Timestamp lu){
35          setActorID(act_id);
36          setFilmID(film_id);
37          setLastUpdate(lu);
38      }
39
40      //Sixteen-params constructor
41      FilmActor(int act_id, int film_id,
42  String title, String description, int
43  year, int lang_id, int ori_lang_id,
44              int duration, double
45  rental_rate, int length, double cost,
46  String rating,
47              String features, String
48  lang_name, String actor_name, Timestamp
49  lu){
50          this(act_id, film_id, lu);
51          this.title = title;
52          this.description = description;
53          this.release_year = year;
54          this.language_id = lang_id;
55          this.original_language_id =
56  ori_lang_id;
57          this.rental_duration =
58  duration;
59          this.rental_rate = rental_rate;
60
```

```
61          this.length = length;
62          this.replacement_cost = cost;
63          this.rating = rating;
64          this.special_features =
65      features;
66          this.lang_name = lang_name;
67          this.actor_name = actor_name;
68      }
69
70          // Getter methods
71          public int getActorID() {return
72      actor_id;}
73          public int getFilmID() {return
74      film_id;}
75          public Timestamp getLastUpdate()
76      {return last_update;}
77
78          public String getTitle() {return
79      title;}
80          public String getDescription()
81      {return description;}
82          public int getReleaseYear() {return
83      release_year;}
84          public int getLanguageID() {return
85      language_id;}
86          public int getOriginalLanguageID()
87      {return original_language_id;}
88          public int getRentalDuration()
89      {return rental_duration;}
90          public double getRentalRate()
91      {return rental_rate;}
92          public int getLength() {return
93      length;}
94          public double getReplacementCost()
95      {return replacement_cost;}
96          public String getRating() {return
97      rating;}
98          public String getSpecialFeatures()
99      {return special_features;}
100         public String getLanguageName()
101     {return lang_name;}
102         public String getActorName()
103     {return actor_name;}
104
105         //Setter methods
106         public void setActorID(int id) {
107             if (id <= 0) {
108                 throw new
109     IllegalArgumentException("Actor ID must
110     be greater than zero.");
111             }
112             this.actor_id = id;
113         }
114
```

```
115         public void setFilmID(int id) {
116             if (id <= 0) {
117                 throw new
118     IllegalArgumentException("Film ID must
119     be greater than zero.");
120             }
121             this.film_id = id;
122         }
123 
124         public void setLastUpdate(Timestamp
125     date){
126127             if (date == null) {
128             throw new
129     IllegalArgumentException("Date cannot
130     be null");
131             }
132             this.last_update = date;
133         }
134 
135         // Override the hashCode() method
136         @Override
137         public int hashCode() {
138             return Objects.hash(film_id,
139     actor_id, last_update);
        }

        // Override the equals() method
        @Override
        public boolean equals(Object o) {
            if (this == o) return true;
            if (o == null || getClass() !=
    o.getClass()) return false;
            FilmActor fa = (FilmActor) o;
            return film_id == fa.film_id &&
                   actor_id == fa.actor_id
    &&

    Objects.equals(last_update,
    fa.last_update);
        }

        @Override
        public String toString(){
            return  "\nFilm ID
    : " + this.getFilmID() +
                    "\nActor ID
    : " + this.getActorID() +
                    "\nActor Name
    : " + this.getActorName() +
                    "\nTitle
    : " + this.getTitle() +
                    "\nDescription
    : " + this.getDescription() +
```

```
                "\nRelease Year
: " + this.getReleaseYear() +
                "\nLanguage ID
: " + this.getLanguageID() +
                "\nLanguage Name
: " + this.getLanguageName() +
                "\nOriginal Language ID
: " + this.getOriginalLanguageID() +
                "\nRental Duration
: " + this.getRentalDuration()+
                "\nRental Rate
: " + this.getRentalRate()+
                "\nLength
: " + this.getLength()+
                "\nReplacement Cose
: " + this.getReplacementCost()+
                "\nRating
: " + this.getRating()+
                "\nSpecial Features
: " + this.getSpecialFeatures()+
                "\nLast Update
: " + this.getLastUpdate();
    }
}
```

Step 3

Create a new public class named **FilmActor_Utils**. It contains several methods for managing **film_actor** data in Sakila database. Here's an explanation of each method:

1. **create_film_actor_table()**: This method creates a new table named **film_actor** in the Sakila database. It uses a SQL script defined in **Query_FilmActor.get_sql_film_actor()** to define the structure of the table, and executes the script using a **Statement** object. If the execution is successful, a message dialog will pop up showing that the table was created. Otherwise, an error dialog will pop up showing the error message.
2. **populate_film_actor_table()**: This method populates the film_actor table with two rows of data. It creates two **FilmActor** objects (one with the default constructor, and another with a three-parameter constructor), and inserts the data of each object into the table using a prepared statement. If the execution is

successful, the table will have two new rows of data.

3. **read_film_actor_table()**: This method reads the content of the **film_actor**, **actor**, **film**, and **language** tables using a SQL script defined in **Query_FilmActor.get_sql_film_actor_joint()**. It executes the script using a Statement object, and retrieves the data using a **ResultSet** object. For each row in the result set, it creates a **FilmActor** object using a sixteen-parameter constructor, and prints the object to the console. The object contains data from all three tables, joined using SQL JOIN clauses. If the execution is successful, the console will display the contents of the **FilmActor** objects. If not, an error dialog will pop up showing the error message.

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
```

```
28  package sakila;
29  import java.awt.Dimension;
30  import java.util.logging.Level;
31  import java.util.logging.Logger;
32  import java.sql.*;
33  import java.util.logging.Level;
34  import java.util.logging.Logger;
35  import java.sql.*;
36  import java.util.ArrayList;
37  import java.util.Arrays;
38  import java.util.HashMap;
39  import java.util.LinkedHashMap;
40  import java.util.List;
41  import java.util.Map;
42  import java.util.Objects;
43  import javax.swing.JComboBox;
44  import javax.swing.JOptionPane;
45  import javax.swing.JPanel;
46  import javax.swing.event.TableModelEvent;
47  import javax.swing.table.DefaultTableModel;
48  import javax.swing.table.TableModel;
49  import org.jfree.data.category.DefaultCategoryDataset;
50  import org.jfree.data.general.DefaultPieDataset;
51
52  public class FilmActor_Utils extends Utility{
53          public static final int FIRST_INDEX = 0;
54      public static final int INVALID_INDEX = -1;
55
56      private static int currentIndex = FIRST_INDEX;
57      private static final String SQL_ID =
58  Query_FilmActor.get_sql_film_actor_joint() + " WHERE
59  fa.film_id = ?";
60
61      //Creates film_actor table
62      public static void create_film_actor_table() {
63          try (Connection conn = getConnection()) {
64              Statement stmt = conn.createStatement();
65
66  stmt.addBatch(Query_FilmActor.get_sql_film_actor());
67              stmt.executeBatch();
68
69              String message = String.format("Successfully creates film_actor
70  table");
71              JOptionPane.showMessageDialog(null, message,
72              "INFORMATION",JOptionPane.INFORMATION_MESSAGE);
73
74          } catch (SQLException ex) {
75              JOptionPane.showMessageDialog(null,
76  ex.getMessage(),
77                      "ERROR",JOptionPane.ERROR_MESSAGE);
78          }
79      }
80
```

```
81
82      //Populates film_actor table with some rows of data
83      public static void populate_film_actor_table(){
84          try(Connection conn = getConnection()){
85              String sql = """
86                      INSERT INTO film_actor(actor_id, film_id,
87  last_update)
88                      VALUES(?, ?, ?)""";
89
90              PreparedStatement ps1 =
91  conn.prepareStatement(sql);
92              //Creates a new FilmActor class with default
93  constructor
94              FilmActor obj1 = new FilmActor();
95              ps1.setInt(1,obj1.getActorID());
96              ps1.setInt(2, obj1.getFilmID());
97              ps1.setTimestamp(3, obj1.getLastUpdate());
98
99              PreparedStatement ps2 =
100 conn.prepareStatement(sql);
101             //Creates a new FilmActor class with three-params
102 constructor
103             FilmActor obj2 = new FilmActor(2, 2, new
104 Timestamp(System.currentTimeMillis()));
105             ps2.setInt(1,obj2.getActorID());
106             ps2.setInt(2, obj2.getFilmID());
107             ps2.setTimestamp(3, obj2.getLastUpdate());
108
109             ps1.executeUpdate();
110             ps2.executeUpdate();
111
112         }catch(SQLException ex){
113             JOptionPane.showMessageDialog(null,
114 ex.getMessage(),
115                     "ERROR",JOptionPane.ERROR_MESSAGE);
116         }
117     }
118
119     //Reads the content of joined film_actor, film, and
    language tables
    public static void read_film_actor_table(){
        try(Connection conn = getConnection()){
            Statement stmt = conn.createStatement();
            ResultSet rs =
stmt.executeQuery(Query_FilmActor.get_sql_film_actor_joint());
            while(rs.next()){
                int actor_id = rs.getInt("actor_id");
                int film_id = rs.getInt("film_id");
                String title = rs.getString("title");
                String description =
rs.getString("description");
                int year = rs.getInt("release_year");
                int lang_id = rs.getInt("language_id");
```

```
                int ori_lang_id =
rs.getInt("original_language_id");
                int duration = rs.getInt("rental_duration");
                double rate = rs.getDouble("rental_rate");
                int length = rs.getInt("length");
                double cost =
rs.getDouble("replacement_cost");
                String rating = rs.getString("rating");
                String features =
rs.getString("special_features");
                String lang_name =
rs.getString("language_name");
                String actor_name =
rs.getString("actor_name");
                Timestamp lu = rs.getTimestamp("last_update");

                //Creates an FilmActor object using sixteen-
params constructor
                FilmActor obj = new FilmActor(actor_id,
film_id, title, description, year, lang_id, ori_lang_id,
duration, rate, length, cost, rating, features, lang_name,
actor_name, lu);
                System.out.println(obj);
            }
            rs.close();
            stmt.close();

        }catch(SQLException ex){
            JOptionPane.showMessageDialog(null,
ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }
}
```

Step 4

In the
create_
populat
read_fil

```
1  pack
2
3  publ
4
5  //
6  //
7  //
8  Acto
9  //
10 //
```

```
11 //
12
13 //
14 Lang
15 //
16 Lang
17 //
18 Lang
19 //
20 Lang
21 //
22
23 //
24 Cate
25 //
26 Cate
27 //
28 Cate
29 //
30 Cate
31 //
32
33 //
34 //
   //
   //
   //


   Film

   Film

   Film

   }
```

Run pro

```
Film ID
Actor ID
Actor Na
Title
Descript
Release
Language
Language
Original
Rental D
Rental R
Length
Replacem
```

	Rating Special Last Upd Film ID Actor ID Actor Na Title Descript Release Language Language Original Rental D Rental R Length Replacem Rating Special Last Upd

DESIGNING GUI
DESIGNING GUI

Step 1	In the project, create a new **JFrame Form** and name it as **FilmActorForm.java**. In the Design tab, add fifteen **JLabel** to the form and set their corresponding **text** properties as FILM ID, TITLE, DESCRIPTION, RELEASE YEAR, RENTAL RATE, RENTAL DURATION, LENGTH, REPLACEMENT COST, RATING, LANGUAGE, ORIGINAL LANGUAGE, SPECIAL FEATURES, ACTOR ID, ACTOR NAME, and LAST UPDATE.
Step 2	Then, add thirteen **JTextField** to the form and set set their corresponding **Variable Name** as **jtfFilmID**, **jtfActorID**, **jtfDescription**, **jtfReleaseYear**, **jtfRentalRate**, **jtfLength**, **jtfRentalDuration**, **jtfRating**, **jtfReplacementCost**, **jtfLanguageName**, **jtfOriginalLanguageName**, **jtfSpecialFeatures**, and **jtfLastUpdate**.
Step 3	Then, add eleven **JButton** to the form and set their corresponding **Variable Name** as **jbFirst**, **jbPrev**, **jbNext**, **jbLast**, **jbEdit**, **jbInsert**, **jbDelete**, **jbChart**, jbLanguageForm, **jbActorForm**, and

	jbFilmForm. Set their corresponding **text** properties as FIRST, PREV, NEXT, LAST, EDIT, INSERT, DELETE, CHART, LANGUAGE FORM, ACTOR FORM, and FILM FORM.
Step 4	Then, add four **JComboBox** to the form and set set their corresponding **Variable Name** as **jcbFilmID**, **jcbTitle** **jcbActorID**, and **jcbActorName**.
Step 5	Lastly, add a new **JTable** to the form set set its **Variable Name** as **jtFilmActor**. Then, right-click on it, then choose **Table Contents...** and set the number of columns to 16 and the number of rows to 50.
Step 6	In the driver class, **Sakila.java**, create a new object of **FilmActorForm** class using its default constructor as shown in 33 – 34:

```
1  package sakila;
2
3  public class Sakila {
4      public static void main(String[] args) {
5  //         Utility.testConnection();
6  //         Actor_Utils.create_actor_table();
7  //
8  Actor_Utils.populate_actor_table();
9  //         Actor_Utils.read_actor_table();
10 //         ActorForm frm = new ActorForm();
11 //         frm.setVisible(true);
12
13 //
14 Language_Utils.create_language_table();
15 //
16 Language_Utils.populate_language_table();
17 //
18 Language_Utils.read_language_table();
19 //         LanguageForm frm = new
20 LanguageForm();
21 //         frm.setVisible(true);
22
23 //
24 Category_Utils.create_category_table();
25 //
26 Category_Utils.populate_category_table();
27 //
28 Category_Utils.read_category_table();
29
```

```
30  //        CategoryForm frm = new
31  CategoryForm();
32  //        frm.setVisible(true);
33
34  //        Film_Utils.create_film_table();
35  //        Film_Utils.populate_film_table();
36  //        Film_Utils.read_film_table();
    //        FilmForm frm = new FilmForm();
    //        frm.setVisible(true);

    //
    FilmActor_Utils.create_film_actor_table();
    //
    FilmActor_Utils.populate_film_actor_table();
    //
    FilmActor_Utils.read_film_actor_table();
            FilmActorForm frm = new
    FilmActorForm();
            frm.setVisible(true);
        }
    }
```

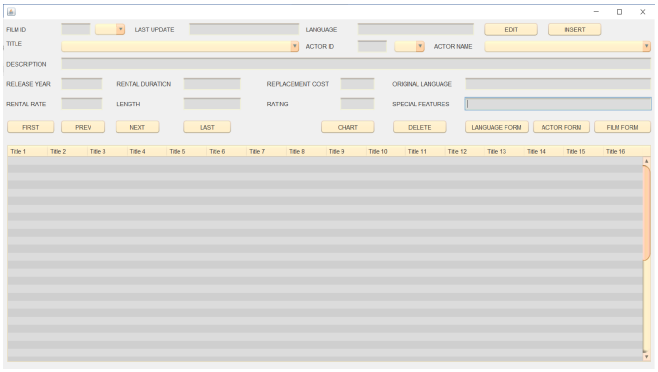


Figure 6.1 The layout of film actor form

Step 8

In **FilmActorForm**'s constructor, invoke **setLookAndFeel()** to set the look and feel of the form as shown in line 17.

```
1   package sakila;
2
3   import java.awt.Toolkit;
4   import java.awt.event.ActionEvent;
5   import
6   java.awt.event.ActionListener;
7   import javax.swing.JButton;
8   import javax.swing.JComboBox;
9   import javax.swing.JMenuItem;
10  import javax.swing.JOptionPane;
```

```
11 import javax.swing.JPopupMenu;
12 import javax.swing.JTable;
13 import javax.swing.JTextField;
14
15 public class FilmActorForm extends
16 javax.swing.JFrame {
17     public FilmActorForm() {
18         initComponents();
19
20 Utility.setLookAndFeel(this);
      }
      //...
  }
```

Run the project to see the film actor form as shown in Figure 6.1.

Step 9

In **FilmActorForm.java**, define **getter()** method for every object in the form and place them outside and beneath its default constructor:

```
1     //Getter methods for JTextField
2  variable instances
3     public JTextField
4  getJTFFilmID(){return
5  this.jtfFilmID;}
6     public JTextField
7  getJTFActorID(){return
8  this.jtfActorID;}
9     public JTextField
10 getJTFDescription(){return
11 this.jtfDescription;}
12    public JTextField
13 getJTFLastUpdate(){return
14 this.jtfLastUpdate;}
15    public JTextField
16 getJTFLanguageName(){return
17 this.jtfLanguageName;}
18    public JTextField
19 getJTFReleaseYear(){return
20 this.jtfReleaseYear;}
21    public JTextField
22 getJTFRentalDuration(){return
23 this.jtfRentalDuration;}
24    public JTextField
25 getJTFOriginalLanguageName(){return
26 this.jtfOriginalLanguageName;}
27    public JTextField
28 getJTFRentalRate(){return
29 this.jtfRentalRate;}
30
```

```
    public JTextField
getJTFLength(){return
this.jtfLength;}
    public JTextField
getJTFReplacementCost(){return
this.jtfReplacementCost;}
    public JTextField
getJTFSpecialFeatures(){return
this.jtfSpecialFeatures;}
    public JTextField
getJTFRating(){return
this.jtfRating;}

    //Getter methods for JComboBox
variable instances
    public JComboBox getJCBFilmID()
{return this.jcbFilmID;}
    public JComboBox
getJCBActorID(){return
this.jcbActorID;}
    public JComboBox getJCBTitle()
{return this.jcbTitle;}
    public JComboBox
getJCBActorName(){return
this.jcbActorName;}

    //Getter methods for JTable
variable instance
    public JTable getJTFilmActor()
{return this.jtFilmActor;}

    //Getter methods for JButton
variable instances
    public JButton getJBEdit()
{return this.jbEdit;}
    public JButton getJBInsert()
{return this.jbInsert;}
    public JButton getJBDelete()
{return this.jbDelete;}
    public JButton getJBChart()
{return this.jbChart;}
    public JButton getJBFirst()
{return this.jbFirst;}
    public JButton getJBPrev()
{return this.jbPrev;}
    public JButton getJBNext()
{return this.jbNext;}
    public JButton getJBLast()
{return this.jbLast;}
```

POPULATING TABLE AND COMBOBOXES
POPULATING TABLE AND COMBOBOXES

Step 1

In **FilmActor_Utils.java**, add two new methods: **get_film_actor_list()** an

The **get_film_actor_list()** method takes three parameters: a **FilmActorF** **sql**, and another **String** variable **item**. It returns an **ArrayList** of **FilmAc** **getConnection()** method is called to establish a database connection. The created with the SQL statement passed as a parameter. If the item pa "none", the first parameter of the **PreparedStatement** object is set to iter the **executeQuery()** method, and the resulting **ResultSet** is looped thro used to create a new **FilmActor** object using a sixteen-params constructor the **ArrayList** and finally, the **ArrayList** is returned.

The **show_table_film_actor()** method takes two parameters: a **FilmActor** of **FilmActor** objects **list**. This method does not return anythi **DefaultTableModel** object is created with no rows and no columns. Th String array named **header**. The **set_column_header()** method is called header array. Then, the **DefaultTableModel** is set to the **JTable** compo passed as a parameter. A new **Object** array named row is created with 16 through the **ArrayList** of **FilmActor** objects. Inside the loop, each proper to the row array. Finally, the row array is added as a new row to t **addRow()** method.

```
1     private static ArrayList<FilmActor>
2  get_film_actor_list(FilmActorForm frm, String sql, String item){
3          ArrayList<FilmActor> list = new ArrayList<>();
4          Connection conv = null;
5
6          try(Connection conn = getConnection();
7              PreparedStatement ps = conn.prepareStatement(sql)){
8              if (item.equalsIgnoreCase("none")==false) {
9                   ps.setString(1,item);
10             }
11             ResultSet rs = ps.executeQuery();
12
13             FilmActor obj;
14             while(rs.next()){
15                 //Using sixteen-params constructor
16                 obj = new FilmActor(rs.getInt("actor_id"),
17 rs.getInt("film_id"),
18                     rs.getString("title"),
19                     rs.getString("description"),
20 rs.getInt("release_year"),
21                     rs.getInt("language_id"),
22 rs.getInt("original_language_id"),
23
```

```
                    rs.getInt("rental_duration"),
rs.getDouble("rental_rate"),
                    rs.getInt("length"),
rs.getDouble("replacement_cost"),
                    rs.getString("rating"),
rs.getString("special_features"),
                    rs.getString("language_name"),
rs.getString("actor_name"),
                        rs.getTimestamp("last_update"));

                list.add(obj);
            }
        }catch (SQLException ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
                "ERROR",JOptionPane.ERROR_MESSAGE);
        }
        return list;
    }

    private static void show_table_film_actor(FilmActorForm frm,
ArrayList<FilmActor> list) throws SQLException{
        DefaultTableModel model = new DefaultTableModel(0,0);

        String header[] = {"Actor ID", "Actor Name", "Film ID",
"Title", "Description", "Release Year", "Language ID", "Original
Language", "Rental Duration", "Rental Rate", "Length",
"Replacement Cost", "Rating", "Special Features", "Language Name",
"Last Update"};


model.setColumnIdentifiers(set_column_header(frm.getJTFilmActor(),
header));
        frm.getJTFilmActor().setModel(model);

        Object[] row = new Object[16];

        for(int i=0; i<list.size(); i++){
            row[0] = list.get(i).getActorID();
            row[1] = list.get(i).getActorName();
            row[2] = list.get(i).getFilmID();
            row[3] = list.get(i).getTitle();
            row[4] = list.get(i).getDescription();
            row[5] = list.get(i).getReleaseYear();
            row[6] = list.get(i).getLanguageID();
            row[7] = list.get(i).getOriginalLanguageID();
            row[8] = list.get(i).getRentalDuration();
            row[9] = list.get(i).getRentalRate();
            row[10] = list.get(i).getLength();
            row[11] = list.get(i).getReplacementCost();
            row[12] = list.get(i).getRating();
            row[13] = list.get(i).getSpecialFeatures();
            row[14] = list.get(i).getLanguageName();
            row[15] = list.get(i).getLastUpdate();
```

```
            model.addRow(row);
        }
    }
```

Step 2

In **FilmActor_Utils.java**, define **refresh_controls()** method. It is used
data from the database and set the initial state of the form.

Here is what this method does:

1. It sets the location and title of the form.
2. It sets the renderer for the **JTable** in the form to have alternating r
3. It retrieves the list of film actors from the database by calling the
   a SQL query that joins the film and actor tables and orders the resu
4. It populates the **JTable** in the form with the retrieved data by ca
   method with the list of film actors.
5. It populates the **JComboBox** for the film ID with data fro
   populate_combobox() method with a SQL query that selects all
   and orders them by **film_id**.
6. It populates the **JComboBox** for the film title with data fr
   populate_combobox() method with a SQL query that selects all
   table and orders them by **title**.
7. It populates the **JComboBox** for the actor ID with data fro
   populate_combobox() method with a SQL query that selects a
   table and orders them by **actor_id**.
8. It populates the **JComboBox** for the actor name with data fr
   populate_combobox() method with a SQL query that selects all
   concatenated values from the **actor** table and orders them by the c

If any error occurs during the data retrieval or population, it shows an erro

```
1     public static void
2  refresh_controls(FilmActorForm frm){
3          frm.setLocationRelativeTo(null);
4          frm.setTitle("FILM ACTOR FORM");
5
6          //Shows the content of film table
7  and populates combobox
8          try{
9              //Makes alternating color for
10 table rows
11
12 table_renderer(frm.getJTFilmActor());
13
14             //Populates table
15             ArrayList<FilmActor> list =
16 get_film_actor_list(frm,
17 Query_FilmActor.get_sql_film_actor_joint()
18 + " ORDER BY f.film_id", "none");
```

```
            show_table_film_actor(frm,
list);

            //Populates jcbFilmID
            String sql_id = "SELECT
film_id FROM film ORDER BY film_id";
            populate_combobox(sql_id,
frm.getJCBFilmID(), frm);

            //Populates getJCBTitle()
            String sql_title = "SELECT
DISTINCT title FROM film ORDER BY title";
            populate_combobox(sql_title,
frm.getJCBTitle(), frm);

            //Populates jcbActorID
            String sql_act_id = "SELECT
actor_id FROM actor ORDER BY actor_id";
            populate_combobox(sql_act_id,
frm.getJCBActorID(), frm);

            //Populates jcbActorName
            String sql_act_name = "SELECT
DISTINCT CONCAT(first_name,' ',last_name)
FROM actor ORDER BY CONCAT(first_name,'
',last_name)";

populate_combobox(sql_act_name,
frm.getJCBActorName(), frm);

        }catch (SQLException ex){

JOptionPane.showMessageDialog(frm,
ex.getMessage(),

 "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }
```

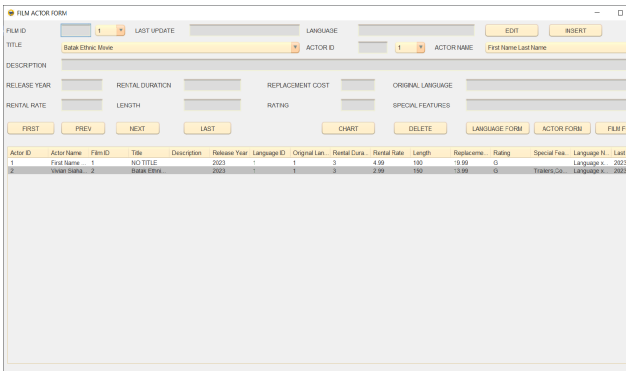


Figure 6.2 The content of joined **film_actor**, **film**, **actor**, and **languag**

Step 3

In **FilmActorForm**'s default constructor, the **FilmActor_Utils.refresh** and populates the controls in the **FilmActorForm** with data from a data method from the **FilmActor_Utils** class.

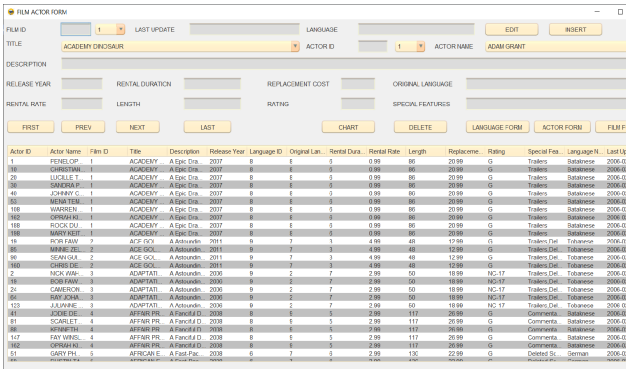


Figure 6.3 The the content of joined **film_actor**, **film**, **actor**, and **lan** database available in the internet displayed in

The **this.setIconImage()** method sets the icon of the **FilmA** **this.setDefaultCloseOperation(this.HIDE_ON_CLOSE)** method set **FilmActorForm** to hide the form instead of exiting the application when t

```
1  package sakila;
2  import java.awt.Toolkit;
3  import java.awt.event.ActionEvent;
4  import java.awt.event.ActionListener;
5  import java.io.IOException;
6  import java.text.ParseException;
7  import java.util.logging.Level;
8  import java.util.logging.Logger;
9  import javax.swing.JButton;
10 import javax.swing.JComboBox;
11 import javax.swing.JLabel;
12 import javax.swing.JTable;
13 import javax.swing.JTextField;
14
15 public class FilmActorForm extends javax.swing.JFrame {
16     public FilmActorForm() {
17         initComponents();
18         Utility.setLookAndFeel(this);
19         FilmActor_Utils.refresh_controls(this);
20
21  this.setIconImage(Toolkit.getDefaultToolkit().getImage(getClass().
22 ;
23         this.setDefaultCloseOperation(this.HIDE_ON_CLOSE);
24     }
25     //...
26 }
```

Step 4	Run the project to see the content of joined **film_actor**, **film**, **actor**, **jtFilmActor** as shown in Figure 6.2. If you use the data from **Sakila** MySQL database available in the intern joined **film_actor**, **film**, **actor**, and **language** tables displayed in **jtFilmAc**

DISPLAYING AND NAVIGATING DATA ROW BY ROW
DISPLAYING AND NAVIGATING DATA ROW BY ROW

Step 1

In **FilmActor_Utils** class, define two new methods named **cle** **display_film_data()**.

- **clear_controls(FilmActorForm frm)**: This method takes in a **Fil** as a parameter and clears the values of all the controls in the for all the **JTextFields** to an empty string.
- **display_film_data(FilmActorForm frm, String sql, T item)**: Th **FilmActorForm** object, a SQL query string, and an item of type executes the SQL query with the given parameter using a **P** retrieves the **ResultSet**, and iterates over the rows. For each values of various controls in the form with the correspondi **ResultSet**. It also calls other methods to update the values of **JTextFields** that are related to the current row. If no rows ar **clear_controls()** method to clear the values of all the controls in t

```
1     private static void clear_controls(FilmActorForm frm){
2         frm.getJTFFilmID().setText("");
3         frm.getJTFDescription().setText("");
4         frm.getJTFReleaseYear().setText("");
5         frm.getJTFLanguageName().setText("");
6         frm.getJTFOriginalLanguageName().setText("");
7         frm.getJTFRentalDuration().setText("");
8         frm.getJTFRentalRate().setText("");
9         frm.getJTFLength().setText("");
10        frm.getJTFReplacementCost().setText("");
11        frm.getJTFSpecialFeatures().setText("");
12        frm.getJTFLastUpdate().setText("");
13    }
14
15   //Displays film data result row by row
16    private static <T> void display_film_data(FilmActorForm frm, St
17        try(Connection conn = getConnection()){
18            PreparedStatement ps = conn.prepareStatement(sql);
19            ps.setObject(1,item);
20            ResultSet rs = ps.executeQuery();
21
```

```
22
23              if (!rs.next()) {
24                  // no row found, clear the form fields
25                  clear_controls(frm);
26                  return;
27              }
28
29              do{
30                  frm.getJTFFilmID().setText(String.valueOf(rs.getInt
31                  frm.getJTFDescription().setText(rs.getString("descr
32
33  frm.getJTFReleaseYear().setText(String.valueOf(rs.getInt("release_y
34                  frm.getJTFLanguageName().setText(rs.getString("lang
35
36  frm.getJTFRentalDuration().setText(String.valueOf(rs.getInt("rental
37
38  frm.getJTFRentalRate().setText(String.valueOf(rs.getDouble("rental_
39                  frm.getJTFLength().setText(String.valueOf(rs.getInt
40
41  frm.getJTFReplacementCost().setText(String.valueOf(rs.getDouble("re
42                  frm.getJTFSpecialFeatures().setText(rs.getString("s
43                  frm.getJTFRating().setText(rs.getString("rating"));
44
45  frm.getJTFLastUpdate().setText(String.valueOf(rs.getDate("last_upda
46
47                  // Determines item selected from jcbFilmID
48                  find_combo_value_selected(frm.getJCBFilmID(), rs.ge
49
50                  // Determines item selected from jcbTitle
51                  find_combo_value_selected(frm.getJCBTitle(), rs.get
52
53                 // Finds original language name
54                  int ori_lang_id = rs.getInt("original_language_id")
55                  if(ori_lang_id != 0) {
56                      Object lang_name = get_val_from_database("langu
57  "language_id", ori_lang_id);
58
59  frm.getJTFOriginalLanguageName().setText(String.valueOf(lang_name))
60
61                  }
62
63              }while(rs.next());
64
65              rs.close();
66              ps.close();
67          }catch(SQLException ex){
68              JOptionPane.showMessageDialog(frm, ex.getMessage(),
69                      "ERROR",JOptionPane.ERROR_MESSAGE);
70          }
71      }
72
73
74
75
```

Step 2

In the same class, define another method named **jcbFilm_handler()**. It is when the user selects an item in either the **jcbFilmID** or **jcbTitle** **FilmActorForm**.

The method first retrieves the selected item from the combo box and ther to retrieve film data from the database based on the selected item. If t **jcbFilmID**, the query is constructed to find the film with the matching **jcbTitle**, the query is constructed to find the film with the matching **title**.

The method then tries to retrieve a list of **FilmActor** objects that are selected film. This is done by calling the **get_film_actor_list()** method **FilmActorForm**, the SQL query to retrieve the **FilmActor** data, and the error occurs during this process, an error message is displayed using a **JO** After retrieving the list of **FilmActor** objects, the method calls the **show** method to display the data in the **JTable** in the **FilmActorForm**. This **FilmActorForm** and the list of **FilmActor** objects as parameters.

Finally, the method calls the **display_film_data()** method, passing in the constructed SQL query, and the selected item. This method retrieves database and displays it in the appropriate fields in the **FilmActorForm**.

```
1   public static void
2  jcbFilm_handler(FilmActorForm frm,
3  JComboBox<String> jcb) {
4          Object item =
5  jcb.getSelectedItem();
6          String sql = "";
7          if
8  (jcb.equals(frm.getJCBFilmID())) {
9              sql =
10 Query_Film.get_sql_film_joint() + " WHERE
11 f.film_id = ?";
12
13             try{
14                 //Filters jtFilmActor
15                 ArrayList<FilmActor> list
16 = get_film_actor_list(frm,
17 Query_FilmActor.get_sql_film_actor_joint()
18 + " WHERE fa.film_id = ?",
19 String.valueOf(item));
20                 show_table_film_actor(frm,
21 list);
22             }catch (SQLException ex){
23
24 JOptionPane.showMessageDialog(frm,
25 ex.getMessage(),
26
```

```
27
28  "ERROR",JOptionPane.ERROR_MESSAGE);
29              }
30          } else if
31 (jcb.equals(frm.getJCBTitle())) {
32              sql =
33 Query_Film.get_sql_film_joint() + " WHERE
34 f.title = ?";

                try{
                    //Filters jtFilmActor
                    ArrayList<FilmActor> list
= get_film_actor_list(frm,
Query_FilmActor.get_sql_film_actor_joint()
+ " WHERE f.title = ?",
String.valueOf(item));
                    show_table_film_actor(frm,
list);
                }catch (SQLException ex){

JOptionPane.showMessageDialog(frm,
ex.getMessage(),

 "ERROR",JOptionPane.ERROR_MESSAGE);
                }
            }

            display_film_data(frm, sql, item);
        }
```

Step 3

In **FilmActorForm**, double click on **jcbFilmID** and **jcbTitle** combobox corresponding event handler as follows:

```
1       private void
2 jcbFilmIDActionPerformed(java.awt.event.ActionEvent
3 evt) {
4           FilmActor_Utils.jcbFilm_handler(this,
5 this.jcbFilmID);
6       }
7

        private void
jcbTitleActionPerformed(java.awt.event.ActionEvent
evt) {
            FilmActor_Utils.jcbFilm_handler(this,
this.jcbTitle);
        }
```

These are event handlers for the two **JComboBoxes** in the **FilmActo** selection is made in either of the **JComboBoxes**, the corresponding eve

which then calls the **jcbFilm_handler()** method in the **FilmActor_Utils** **FilmActorForm** instance and the selected **JComboBox**.

The **jcbFilm_handler()** method then gets the selected item from th determines the SQL query to use based on the selected **JCombo** **get_film_actor_list()** to retrieve the list of **FilmActor** objects based on t displays the list in the table. Finally, it calls **display_film_data()** to displ selected item in the form fields.

Step 4

Run the project. Choose one of items in **jcbFilmID** and/or **jcbTitle** combo row the content of **film** table as shown in Figure 6.4.

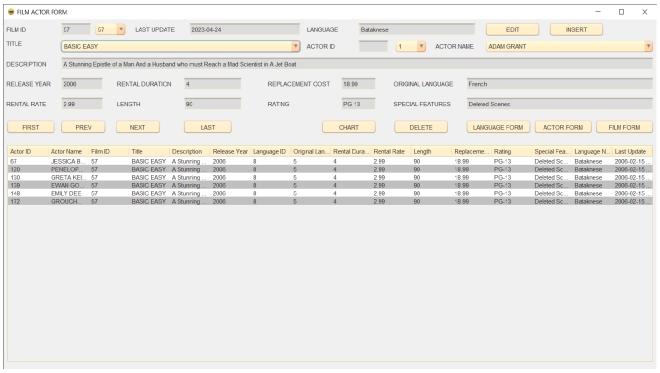


Figure 6.4 Displaying row by row the content of **film** ta

Step 5

In **FilmActor_Utils** class, define a new method named **display_actor_da** arguments:

1. **FilmActorForm** frm: an instance of the **FilmActorForm** class th elements for displaying actor data.
2. **String** sql: a SQL query that retrieves actor data from the databas
3. T item1, T item2: two items of type T that are used as paramet statement. These items correspond to the ? placeholders in the SQ

The method establishes a database connection and creates a prepared provided SQL query. It sets the first and second parameters of the prep the **item1** and **item2** arguments, respectively. The method then exec retrieves the result set.

If the result set does not contain any rows, the method clears the acto fields in the **FilmActorForm**. If there are rows in the result set, the met them and sets the actor ID and last update fields in the **FilmActorFo** **find_combo_value_selected()** method to set the selected item in t **jcbActorName** combo boxes based on the actor ID and name retrieved fr

Finally, the method closes the result set and prepared statement, an exceptions that may occur, displaying an error message in a dialog box if n

```
    private static <T> void display_actor_data(FilmActorForm
frm, String sql, T item1, T item2) {
        try(Connection conn = getConnection()){
            PreparedStatement ps =
conn.prepareStatement(sql);
            ps.setObject(1,item1);
            ps.setObject(2,item2);
            ResultSet rs = ps.executeQuery();

            if (!rs.next()) {
                // no row found, clear the form fields
                frm.getJTFLastUpdate().setText("");
                return;
            }

            do {

frm.getJTFActorID().setText(rs.getString("actor_id"));

frm.getJTFLastUpdate().setText(rs.getString("last_update"));

                // Determines item selected from jcbActorID

find_combo_value_selected(frm.getJCBActorID(),
rs.getInt("actor_id"));

                // Determines item selected from
jcbActorName
                String actor_name =
rs.getString("first_name") + " " +
rs.getString("last_name");

find_combo_value_selected(frm.getJCBActorName(),
actor_name);
            } while(rs.next());

            rs.close();
            ps.close();
        } catch(SQLException ex){
            JOptionPane.showMessageDialog(frm,
ex.getMessage(), "ERROR", JOptionPane.ERROR_MESSAGE);
        }
    }
```

Step 6	In the same class, define another method named **jcbActor_handler()**. T the events when the user selects an item from the **JComboBox FilmActorForm**. It takes in the **FilmActorForm** object and two **JCo** arguments. The first **JComboBox** object (**jcb1**) represents either the **jcbActorName** JComboBox in the **FilmActorForm**, depending on which i

The second **JComboBox** object (**jcb2**) represents the **jcbFilmID FilmActorForm**.

The method first gets the selected items from both **JComboBoxes**. Then, **JComboBox** is calling the method, it sets the SQL query accordingly to re database. If **jcb1** represents the **jcbActorID**, the query will retrieve data table where the **actor_id** and **film_id** match the selected items. If j **jcbActorName**, the query will retrieve data from the **film_actor** and ac actor's full name (first name and last name concatenated) and **film_id** items.
Finally, the method calls the **display_actor_data()** method to display the **FilmActorForm**. It passes the **FilmActorForm** object, the SQL query, ar as arguments.

```
    public static void
jcbActor_handler(FilmActorForm frm,
JComboBox<String> jcb1, JComboBox<String>
jcb2) {
        Object item1 =
jcb1.getSelectedItem();
        Object item2 =
jcb2.getSelectedItem();
        String sql = "";
        if
(jcb1.equals(frm.getJCBActorID())) {
            sql =
Query_FilmActor.get_sql_film_actor_joint()
+ " WHERE fa.actor_id = ? AND fa.film_id =
?";
        } else if
(jcb1.equals(frm.getJCBActorName())) {
            sql =
Query_FilmActor.get_sql_film_actor_joint()
+ " WHERE CONCAT(a.first_name,'
',a.last_name) = ? AND fa.film_id = ?";
        }

        display_actor_data(frm, sql,
item1, item2);
    }
```

Step 7

In **FilmActorForm**, double click on **jcbActorID** and **jcbActorName** cor their corresponding event handler as follows:

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

```

```
5           FilmActor_Utils.jcbActor_handler(this,
6   this.jcbActorID, this.jcbFilmID);
7       }

        private void
    jcbActorNameActionPerformed(java.awt.event.ActionEvent
    evt) {
            FilmActor_Utils.jcbActor_handler(this,
    this.jcbActorName, this.jcbFilmID);
        }
```

These are event handlers for the **JComboBox** components **jcbActorID** When an action (such as selecting an item from the list) is performed component, it triggers the **jcbActorIDActionPerformed()** method, whi **jcbActor_handler()** method from the **FilmActor_Utils** class, passi **FilmActorForm** object as well as the **jcbActorID** and **jcbFil jcbActor_handler()** method then retrieves the selected items from t **jcbFilmID** objects, constructs an SQL query string using the **Query_F** calls the **display_actor_data()** method with the query string and parameters.

The same process occurs when an action is performed on the **jcbActo** except the **jcbActorNameActionPerformed()** method is triggered inst **jcbActor_handler()** method with the **jcbActorName** and **jcbFilmID** obje

Step 8

Run the project. Choose **film_id** using **jcbFilmID** and/or **jcbTitle** comb **film_id = 3**). Then, choose available **actor_id** relating to chosen **film_i** and/or **jcbActorName** comboboxes (in this case, **actor_id = 2**) as shown

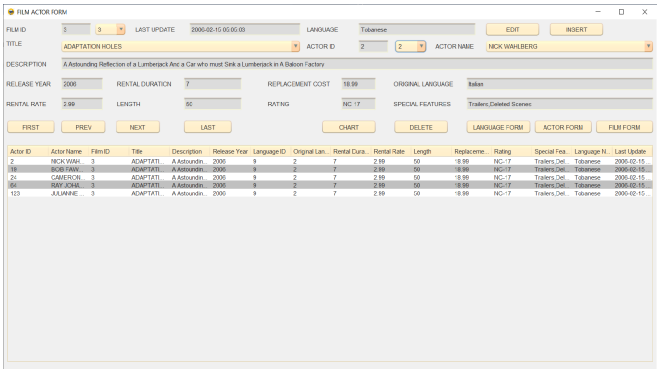


Figure 6.5 Displaying row by row the content of **actor** ta

Step 9

Define four navigating methods in **FilmActor_Utils** class. These methods through the available film records in the **JComboBox**. Here's a brief expl method does:

1. **show_first_row()**: Displays the first row of data in the **JComl** **currentIndex** to 0.
2. **show_last_row()**: Displays the last row of data in the **JComb** currentIndex to the index of the last item in the **JComboBox**.
3. **show_prev_row()**: Displays the previous row of data in the **JCom** current index. If the **currentIndex** is less than the first index (beyond the first index.
4. **show_next_row()**: Displays the next row of data in the **JComb** current index. If the **currentIndex** is greater than the last index won't go beyond the last index.

These methods use the **display_film_data()** method to display the film d item in the **JComboBox**. The **currentIndex** is used to keep track of the row, and it's updated whenever the user navigates through the records.

```
1     public static void show_first_row(FilmActorForm frm){
2         String item =
3 String.valueOf(frm.getJCBFilmID().getItemAt(FIRST_INDEX));
4         display_film_data(frm, SQL_ID, item);
5         currentIndex = FIRST_INDEX;
6     }
7
8     public static void show_last_row(FilmActorForm frm){
9         int endIndex = frm.getJCBFilmID().getItemCount() -
10 1;
11        String item =
12 String.valueOf(frm.getJCBFilmID().getItemAt(endIndex));
13        display_film_data(frm, SQL_ID, item);
14        currentIndex = endIndex;
15    }
16
17    public static void show_prev_row(FilmActorForm frm){
18        currentIndex--;
19        if(currentIndex < FIRST_INDEX){
20            currentIndex = FIRST_INDEX;
21            return;
22        }
23        String item =
24 String.valueOf(frm.getJCBFilmID().getItemAt(currentIndex));
25        display_film_data(frm, SQL_ID, item);
26    }
27
28    public static void show_next_row(FilmActorForm frm){
29        int endIndex = frm.getJCBFilmID().getItemCount() -
30 1;
31        currentIndex++;
32        if(currentIndex > endIndex){
33            currentIndex = endIndex;
              return;
          }
```

```
        String item =
String.valueOf(frm.getJCBFilmID().getItemAt(currentIndex));
        display_film_data(frm, SQL_ID, item);
    }
```

Step 10

Then in **FilmActorForm**, double click on each navigation butto corresponding event handler:

```
    private void
jbFirstActionPerformed(java.awt.event.ActionEvent
evt) {
       FilmActor_Utils.show_first_row(this);
    }

    private void
jbLastActionPerformed(java.awt.event.ActionEvent
evt) {
        FilmActor_Utils.show_last_row(this);
    }

    private void
jbPrevActionPerformed(java.awt.event.ActionEvent
evt) {
        FilmActor_Utils.show_prev_row(this);
    }

    private void
jbNextActionPerformed(java.awt.event.ActionEvent
evt) {
        FilmActor_Utils.show_next_row(this);
    }
```

These methods are event handlers for the four buttons in the user interfac to navigate through the list of films.

- **jbFirstActionPerformed()**: When the user clicks this button, tl method from the **FilmActor_Utils** class is called, which displays film list in the user interface.
- **jbLastActionPerformed()**: When the user clicks this button, t method from the **FilmActor_Utils** class is called, which displays film list in the user interface.
- **jbPrevActionPerformed()**: When the user clicks this button, th method from the **FilmActor_Utils** class is called, which displays the film list in the user interface.
- **jbNextActionPerformed()**: When the user clicks this button, th method from the **FilmActor_Utils** class is called, which displays film list in the user interface.

Step 11

Run the project. Click on one or more navigation buttons to see the resul 6.6.

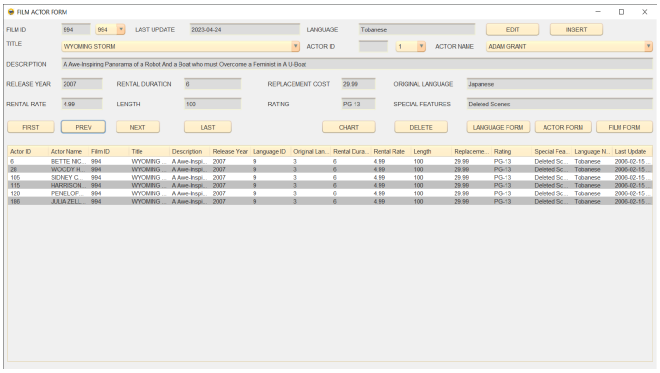


Figure 6.6 User clicks on one or more navigation buttons on film

Step 12

Define **mouse_pressed_handler()** method in **FilmActor_Utils** class. Th handle mouse press events on the **JTable** component in the **FilmActorFor** row is selected, and if not, displays a message to select a row. If a row is values of the first and third columns (**actor_id** and **film_id**) and uses database and display the film actor data in the form. The SQL query used **jcbActor_handler()** method. If there is an SQL exception, it logs the e error message dialog.

```
1     public static void mouse_pressed_handler(FilmActorForm frm) {
2         Objects.requireNonNull(frm, "frm must not be null");
3
4         int selectedIndex = frm.getJTFilmActor().getSelectedRow();
5         if (selectedIndex == -1) {
6             JOptionPane.showMessageDialog(frm, "Please select a row to view its d
7                 "No row selected", JOptionPane.INFORMATION_MESSAGE)
8             return;
9         }
10
11        try (Connection conn = getConnection()) {
12            String item1 =
13 String.valueOf(frm.getJTFilmActor().getModel().getValueAt(selectedI
14 0));
15            String item2 =
16 String.valueOf(frm.getJTFilmActor().getModel().getValueAt(selectedI
17 2));
18
19            // Displays film actor data
20            String sql = Query_FilmActor.get_sql_film_actor_joint()
21 WHERE fa.actor_id = ? AND fa.film_id = ?";
22            display_actor_data(frm, sql, item1, item2);
23
24        } catch (SQLException ex) {
25
```

```
26
27 Logger.getLogger(FilmActorForm.class.getName()).log(Level.SEVERE, "
28 displaying film actor data", ex);
29             String message = "Error displaying film actor data: " +
30 ex.getMessage();
31             String stackTrace = Arrays.toString(ex.getStackTrace())
               JOptionPane.showMessageDialog(frm, message + "\n\n" +
               stackTrace, "ERROR", JOptionPane.ERROR_MESSAGE);
         }
     }
```

Step 13

Right click on **jtFilmActor**. Then, choose **Events > Mouse > mousePres** handler:

```
1     private void
2 jtFilmActorMousePressed(java.awt.event.MouseEvent
3 evt) {

  FilmActor_Utils.mouse_pressed_handler(this);
     }
```

Step 14

Run the project. Choose **film_id** using **jcbFilmID** and/or **jcbTitle** comb **film_id = 999**). Then, double click on any row in **jtFilmActor** tabl corresponding row in **film_actor** table displayed in textfields and comb Figure 6.7.

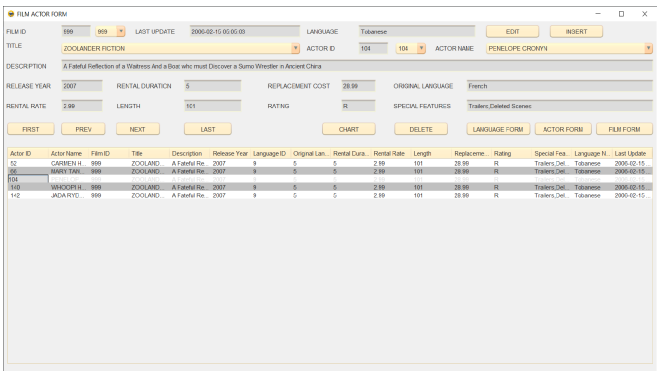


Figure 6.7 User double-clicks on any row in **jtFilmAct**

UPDATING RECORD
UPDATING RECORD

Step 1

In **FilmActor_Utils** class, define a new method named **update_row_by_film_actor_id()**. This method updates a row of data in the **film_actor** table by the **film_id** and **actor_id**. It takes a **FilmActorForm** object, which represents the form where the data is displayed, and three integers: **film_id**, **act_id**, and **act_id_new**, where **film_id** and **act_id** represent the primary key of the row to be updated, and **act_id_new** represents the new value for the **actor_id** column.

The method first establishes a connection to the database using the **getConnection()** method, and then creates two prepared statements: **idPs** and **updatePS**. The **idPs** prepared statement is used to retrieve the row with the given **film_id**, and the **updatePS** prepared statement is used to update the row with the new **actor_id**.

If the **idPs** prepared statement doesn't return any rows, the method displays an error message using a **JOptionPane**. If the **idPs** prepared statement returns a row, the method creates a new **FilmActor** object using the **act_id**, **film_id**, and the current time as parameters, and sets the parameters of the **updatePS** prepared statement using the **FilmActor** object and the **act_id_new** parameter.

Finally, the method executes the **updatePS** prepared statement and closes the resources used (the **ResultSet**, **updatePS**, **idPs**, and the **Connection**). If an exception occurs during the process, the method logs the error and displays an error message using a **JOptionPane**.

```
    //Updates row of data in film_actor tabel by film_id and
actor_id
    public static void update_row_by_film_actor_id(FilmActorForm
frm, int film_id, int act_id, int act_id_new) throws
SQLException{
        Connection conn = getConnection();
        ResultSet rs = null;
        String query_id = "SELECT film_id FROM film WHERE film_id
= ?";
        String update_query = """
            UPDATE film_actor SET film_id = ?, actor_id = ?
            WHERE film_id = ? AND actor_id = ?""";
        try(PreparedStatement idPs =
conn.prepareStatement(query_id,

ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_READ_ONLY);
        PreparedStatement updatePS =
conn.prepareStatement(update_query,
                ResultSet.TYPE_SCROLL_SENSITIVE,
ResultSet.CONCUR_UPDATABLE))
        {
            idPs.setInt(1,film_id);
            if(!idPs.execute()){
```

```
                String message = "Can't find film_id " + film_id;

                JOptionPane.showMessageDialog(frm, message,
                    "ERROR",JOptionPane.ERROR_MESSAGE);
            } else{
                rs = idPs.getResultSet();
                rs.next();

                //Creates a FilmActor object using three-params
constructor
                FilmActor obj = new FilmActor(act_id, film_id,
new Timestamp(System.currentTimeMillis()));
                updatePS.setInt(1, obj.getFilmID());
                updatePS.setInt(2, act_id_new);
                updatePS.setInt(3, obj.getFilmID());
                updatePS.setInt(4, obj.getActorID());

                updatePS.executeUpdate();
                rs.close();
                updatePS.close();
                idPs.close();
                conn.close();
            }
        }catch(SQLException ex){

Logger.getLogger(FilmActorForm.class.getName()).log(Level.SEVERE,
"Error updating film-actor data", ex);
            String message = "Error updating film-actor data: " +
ex.getMessage();
            String stackTrace =
Arrays.toString(ex.getStackTrace());
            JOptionPane.showMessageDialog(null, message + "\n\n"
+ stackTrace, "ERROR", JOptionPane.ERROR_MESSAGE);
        }catch(java.lang.NumberFormatException ex){

Logger.getLogger(FilmActorForm.class.getName()).log(Level.SEVERE,
"Invalid Input", ex);
            String message = "Invalid Input: " + ex.getMessage();
            String stackTrace =
Arrays.toString(ex.getStackTrace());
            JOptionPane.showMessageDialog(null, message + "\n\n"
+ stackTrace, "ERROR", JOptionPane.ERROR_MESSAGE);
        }
    }
```

Step 2	Then in the same class, define a new method **read_inputs()**. It reads user input from the **FilmActorForm** object and returns a **HashMap** object containing the input data.

The method first initializes an empty **HashMap** object to store the input data. It then retrieves the values of the film ID, current actor ID, and new actor ID fields from the form.

Next, the method validates the film ID input to ensure that it is a positive integer. If the input is invalid, the method displays an error message and throws a **NumberFormatException** or an **IllegalArgumentException**.

The method also checks that the current actor ID field is not empty, and that the new actor ID field is not null or empty. If either of these fields is invalid, the method displays an error message and throws an **IllegalArgumentException**.

Finally, the method stores the validated input data in the **HashMap** object and returns it.

```
    private static HashMap<String, String>
read_inputs(FilmActorForm frm) {
        HashMap<String, String> input_data = new
HashMap<>();
        String film_id =
String.valueOf(frm.getJCBFilmID().getSelectedItem());
        String act_id = frm.getJTFActorID().getText();
        String act_id_new =
String.valueOf(frm.getJCBActorID().getSelectedItem());

        // Validate user input
        int film_id_int = 0;
        try {
            film_id_int = Integer.parseInt(film_id);
            if (film_id_int <= 0) {
                throw new
IllegalArgumentException("Film ID cannot be negative
or zero");
            }
        } catch (NumberFormatException ex) {
            JOptionPane.showMessageDialog(frm,
"Invalid Film ID: " + film_id,
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        } catch (IllegalArgumentException ex) {
            JOptionPane.showMessageDialog(frm,
ex.getMessage(),
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        }

        if (act_id == null) {
            JOptionPane.showMessageDialog(frm, "Actor
ID cannot be empty",
            "Error", JOptionPane.ERROR_MESSAGE);
```

```
            throw new IllegalArgumentException("Actor
ID cannot be empty");
        }

        if (act_id_new == null ||
act_id_new.isEmpty()) {
            JOptionPane.showMessageDialog(frm, "New
Actor ID cannot be empty",
            "Error", JOptionPane.ERROR_MESSAGE);
            throw new IllegalArgumentException("Ner
Actor ID cannot be empty");
        }

        input_data.put("film_id", film_id);
        input_data.put("act_id", act_id);
        input_data.put("act_id_new", act_id_new);

        return input_data;
    }
```

Step 3

Still in the same class, define another method named **edit_actual()**. It is responsible for handling the actual editing of a film-actor row. It first reads the inputs from the **FilmActorForm**, validates them using the read_inputs method, and then parses the **film_id**, **act_id**, and **act_id_new** values as integers. It then calls the **update_row_by_film_actor_id()** method to update the row in the database, passing in the parsed values. Finally, it refreshes all the objects on the form using the **refresh_controls()** method.

```
    private static void
edit_actual(FilmActorForm frm){
        try{
            HashMap<String, String> input_data
= read_inputs(frm);
            int film_id =
Integer.parseInt(input_data.get("film_id"));
            int act_id =
Integer.parseInt(input_data.get("act_id"));
            int act_id_new =
Integer.parseInt(input_data.get("act_id_new"));

            System.out.println(film_id);
            System.out.println(act_id);
            System.out.println(act_id_new);

            update_row_by_film_actor_id(frm,
film_id, act_id, act_id_new);

            //Refreshes all objects on form
            refresh_controls(frm);
```

```
        }catch(SQLException ex){
            JOptionPane.showMessageDialog(frm,
ex.getMessage(),

 "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }
```

Step 4

Lastly, define two new methods named **enable_controls()** and **edit_handler()**. The **edit_handler()** handles the editing of a film actor record in the form. The method first checks whether the "EDIT" button has been clicked, and if so, changes its text to "CONFIRM" and disables all other controls on the form. If the "CONFIRM" button is clicked, the method calls **edit_actual()** to perform the editing of the record, and then changes the button text back to "EDIT" and re-enables all controls on the form.

The **enable_controls()** method is a helper method that takes a boolean state and a **FilmActorForm** object **frm**, and sets the enabled state of all controls on the form based on the value of state.

```
1   private static void enable_controls(boolean
2  state, FilmActorForm frm){
3          frm.getJBFirst().setEnabled(state);
4          frm.getJBPrev().setEnabled(state);
5          frm.getJBNext().setEnabled(state);
6          frm.getJBLast().setEnabled(state);
7          frm.getJBInsert().setEnabled(state);
8          frm.getJBDelete().setEnabled(state);
9          frm.getJTFFilmID().setEnabled(state);
10         frm.getJTFDescription().setEnabled(state);
11         frm.getJTFReleaseYear().setEnabled(state);
12         frm.getJTFLanguageName().setEnabled(state);
13
14 frm.getJTFOriginalLanguageName().setEnabled(state);
15
16 frm.getJTFRentalDuration().setEnabled(state);
17         frm.getJTFRentalRate().setEnabled(state);
18         frm.getJTFLength().setEnabled(state);
19
20 frm.getJTFReplacementCost().setEnabled(state);
21
22 frm.getJTFSpecialFeatures().setEnabled(state);
23         frm.getJTFLastUpdate().setEnabled(state);
24     }
25
26     public static void edit_handler(FilmActorForm
27 frm){
```

```
28
29  if(frm.getJBEdit().getText().equals("EDIT")){
30              frm.getJBEdit().setText("CONFIRM");
31
32              // Disables controls
33              enable_controls(false, frm);
34          }
35
36          else {
37              frm.getJBEdit().setText("EDIT");
38
                // Actual editing
                edit_actual(frm);

                //Enables controls
                enable_controls(true, frm);
            }
    }
```

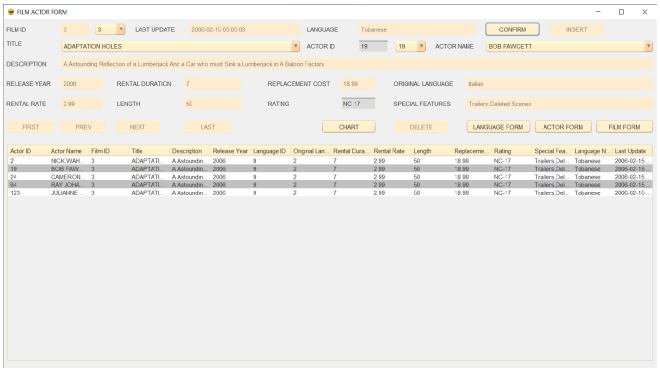


Figure 6.8 The film actor form is in editing state

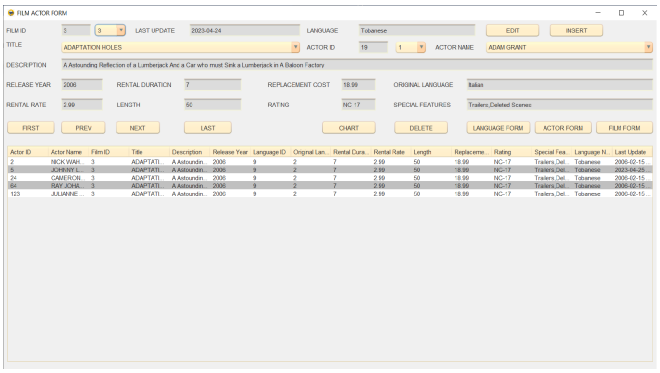
Step 5

In **FilmActorForm.java**, double click on EDIT button to create its event listener:

```
1     private void
2 jbEditActionPerformed(java.awt.event.ActionEvent
3 evt) {
          FilmActor_Utils.edit_handler(this);
      }
```

Step 6

Run the project. Choose **film_id** using **jcbFilmID** or **jcbTitle** combobox (in this case, **film_id = 3**). Then, choose available **actor_id** relating to chosen **film_id** using **jcbActorID** or **jcbActorName** (in this case, **actor_id = 19**). Then, click on EDIT button as shown in Figure 6.8.

	Then, choose new **actor_id** using **jcbActorID** or **jcbActorName** (in this case, **actor_id = 5**). Click on CONFIRM button. The edited row had been saved into **film_actor** table as shown in Figure 6.9.  **Figure 6.9** The edited row had been saved into database

INSERTING NEW RECORD
INSERTING NEW RECORD

Step 1	In **FilmActor_Utils** class, define a method named **insert_row()**. It inserts a new row into the **film_actor** table. The method takes in a **FilmActorForm** object as a parameter and uses a **HashMap** to read the input data from the form. The **film_id** and **act_id_new** values are then extracted from the input data. The SQL insert statement is defined as a multi-line string using the new Java text blocks feature, and it inserts a new row into the **film_actor** table with the **film_id** and **actor_id** values. The **getConnection()** method returns a Connection object to the database. A **PreparedStatement** object is then created using the SQL statement, and the **film_id** and **actor_id** values are set using the **setInt()** method. Finally, the **executeUpdate()** method is called on the **PreparedStatement** object to execute the SQL statement and insert the new row into the **film_actor** table. If a **SQLException** is thrown during the process, the method catches it, logs it, and displays an error message to the user. 1 `//Inserts new row into film_actor table` 2

```
    private static void insert_row(FilmActorForm frm) throws
SQLException{
        HashMap<String, String> input_data = read_inputs(frm);
        int film_id =
Integer.parseInt(input_data.get("film_id"));
        int act_id_new =
Integer.parseInt(input_data.get("act_id_new"));

        // SQL insert statement
        String sql = """
            INSERT INTO film_actor(film_id, actor_id)
                VALUES(?, ?)""";

        try(Connection conn = getConnection();
         PreparedStatement pstmt = conn.prepareStatement(sql)){

            //Creates a FilmActor object three-params constructor
            FilmActor obj = new FilmActor(act_id_new, film_id,
new Timestamp(System.currentTimeMillis()));
            pstmt.setInt(1,obj.getFilmID());
            pstmt.setInt(2,obj.getActorID());

            //Executes the sql insert statement
            pstmt.executeUpdate();
        } catch (SQLException ex) {

Logger.getLogger(FilmActorForm.class.getName()).log(Level.SEVERE,
"Database error", ex);
            JOptionPane.showMessageDialog(frm, "Error: Database
error\n" + ex.getMessage());
        }
    }
```

Step 2

Still in **FilmActor_Utils.java**, define **insert_actual()** and **insert_handler()** methods.

- **insert_actual(FilmActorForm frm)**: This method is a helper method that is called by the **insert_handler()** method. It calls the **insert_row()** method to perform the actual insertion of a new row into the **film_actor** table. If the insertion is successful, it calls the **refresh_controls()** method to refresh the table and comboboxes.

- **insert_handler(FilmActorForm frm)**: This method handles the insertion of a new row into the **film_actor** table in response to the user clicking on the "INSERT" button in a **FilmActorForm** object. It checks the text of the "INSERT" button to see if it currently displays "INSERT" or "CONFIRM". If it displays "INSERT", the method changes the text to "CONFIRM", disables the "JBEdit" button, and disables all controls on the form. If the

"INSERT" button displays "CONFIRM", the method calls the **insert_actual()** method to actually insert the new row into the film_actor table. This method also refreshes the table and comboboxes. If a **SQLException** is thrown during the insertion process, an error message is displayed to the user. After the insertion is complete, the method changes the text of the "INSERT" button back to "INSERT", enables the "JBEdit" button, and re-enables all controls on the form.

```
    private static void
insert_actual(FilmActorForm frm){
        try{
            insert_row(frm);

            //Refreshes table and comboboxes
            refresh_controls(frm);

        }catch(SQLException ex){
            JOptionPane.showMessageDialog(frm,
ex.getMessage(),

 "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    public static void
insert_handler(FilmActorForm frm){

 if(frm.getJBInsert().getText().equals("INSERT")
){

frm.getJBInsert().setText("CONFIRM");

            //Disables jbEdit
            frm.getJBEdit().setEnabled(false);

            // Disables controls
            enable_controls(false, frm);

            // Enables
            frm.getJBInsert().setEnabled(true);
        }

        else {
            frm.getJBInsert().setText("INSERT");

            // Actual insertion
            insert_actual(frm);

            //Enables jbEdit
            frm.getJBEdit().setEnabled(true);
```

```
            //Enables controls
            enable_controls(true, frm);
        }
    }
```

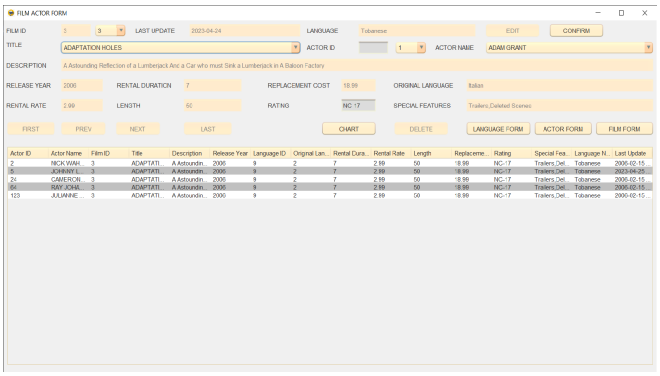


Figure 6.10 When user clicks on INSERT button, the film actor form will be in state of insertion

Step 3

In **FilmActorForm.java**, double click on INSERT button to create its event listener:

```
    private void jbInsertActionPerformed(java.awt.event.ActionEvent evt) {
        FilmActor_Utils.insert_handler(this);
    }
```

Step 4

Run the project. Choose **film_id** using **jcbFilmID** or **jcbTitle** combobox (in this case, **film_id = 3**). Click on INSERT button. You will see the state of film actor form when insertion is in progress as shown in Figure 6.10.

Then, choose **actor_id** that is not available relating to chose **film_id** using **jcbActorID** or **jcbActorName** (in this case, **actor_id = 3**). Then, click CONFIRM button to save the new record into **film_actor** table as shown in Figure 6.11.

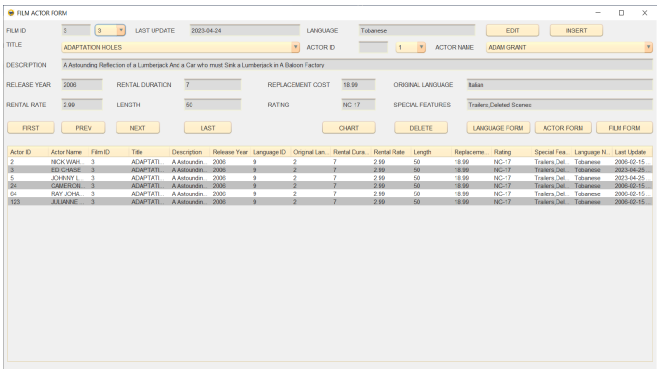


Figure 6.11 The new data had been saved into **film_actor** table

DELETING RECORD
DELETING RECORD

Step 1

Then in **FilmActor_Utils** class, define **delete_handler()** method. It handl deletion of a row from the **film_actor** table. It first retrieves the valu **film_id** and **actor_id** from the corresponding combo boxes in **FilmActorForm** object passed as a parameter. It then creates a confirr dialog box to confirm whether the user really wants to delete the row.

If the user chooses to delete the row, the method constructs a SQL DE statement that deletes the row with the specified **film_id** and **actor_id**. It **PreparedStatement** to avoid SQL injection attacks. The method then exe the DELETE statement and refreshes the table and combo boxes by calli **refresh_controls()** method.

If an SQL exception occurs during the execution of the DELETE statemer method displays an error message in a dialog box.

```
1     public static void delete_handler(FilmActorForm frm){
2         int dialogButton = JOptionPane.YES_NO_OPTION;
3         int film_id =
4 Integer.parseInt(String.valueOf(frm.getJCBFilmID().getSelectedItem(
5         int actor_id =
6 Integer.parseInt(String.valueOf(frm.getJCBActorID().getSelectedItem
7 
8         String message = String.format("Are you sure you want to de
9 the row (Film ID: %d, Actor ID: %d)", film_id, actor_id);
10        int answer = JOptionPane.showConfirmDialog(frm, message,
11 "DELETING ROW OF DATA", dialogButton);
12 
13        if(answer == JOptionPane.YES_OPTION){
14            String query = """
15 
```

```
                DELETE FROM film_actor WHERE film_id = ? AND actor_
?""";
            try(Connection conn = getConnection();
                PreparedStatement ps = conn.prepareStatement(query)
                // Use PreparedStatement to avoid SQL injection att
                ps.setInt(1, film_id);
                ps.setInt(2, actor_id);
                ps.executeUpdate();

                // Refresh table and comboboxes
                refresh_controls(frm);

            } catch (SQLException ex){
                JOptionPane.showMessageDialog(frm, ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
            }
        }
    }
```

Step 2	In **FilmActorForm.java**, double click on DELETE button to generate its listener: `private void jbDeleteActionPerformed(java.awt.event.ActionEvent evt) { FilmActor_Utils.delete_handler(this); }`
Step 3	Run the project. Choose **film_id** using **jcbFilmID** or **jcbFilm** combobo choose **actor_id** using **jcbActorID** or **jcbActorName** combobox Then, Cl DELETE button. The corresponding row of data had been deleted database.

PLOTTING CHART
PLOTTING CHART

Step 1	Create a new **JFrame** and save it as **Charts_FilmActor.java**.

Step 2	In **Charts_FilmActor.java**, add six **JPanels** and set their corresp **Variable Name** as **jPanel1**, **jPanel2**, **jPanel3**, **jPanel4**, **jPanel5**, and **jP** Then, add getter method for each object as follows:
	
	`//Getter method for jPanel1`
	`public JPanel getJPanel1(){`
	`  return this.jPanel1;`
	`}`
	
	`//Getter method for jPanel2`
	`public JPanel getJPanel2(){`
	`  return this.jPanel2;`
	`}`
	
	`//Getter method for jPanel3`
	`public JPanel getJPanel3(){`
	`  return this.jPanel3;`
	`}`
	
	`//Getter method for jPanel4`
	`public JPanel getJPanel4(){`
	`  return this.jPanel4;`
	`}`
	
	`//Getter method for jPanel5`
	`public JPanel getJPanel5(){`
	`  return this.jPanel5;`
	`}`
	
	`//Getter method for jPanel6`
	`public JPanel getJPanel6(){`
	`  return this.jPanel6;`
	`}`

Step 3	In **FilmActor_Utils** class, define six new methods. These methods are u draw different types of charts on a Java Swing GUI using **JFreeChart** l Here's a brief explanation of each method: 1. **draw_pie_chart_film_by_actor_name(Charts_FilmActor frm, J jp)**: This method draws a pie chart on the JPanel **jp** that shows t 10 film distribution by actor name. The **DefaultPieDataset** is c using the **create_pie_dataset()** method and then passed t **draw_piechart_with_dataset()** method along with the chart tit the parent frame **frm**. 2. **draw_bar_chart_actor_by_avg_rental_rate(Charts_FilmActor JPanel jp)**: This method draws a bar chart on the JPanel **jp** that

the top 10 actor name distribution by average rental rate **DefaultCategoryDataset** is created using the **create_bar_dat** method and then passed to the **draw_barchart_with_dataset()** n along with the chart title, X-axis label, Y-axis label, and the parent frm.

3. **draw_bar_chart_actor_by_avg_replacement_cost(Charts_FilmA frm, JPanel jp)**: This method draws a bar chart on the JPanel **j** shows the top 10 actor name distribution by average replacemen The **DefaultCategoryDataset** is created using **create_bar_dataset()** method and then passed to **draw_barchart_with_dataset()** method along with the chart ti axis label, Y-axis label, and the parent frame **frm**.
4. **draw_pie_chart_film_by_rating(Charts_FilmActor frm, JPane** This method draws a pie chart on the JPanel **jp** that shows th distribution by rating. The **DefaultPieDataset** is created usir **create_pie_dataset()** method and then passed to **draw_piechart_with_dataset()** method along with the chart tit the parent frame **frm**.
5. **draw_bar_chart_rating_by_avg_rental_rate(Charts_FilmActor JPanel jp)**: This method draws a bar chart on the JPanel **jp** that the rating distribution by average rental rate. **DefaultCategoryDataset** is created using the **create_bar_dat** method and then passed to the **draw_barchart_with_dataset()** n along with the chart title, X-axis label, Y-axis label, and the parent frm.
6. **draw_bar_chart_rating_by_avg_replacement_cost(Charts_Film frm, JPanel jp)**: This method draws a bar chart on the JPanel **j** shows the rating distribution by average replacement cost **DefaultCategoryDataset** is created using the **create_bar_dat** method and then passed to the **draw_barchart_with_dataset()** n along with the chart title, X-axis label, Y-axis label, and the parent **frm**.

```
    private static void
draw_pie_chart_film_by_actor_name(Charts_FilmActor frm, JPanel
jp){
        jp.setPreferredSize(new Dimension(jp.getWidth(),
jp.getHeight()));
        DefaultPieDataset dataset =
create_pie_dataset(Query_FilmActor.get_sql_film_actor_dist(),
"Number", "actor_name");

        //Draws piechart film distribution by actor name
        draw_piechart_with_dataset(frm, jp, dataset, "TOP 10
FILM DISTRIBUTION BY ACTOR NAME");
    }

    private static void
draw_bar_chart_actor_by_avg_rental_rate(Charts_FilmActor frm,
```

```
JPanel jp){
        jp.setPreferredSize(new Dimension(jp.getWidth(),
jp.getHeight()));

        DefaultCategoryDataset dataset =
create_bar_dataset(Query_FilmActor.get_sql_film_actor_dist(),
"avg_rental_rate", "actor_name");

        //Draws barchart actor name distribution by average
rental rate
        draw_barchart_with_dataset(frm, jp, dataset, "TOP 10
ACTOR NAME DISTRIBUTION BY AVERAGE RENTAL RATE", "ACTOR NAME",
"AVG RENTAL RATE");
    }

    private static void
draw_bar_chart_actor_by_avg_replacement_cost(Charts_FilmActor
frm, JPanel jp){
        jp.setPreferredSize(new Dimension(jp.getWidth(),
jp.getHeight()));

        DefaultCategoryDataset dataset =
create_bar_dataset(Query_FilmActor.get_sql_film_actor_dist(),
"avg_replacement_cost", "actor_name");

        //Draws barchart actor name distribution by average
replacement cost
        draw_barchart_with_dataset(frm, jp, dataset, "TOP 10
ACTOR NAME DISTRIBUTION BY AVERAGE REPLACEMENT COST", "ACTOR
NAME", "AVG REPLACEMENT COST");
    }

    private static void
draw_pie_chart_film_by_rating(Charts_FilmActor frm, JPanel jp)
{
        jp.setPreferredSize(new Dimension(jp.getWidth(),
jp.getHeight()));
        DefaultPieDataset dataset =
create_pie_dataset(Query_FilmActor.get_sql_film_rating_dist(),
"Number", "rating");

        //Draws piechart film distribution by rating
        draw_piechart_with_dataset(frm, jp, dataset, "THE FILM
DISTRIBUTION BY RATING");
    }

    private static void
draw_bar_chart_rating_by_avg_rental_rate(Charts_FilmActor frm,
JPanel jp){
        jp.setPreferredSize(new Dimension(jp.getWidth(),
jp.getHeight()));


```

```
        DefaultCategoryDataset dataset =
create_bar_dataset(Query_FilmActor.get_sql_film_rating_dist(),
"avg_rental_rate", "rating");

        //Draws barchart rating distribution by average rental
rate
        draw_barchart_with_dataset(frm, jp, dataset, "THE
RATING DISTRIBUTION BY AVERAGE RENTAL RATE", "RATING", "AVG
RENTAL RATE");
    }

    private static void
draw_bar_chart_rating_by_avg_replacement_cost(Charts_FilmActor
frm, JPanel jp){
        jp.setPreferredSize(new Dimension(jp.getWidth(),
jp.getHeight()));

        DefaultCategoryDataset dataset =
create_bar_dataset(Query_FilmActor.get_sql_film_rating_dist(),
"avg_replacement_cost", "rating");

        //Draws barchart rating distribution by average
replacement cost
        draw_barchart_with_dataset(frm, jp, dataset, "THE
RATING DISTRIBUTION BY AVERAGE REPLACEMENT COST", "RATING",
"AVG REPLACEMENT COST");
    }
```

Step 4

In **FilmActor_Utils** class, define a new method named **jbchart_handler()**

```
    public static void
jbchart_handler(Charts_FilmActor frm){
        //Draws piechart film distribution by
actor name
        draw_pie_chart_film_by_actor_name(frm,
frm.getJPanel1());

        //Draws barchart actor name distribution
by average rental rate

draw_bar_chart_actor_by_avg_rental_rate(frm,
frm.getJPanel2());

        //Draws barchart actor name distribution
by replacement cost

draw_bar_chart_actor_by_avg_replacement_cost(frm,
frm.getJPanel3());

        //Draws piechart film distribution by
rating
```

```
        draw_pie_chart_film_by_rating(frm,
frm.getJPanel4());

        //Draws barchart rating distribution by
average rental rate

draw_bar_chart_rating_by_avg_rental_rate(frm,
frm.getJPanel5());

        //Draws barchart rating distribution by
average replacement cost

draw_bar_chart_rating_by_avg_replacement_cost(frm,
frm.getJPanel6());
    }
```

Step 5

In **FilmActorForm**, double click on **jbChart** button to define its event liste

```
1  private void
2  jbChartActionPerformed(java.awt.event.ActionEvent
3  evt) {
4          Charts_FilmActor frm1 = new
5  Charts_FilmActor();
6          frm1.setLocationRelativeTo(null);
7          frm1.setTitle("SIX DISTRIBUTIONS IN FILM-
   ACTOR TABLE");
           frm1.setVisible(true);
           FilmActor_Utils.jbchart_handler(frm1);
       }
```

It is an event handler method for a button with the name **jbChart**. Whe button is clicked, it creates a new instance of the **Charts_FilmActor** cla sets its location to the center of the screen, sets the title of the window t DISTRIBUTIONS IN FILM-ACTOR TABLE", and sets the window to visible. After creating and showing the window, it calls the static n jbchart_handler of the **FilmActor_Utils** class, passing the **Charts_Film** instance as a parameter. The **jbchart_handler()** method is responsib drawing six charts related to the film-actor table on six different **J** components of the **Charts_FilmActor** window.

Step 6

Run the project. Click on CHART button on the form. You will see the six displayed on the panels as shown in Figure 6.12.

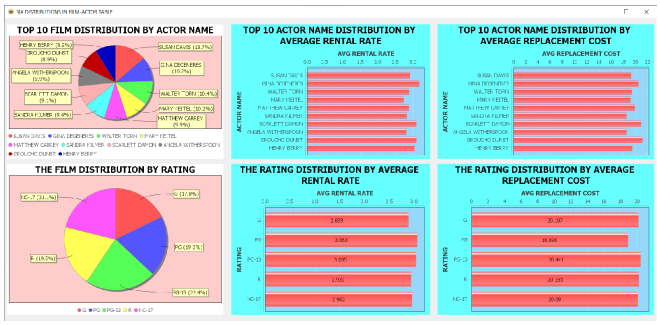


Figure 6.12 The top 10 film distribution by actor name, top 10 actor na distribution by average rental rate, top 10 actor name distribution by ave replacement cost, film distribution by rating, rating distribution by aver rental rate, and rating distribution by average replacement cost

This is the full version of **FilmActor_Utils.java**:

```
package sakila;
import java.awt.Dimension;
import java.util.logging.Level;
import java.util.logging.Logger;
import java.sql.*;
import java.util.logging.Level;
import java.util.logging.Logger;
import java.sql.*;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashMap;
import java.util.LinkedHashMap;
import java.util.List;
import java.util.Map;
import java.util.Objects;
import javax.swing.JComboBox;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.event.TableModelEvent;
import javax.swing.table.DefaultTableModel;
import javax.swing.table.TableModel;
import org.jfree.data.category.DefaultCategoryDataset;
import org.jfree.data.general.DefaultPieDataset;

public class FilmActor_Utils extends Utility{
    public static final int FIRST_INDEX = 0;
    public static final int INVALID_INDEX = -1;

    private static int currentIndex = FIRST_INDEX;
    private static final String SQL_ID = Query_FilmActor.get_sql_film_actor_
" WHERE fa.film_id = ?";
```

```
    //Creates film_actor table
    public static void create_film_actor_table() {
        try (Connection conn = getConnection()) {
            Statement stmt = conn.createStatement();
            stmt.addBatch(Query_FilmActor.get_sql_film_actor());
            stmt.executeBatch();

            String message = String.format("Successfully creates film_actor t
            JOptionPane.showMessageDialog(null, message,
            "INFORMATION",JOptionPane.INFORMATION_MESSAGE);

        } catch (SQLException ex) {
            JOptionPane.showMessageDialog(null, ex.getMessage(),
                "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    //Populates film_actor table with some rows of data
    public static void populate_film_actor_table(){
        try(Connection conn = getConnection()){
            String sql = """
                  INSERT INTO film_actor(actor_id, film_id, last_update)
                  VALUES(?, ?, ?)""";

            PreparedStatement ps1 = conn.prepareStatement(sql);
            //Creates a new FilmActor class with default constructor
            FilmActor obj1 = new FilmActor();
            ps1.setInt(1,obj1.getActorID());
            ps1.setInt(2, obj1.getFilmID());
            ps1.setTimestamp(3, obj1.getLastUpdate());

            PreparedStatement ps2 = conn.prepareStatement(sql);
            //Creates a new FilmActor class with three-params constructor
            FilmActor obj2 = new FilmActor(2, 2, new
Timestamp(System.currentTimeMillis()));
            ps2.setInt(1,obj2.getActorID());
            ps2.setInt(2, obj2.getFilmID());
            ps2.setTimestamp(3, obj2.getLastUpdate());

            ps1.executeUpdate();
            ps2.executeUpdate();

        }catch(SQLException ex){
            JOptionPane.showMessageDialog(null, ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    //Reads the content of joined film_actor, film, and language tables
    public static void read_film_actor_table(){
        try(Connection conn = getConnection()){
            Statement stmt = conn.createStatement();
```

```
            ResultSet rs =
stmt.executeQuery(Query_FilmActor.get_sql_film_actor_joint());
            while(rs.next()){
                int actor_id = rs.getInt("actor_id");
                int film_id = rs.getInt("film_id");
                String title = rs.getString("title");
                String description = rs.getString("description");
                int year = rs.getInt("release_year");
                int lang_id = rs.getInt("language_id");
                int ori_lang_id = rs.getInt("original_language_id");
                int duration = rs.getInt("rental_duration");
                double rate = rs.getDouble("rental_rate");
                int length = rs.getInt("length");
                double cost = rs.getDouble("replacement_cost");
                String rating = rs.getString("rating");
                String features = rs.getString("special_features");
                String lang_name = rs.getString("language_name");
                String actor_name = rs.getString("actor_name");
                Timestamp lu = rs.getTimestamp("last_update");

                //Creates an FilmActor object using sixteen-params constructo
                FilmActor obj = new FilmActor(actor_id, film_id, title, descr
year, lang_id, ori_lang_id,
                    duration, rate, length, cost, rating, features, lang_name
actor_name, lu);
                System.out.println(obj);
            }
            rs.close();
            stmt.close();

        }catch(SQLException ex){
            JOptionPane.showMessageDialog(null, ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    private static ArrayList<FilmActor> get_film_actor_list(FilmActorForm fr
sql, String item){
        ArrayList<FilmActor> list = new ArrayList<>();
        Connection conv = null;

        try(Connection conn = getConnection();
            PreparedStatement ps = conn.prepareStatement(sql)){
            if (item.equalsIgnoreCase("none")==false) {
                 ps.setString(1,item);
            }
            ResultSet rs = ps.executeQuery();

            FilmActor obj;
            while(rs.next()){
                //Using sixteen-params constructor
                obj = new FilmActor(rs.getInt("actor_id"), rs.getInt("film_id
                    rs.getString("title"),
```

```
                    rs.getString("description"), rs.getInt("release_year"),
                    rs.getInt("language_id"), rs.getInt("original_language_id
                    rs.getInt("rental_duration"), rs.getDouble("rental_rate")
                    rs.getInt("length"), rs.getDouble("replacement_cost"),
                    rs.getString("rating"), rs.getString("special_features"),
                    rs.getString("language_name"), rs.getString("actor_name")
                        rs.getTimestamp("last_update"));

                list.add(obj);
            }
        }catch (SQLException ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
                "ERROR",JOptionPane.ERROR_MESSAGE);
        }
        return list;
    }

    private static void show_table_film_actor(FilmActorForm frm, ArrayList<Fi
list) throws SQLException{
        DefaultTableModel model = new DefaultTableModel(0,0);

        String header[] = {"Actor ID", "Actor Name", "Film ID", "Title",
"Description", "Release Year",
            "Language ID", "Original Language", "Rental Duration", "Rental Ra
"Length",
            "Replacement Cost", "Rating", "Special Features", "Language Name"
Update"};

        model.setColumnIdentifiers(set_column_header(frm.getJTFilmActor(), he
        frm.getJTFilmActor().setModel(model);

        Object[] row = new Object[16];

        for(int i=0; i<list.size(); i++){
            row[0] = list.get(i).getActorID();
            row[1] = list.get(i).getActorName();
            row[2] = list.get(i).getFilmID();
            row[3] = list.get(i).getTitle();
            row[4] = list.get(i).getDescription();
            row[5] = list.get(i).getReleaseYear();
            row[6] = list.get(i).getLanguageID();
            row[7] = list.get(i).getOriginalLanguageID();
            row[8] = list.get(i).getRentalDuration();
            row[9] = list.get(i).getRentalRate();
            row[10] = list.get(i).getLength();
            row[11] = list.get(i).getReplacementCost();
            row[12] = list.get(i).getRating();
            row[13] = list.get(i).getSpecialFeatures();
            row[14] = list.get(i).getLanguageName();
            row[15] = list.get(i).getLastUpdate();

            model.addRow(row);
        }
    }
```

```
    public static void refresh_controls(FilmActorForm frm){
        frm.setLocationRelativeTo(null);
        frm.setTitle("FILM ACTOR FORM");

        //Shows the content of film table and populates combobox
        try{
            //Makes alternating color for table rows
            table_renderer(frm.getJTFilmActor());

            //Populates table
            ArrayList<FilmActor> list = get_film_actor_list(frm,
Query_FilmActor.get_sql_film_actor_joint() + " ORDER BY f.film_id", "none");
            show_table_film_actor(frm, list);

            //Populates jcbFilmID
            String sql_id = "SELECT film_id FROM film ORDER BY film_id";
            populate_combobox(sql_id, frm.getJCBFilmID(), frm);

            //Populates getJCBTitle()
            String sql_title = "SELECT DISTINCT title FROM film ORDER BY titl
            populate_combobox(sql_title, frm.getJCBTitle(), frm);

            //Populates jcbActorID
            String sql_act_id = "SELECT actor_id FROM actor ORDER BY actor_id
            populate_combobox(sql_act_id, frm.getJCBActorID(), frm);

            //Populates jcbActorName
            String sql_act_name = "SELECT DISTINCT CONCAT(first_name,' ',last
FROM actor ORDER BY CONCAT(first_name,' ',last_name)";
            populate_combobox(sql_act_name, frm.getJCBActorName(), frm);

        }catch (SQLException ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
            "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    private static void clear_controls(FilmActorForm frm){
        frm.getJTFFilmID().setText("");
        frm.getJTFDescription().setText("");
        frm.getJTFReleaseYear().setText("");
        frm.getJTFLanguageName().setText("");
        frm.getJTFOriginalLanguageName().setText("");
        frm.getJTFRentalDuration().setText("");
        frm.getJTFRentalRate().setText("");
        frm.getJTFLength().setText("");
        frm.getJTFReplacementCost().setText("");
        frm.getJTFSpecialFeatures().setText("");
        frm.getJTFLastUpdate().setText("");
    }

   //Displays film data result row by row
```

```
    private static <T> void display_film_data(FilmActorForm frm, String sql,
        try(Connection conn = getConnection()){
            PreparedStatement ps = conn.prepareStatement(sql);
            ps.setObject(1,item);
            ResultSet rs = ps.executeQuery();

            if (!rs.next()) {
                // no row found, clear the form fields
                clear_controls(frm);
                return;
            }

            do{
                frm.getJTFFilmID().setText(String.valueOf(rs.getInt("film_id"
                frm.getJTFDescription().setText(rs.getString("description"));

frm.getJTFReleaseYear().setText(String.valueOf(rs.getInt("release_year")));
                frm.getJTFLanguageName().setText(rs.getString("language_name"

frm.getJTFRentalDuration().setText(String.valueOf(rs.getInt("rental_duration"

frm.getJTFRentalRate().setText(String.valueOf(rs.getDouble("rental_rate")));
                frm.getJTFLength().setText(String.valueOf(rs.getInt("length")

frm.getJTFReplacementCost().setText(String.valueOf(rs.getDouble("replacement_
                frm.getJTFSpecialFeatures().setText(rs.getString("special_fea
                frm.getJTFRating().setText(rs.getString("rating"));

frm.getJTFLastUpdate().setText(String.valueOf(rs.getDate("last_update")));

                // Determines item selected from jcbFilmID
                find_combo_value_selected(frm.getJCBFilmID(), rs.getInt("film

                // Determines item selected from jcbTitle
                find_combo_value_selected(frm.getJCBTitle(), rs.getString("ti

               // Finds original language name
                int ori_lang_id = rs.getInt("original_language_id");
                if(ori_lang_id != 0) {
                    Object lang_name = get_val_from_database("language", "nam
"language_id", ori_lang_id);

frm.getJTFOriginalLanguageName().setText(String.valueOf(lang_name));

                }

            }while(rs.next());

            rs.close();
            ps.close();
        }catch(SQLException ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
```

```
        }
    }

    public static void jcbFilm_handler(FilmActorForm frm, JComboBox<String> j
        Object item = jcb.getSelectedItem();
        String sql = "";
        if (jcb.equals(frm.getJCBFilmID())) {
            sql = Query_Film.get_sql_film_joint() + " WHERE f.film_id = ?";

            try{
                //Filters jtFilmActor
                ArrayList<FilmActor> list = get_film_actor_list(frm,
Query_FilmActor.get_sql_film_actor_joint() + " WHERE fa.film_id = ?",
String.valueOf(item));
                show_table_film_actor(frm, list);
            }catch (SQLException ex){
                JOptionPane.showMessageDialog(frm, ex.getMessage(),
                "ERROR",JOptionPane.ERROR_MESSAGE);
            }
        } else if (jcb.equals(frm.getJCBTitle())) {
            sql = Query_Film.get_sql_film_joint() + " WHERE f.title = ?";

            try{
                //Filters jtFilmActor
                ArrayList<FilmActor> list = get_film_actor_list(frm,
Query_FilmActor.get_sql_film_actor_joint() + " WHERE f.title = ?",
String.valueOf(item));
                show_table_film_actor(frm, list);
            }catch (SQLException ex){
                JOptionPane.showMessageDialog(frm, ex.getMessage(),
                "ERROR",JOptionPane.ERROR_MESSAGE);
            }
        }

        display_film_data(frm, sql, item);
    }

    private static <T> void display_actor_data(FilmActorForm frm, String sql,
T item2) {
        try(Connection conn = getConnection()){
            PreparedStatement ps = conn.prepareStatement(sql);
            ps.setObject(1,item1);
            ps.setObject(2,item2);
            ResultSet rs = ps.executeQuery();

            if (!rs.next()) {
                // no row found, clear the form fields
                frm.getJTFLastUpdate().setText("");
                return;
            }

            do {
                frm.getJTFActorID().setText(rs.getString("actor_id"));
```

```
                frm.getJTFLastUpdate().setText(rs.getString("last_update"));

                // Determines item selected from jcbActorID
                find_combo_value_selected(frm.getJCBActorID(), rs.getInt("act

                // Determines item selected from jcbActorName
                String actor_name = rs.getString("first_name") + " " +
rs.getString("last_name");
                find_combo_value_selected(frm.getJCBActorName(), actor_name);
            } while(rs.next());

            rs.close();
            ps.close();
        } catch(SQLException ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(), "ERROR",
JOptionPane.ERROR_MESSAGE);
        }
    }

    public static void jcbActor_handler(FilmActorForm frm, JComboBox<String>
JComboBox<String> jcb2) {
        Object item1 = jcb1.getSelectedItem();
        Object item2 = jcb2.getSelectedItem();
        String sql = "";
        if (jcb1.equals(frm.getJCBActorID())) {
            sql = Query_FilmActor.get_sql_film_actor_joint() + " WHERE fa.act
AND fa.film_id = ?";
        } else if (jcb1.equals(frm.getJCBActorName())) {
            sql = Query_FilmActor.get_sql_film_actor_joint() + " WHERE
CONCAT(a.first_name,' ',a.last_name) = ? AND fa.film_id = ?";
        }

        display_actor_data(frm, sql, item1, item2);
    }

    public static void show_first_row(FilmActorForm frm){
        String item = String.valueOf(frm.getJCBFilmID().getItemAt(FIRST_INDE
        display_film_data(frm, SQL_ID, item);
        currentIndex = FIRST_INDEX;
    }

    public static void show_last_row(FilmActorForm frm){
        int endIndex = frm.getJCBFilmID().getItemCount() - 1;
        String item = String.valueOf(frm.getJCBFilmID().getItemAt(endIndex));
        display_film_data(frm, SQL_ID, item);
        currentIndex = endIndex;
    }

    public static void show_prev_row(FilmActorForm frm){
        currentIndex--;
        if(currentIndex < FIRST_INDEX){
            currentIndex = FIRST_INDEX;
            return;
```

```
        }
        String item = String.valueOf(frm.getJCBFilmID().getItemAt(currentInde
        display_film_data(frm, SQL_ID, item);
    }

    public static void show_next_row(FilmActorForm frm){
        int endIndex = frm.getJCBFilmID().getItemCount() - 1;
        currentIndex++;
        if(currentIndex > endIndex){
            currentIndex = endIndex;
            return;
        }
        String item = String.valueOf(frm.getJCBFilmID().getItemAt(currentInde
        display_film_data(frm, SQL_ID, item);
    }

    public static void mouse_pressed_handler(FilmActorForm frm) {
        Objects.requireNonNull(frm, "frm must not be null");

        int selectedIndex = frm.getJTFilmActor().getSelectedRow();
        if (selectedIndex == -1) {
            JOptionPane.showMessageDialog(frm, "Please select a row to view 
data.",
                "No row selected", JOptionPane.INFORMATION_MESSAGE);
            return;
        }

        try (Connection conn = getConnection()) {
            String item1 =
String.valueOf(frm.getJTFilmActor().getModel().getValueAt(selectedIndex, 0));
            String item2 =
String.valueOf(frm.getJTFilmActor().getModel().getValueAt(selectedIndex, 2));

            // Displays film actor data
            String sql = Query_FilmActor.get_sql_film_actor_joint() + " WHERE
fa.actor_id = ? AND fa.film_id = ?";
            display_actor_data(frm, sql, item1, item2);

        } catch (SQLException ex) {
            Logger.getLogger(FilmActorForm.class.getName()).log(Level.SEVERE,
displaying film actor data", ex);
            String message = "Error displaying film actor data: " + ex.getMes
            String stackTrace = Arrays.toString(ex.getStackTrace());
            JOptionPane.showMessageDialog(frm, message + "\n\n" + stackTrace,
JOptionPane.ERROR_MESSAGE);
        }
    }

    //Updates row of data in film_actor tabel by film_id and actor_id
    public static void update_row_by_film_actor_id(FilmActorForm frm, int fil
act_id, int act_id_new) throws SQLException{
        Connection conn = getConnection();
        ResultSet rs = null;
```

```
        String query_id = "SELECT film_id FROM film WHERE film_id = ?";
        String update_query = """
            UPDATE film_actor SET film_id = ?, actor_id = ?
            WHERE film_id = ? AND actor_id = ?""";
        try(PreparedStatement idPs = conn.prepareStatement(query_id,
                    ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_READ_ONL
        PreparedStatement updatePS = conn.prepareStatement(update_query,
                ResultSet.TYPE_SCROLL_SENSITIVE, ResultSet.CONCUR_UPDATABLE))
        {
            idPs.setInt(1,film_id);
            if(!idPs.execute()){
                String message = "Can't find film_id " + film_id;

                JOptionPane.showMessageDialog(frm, message,
                    "ERROR",JOptionPane.ERROR_MESSAGE);
            } else{
                rs = idPs.getResultSet();
                rs.next();

                //Creates a FilmActor object using three-params constructor
                FilmActor obj = new FilmActor(act_id, film_id, new
Timestamp(System.currentTimeMillis()));
                updatePS.setInt(1, obj.getFilmID());
                updatePS.setInt(2, act_id_new);
                updatePS.setInt(3, obj.getFilmID());
                updatePS.setInt(4, obj.getActorID());

                updatePS.executeUpdate();
                rs.close();
                updatePS.close();
                idPs.close();
                conn.close();
            }
        }catch(SQLException ex){
            Logger.getLogger(FilmActorForm.class.getName()).log(Level.SEVERE,
updating film-actor data", ex);
            String message = "Error updating film-actor data: " + ex.getMessa
            String stackTrace = Arrays.toString(ex.getStackTrace());
            JOptionPane.showMessageDialog(null, message + "\n\n" + stackTrace
"ERROR", JOptionPane.ERROR_MESSAGE);
        }catch(java.lang.NumberFormatException ex){
            Logger.getLogger(FilmActorForm.class.getName()).log(Level.SEVERE,
Input", ex);
            String message = "Invalid Input: " + ex.getMessage();
            String stackTrace = Arrays.toString(ex.getStackTrace());
            JOptionPane.showMessageDialog(null, message + "\n\n" + stackTrace
"ERROR", JOptionPane.ERROR_MESSAGE);
        }
    }

    private static HashMap<String, String> read_inputs(FilmActorForm frm) {
        HashMap<String, String> input_data = new HashMap<>();
        String film_id = String.valueOf(frm.getJCBFilmID().getSelectedItem())
```

```
        String act_id = frm.getJTFActorID().getText();
        String act_id_new = String.valueOf(frm.getJCBActorID().getSelectedIte

        // Validate user input
        int film_id_int = 0;
        try {
            film_id_int = Integer.parseInt(film_id);
            if (film_id_int <= 0) {
                throw new IllegalArgumentException("Film ID cannot be negati
zero");
            }
        } catch (NumberFormatException ex) {
            JOptionPane.showMessageDialog(frm, "Invalid Film ID: " + film_id,
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        } catch (IllegalArgumentException ex) {
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        }

        if (act_id == null) {
            JOptionPane.showMessageDialog(frm, "Actor ID cannot be empty",
            "Error", JOptionPane.ERROR_MESSAGE);
            throw new IllegalArgumentException("Actor ID cannot be empty");
        }

        if (act_id_new == null || act_id_new.isEmpty()) {
            JOptionPane.showMessageDialog(frm, "New Actor ID cannot be empty'
            "Error", JOptionPane.ERROR_MESSAGE);
            throw new IllegalArgumentException("Ner Actor ID cannot be empty'
        }

        input_data.put("film_id", film_id);
        input_data.put("act_id", act_id);
        input_data.put("act_id_new", act_id_new);

        return input_data;
    }

    private static void edit_actual(FilmActorForm frm){
        try{
            HashMap<String, String> input_data = read_inputs(frm);
            int film_id = Integer.parseInt(input_data.get("film_id"));
            int act_id = Integer.parseInt(input_data.get("act_id"));
            int act_id_new = Integer.parseInt(input_data.get("act_id_new"));

            System.out.println(film_id);
            System.out.println(act_id);
            System.out.println(act_id_new);

            update_row_by_film_actor_id(frm, film_id, act_id, act_id_new);
```

```
            //Refreshes all objects on form
            refresh_controls(frm);

        }catch(SQLException ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    private static void enable_controls(boolean state, FilmActorForm frm){
        frm.getJBFirst().setEnabled(state);
        frm.getJBPrev().setEnabled(state);
        frm.getJBNext().setEnabled(state);
        frm.getJBLast().setEnabled(state);
        frm.getJBInsert().setEnabled(state);
        frm.getJBDelete().setEnabled(state);
        frm.getJTFFilmID().setEnabled(state);
        frm.getJTFDescription().setEnabled(state);
        frm.getJTFReleaseYear().setEnabled(state);
        frm.getJTFLanguageName().setEnabled(state);
        frm.getJTFOriginalLanguageName().setEnabled(state);
        frm.getJTFRentalDuration().setEnabled(state);
        frm.getJTFRentalRate().setEnabled(state);
        frm.getJTFLength().setEnabled(state);
        frm.getJTFReplacementCost().setEnabled(state);
        frm.getJTFSpecialFeatures().setEnabled(state);
        frm.getJTFLastUpdate().setEnabled(state);
    }

    public static void edit_handler(FilmActorForm frm){
        if(frm.getJBEdit().getText().equals("EDIT")){
            frm.getJBEdit().setText("CONFIRM");

            // Disables controls
            enable_controls(false, frm);
        }

        else {
            frm.getJBEdit().setText("EDIT");

            // Actual editing
            edit_actual(frm);

            //Enables controls
            enable_controls(true, frm);
        }
    }

    //Inserts new row into film_actor table
    private static void insert_row(FilmActorForm frm) throws SQLException{
        HashMap<String, String> input_data = read_inputs(frm);
        int film_id = Integer.parseInt(input_data.get("film_id"));
        int act_id_new = Integer.parseInt(input_data.get("act_id_new"));
```

```
        // SQL insert statement
        String sql = """
            INSERT INTO film_actor(film_id, actor_id)
                VALUES(?, ?)""";

        try(Connection conn = getConnection();
         PreparedStatement pstmt = conn.prepareStatement(sql)){

            //Creates a FilmActor object three-params constructor
            FilmActor obj = new FilmActor(act_id_new, film_id, new
Timestamp(System.currentTimeMillis()));
            pstmt.setInt(1,obj.getFilmID());
            pstmt.setInt(2,obj.getActorID());

            //Executes the sql insert statement
            pstmt.executeUpdate();
        } catch (SQLException ex) {
            Logger.getLogger(FilmActorForm.class.getName()).log(Level.SEVERE,
"Database error", ex);
            JOptionPane.showMessageDialog(frm, "Error: Database error\n" +
ex.getMessage());
        }
    }

    private static void insert_actual(FilmActorForm frm){
        try{
            insert_row(frm);

            //Refreshes table and comboboxes
            refresh_controls(frm);

        }catch(SQLException ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    public static void insert_handler(FilmActorForm frm){
        if(frm.getJBInsert().getText().equals("INSERT")){
            frm.getJBInsert().setText("CONFIRM");

            //Disables jbEdit
            frm.getJBEdit().setEnabled(false);

            // Disables controls
            enable_controls(false, frm);

            // Enables
            frm.getJBInsert().setEnabled(true);
        }

        else {
```

```
            frm.getJBInsert().setText("INSERT");

            // Actual insertion
            insert_actual(frm);

            //Enables jbEdit
            frm.getJBEdit().setEnabled(true);

            //Enables controls
            enable_controls(true, frm);
        }
    }

    public static void delete_handler(FilmActorForm frm){
        int dialogButton = JOptionPane.YES_NO_OPTION;
        int film_id =
Integer.parseInt(String.valueOf(frm.getJCBFilmID().getSelectedItem()));
        int actor_id =
Integer.parseInt(String.valueOf(frm.getJCBActorID().getSelectedItem()));

        String message = String.format("Are you sure you want to delete the r
ID: %d, Actor ID: %d)", film_id, actor_id);
        int answer = JOptionPane.showConfirmDialog(frm, message, "DELETING RO
DATA", dialogButton);

        if(answer == JOptionPane.YES_OPTION){
            String query = """
                DELETE FROM film_actor WHERE film_id = ? AND actor_id = ?""";
            try(Connection conn = getConnection();
                PreparedStatement ps = conn.prepareStatement(query)){
                // Use PreparedStatement to avoid SQL injection attacks
                ps.setInt(1, film_id);
                ps.setInt(2, actor_id);
                ps.executeUpdate();

                // Refresh table and comboboxes
                refresh_controls(frm);

            } catch (SQLException ex){
                JOptionPane.showMessageDialog(frm, ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
            }
        }
    }

    private static void draw_pie_chart_film_by_actor_name(Charts_FilmActor fr
jp){
        jp.setPreferredSize(new Dimension(jp.getWidth(), jp.getHeight()));
        DefaultPieDataset dataset =
create_pie_dataset(Query_FilmActor.get_sql_film_actor_dist(), "Number", "acto

        //Draws piechart film distribution by actor name
```

```
        draw_piechart_with_dataset(frm, jp, dataset, "TOP 10 FILM DISTRIBUTI(
ACTOR NAME");
    }

    private static void draw_bar_chart_actor_by_avg_rental_rate(Charts_FilmA(
JPanel jp){
        jp.setPreferredSize(new Dimension(jp.getWidth(), jp.getHeight()));

        DefaultCategoryDataset dataset =
create_bar_dataset(Query_FilmActor.get_sql_film_actor_dist(), "avg_rental_rat
"actor_name");

        //Draws barchart actor name distribution by average rental rate
        draw_barchart_with_dataset(frm, jp, dataset, "TOP 10 ACTOR NAME DISTR
BY AVERAGE RENTAL RATE", "ACTOR NAME", "AVG RENTAL RATE");
    }

    private static void draw_bar_chart_actor_by_avg_replacement_cost(Charts_F
frm, JPanel jp){
        jp.setPreferredSize(new Dimension(jp.getWidth(), jp.getHeight()));

        DefaultCategoryDataset dataset =
create_bar_dataset(Query_FilmActor.get_sql_film_actor_dist(), "avg_replacemer
"actor_name");

        //Draws barchart actor name distribution by average replacement cost
        draw_barchart_with_dataset(frm, jp, dataset, "TOP 10 ACTOR NAME DISTR
BY AVERAGE REPLACEMENT COST", "ACTOR NAME", "AVG REPLACEMENT COST");
    }

    private static void draw_pie_chart_film_by_rating(Charts_FilmActor frm, J
{
        jp.setPreferredSize(new Dimension(jp.getWidth(), jp.getHeight()));
        DefaultPieDataset dataset =
create_pie_dataset(Query_FilmActor.get_sql_film_rating_dist(), "Number", "rat

        //Draws piechart film distribution by rating
        draw_piechart_with_dataset(frm, jp, dataset, "THE FILM DISTRIBUTION B
RATING");
    }

    private static void draw_bar_chart_rating_by_avg_rental_rate(Charts_FilmA
JPanel jp){
        jp.setPreferredSize(new Dimension(jp.getWidth(), jp.getHeight()));

        DefaultCategoryDataset dataset =
create_bar_dataset(Query_FilmActor.get_sql_film_rating_dist(), "avg_rental_ra
"rating");

        //Draws barchart rating distribution by average rental rate
        draw_barchart_with_dataset(frm, jp, dataset, "THE RATING DISTRIBUTION
AVERAGE RENTAL RATE", "RATING", "AVG RENTAL RATE");
    }
```

```
    private static void draw_bar_chart_rating_by_avg_replacement_cost(Charts_
frm, JPanel jp){
        jp.setPreferredSize(new Dimension(jp.getWidth(), jp.getHeight()));

        DefaultCategoryDataset dataset =
create_bar_dataset(Query_FilmActor.get_sql_film_rating_dist(), "avg_replaceme
"rating");

        //Draws barchart rating distribution by average replacement cost
        draw_barchart_with_dataset(frm, jp, dataset, "THE RATING DISTRIBUTION
AVERAGE REPLACEMENT COST", "RATING", "AVG REPLACEMENT COST");
    }

    public static void jbchart_handler(Charts_FilmActor frm){
        //Draws piechart film distribution by actor name
        draw_pie_chart_film_by_actor_name(frm, frm.getJPanel1());

        //Draws barchart actor name distribution by average rental rate
        draw_bar_chart_actor_by_avg_rental_rate(frm, frm.getJPanel2());

        //Draws barchart actor name distribution by replacement cost
        draw_bar_chart_actor_by_avg_replacement_cost(frm, frm.getJPanel3());

        //Draws piechart film distribution by rating
        draw_pie_chart_film_by_rating(frm, frm.getJPanel4());

        //Draws barchart rating distribution by average rental rate
        draw_bar_chart_rating_by_avg_rental_rate(frm, frm.getJPanel5());

        //Draws barchart rating distribution by average replacement cost
        draw_bar_chart_rating_by_avg_replacement_cost(frm, frm.getJPanel6());
    }
}
```

This is the full version of **FilmActorForm.java**:

```
package sakila;
import java.awt.Toolkit;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.*;
import javax.swing.event.TableModelEvent;
import javax.swing.event.TableModelListener;

public class FilmActorForm extends javax.swing.JFrame {

    public FilmActorForm() {
        initComponents();
        Utility.setLookAndFeel(this);
        FilmActor_Utils.refresh_controls(this);
```

```
 this.setIconImage(Toolkit.getDefaultToolkit().getImage(getClass().getResourc
;
        this.setDefaultCloseOperation(this.HIDE_ON_CLOSE);
    }

    //Getter methods for JTextField variable instances
    public JTextField getJTFFilmID(){return this.jtfFilmID;}
    public JTextField getJTFActorID(){return this.jtfActorID;}
    public JTextField getJTFDescription(){return this.jtfDescription;}
    public JTextField getJTFLastUpdate(){return this.jtfLastUpdate;}
    public JTextField getJTFLanguageName(){return this.jtfLanguageName;}
```

```
    public JTextField getJTFReleaseYear(){return this.jtfReleaseYear;}
    public JTextField getJTFRentalDuration(){return this.jtfRentalDuration;}
    public JTextField getJTFOriginalLanguageName(){return this.jtfOriginalLar
    public JTextField getJTFRentalRate(){return this.jtfRentalRate;}
    public JTextField getJTFLength(){return this.jtfLength;}
    public JTextField getJTFReplacementCost(){return this.jtfReplacementCost;
    public JTextField getJTFSpecialFeatures(){return this.jtfSpecialFeatures;
    public JTextField getJTFRating(){return this.jtfRating;}

    //Getter methods for JComboBox variable instances
    public JComboBox getJCBFilmID(){return this.jcbFilmID;}
    public JComboBox getJCBActorID(){return this.jcbActorID;}
    public JComboBox getJCBTitle(){return this.jcbTitle;}
    public JComboBox getJCBActorName(){return this.jcbActorName;}

    //Getter methods for JTable variable instance
    public JTable getJTFilmActor(){return this.jtFilmActor;}

    //Getter methods for JButton variable instances
    public JButton getJBEdit(){return this.jbEdit;}
    public JButton getJBInsert(){return this.jbInsert;}
    public JButton getJBDelete(){return this.jbDelete;}
    public JButton getJBChart(){return this.jbChart;}
    public JButton getJBFirst(){return this.jbFirst;}
    public JButton getJBPrev(){return this.jbPrev;}
    public JButton getJBNext(){return this.jbNext;}
    public JButton getJBLast(){return this.jbLast;}

    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">
    private void initComponents() {
        //...
        pack();
    }// </editor-fold>

    private void jbFirstActionPerformed(java.awt.event.ActionEvent evt) {
       FilmActor_Utils.show_first_row(this);
    }

    private void jbLastActionPerformed(java.awt.event.ActionEvent evt) {
        FilmActor_Utils.show_last_row(this);
    }

    private void jbPrevActionPerformed(java.awt.event.ActionEvent evt) {
        FilmActor_Utils.show_prev_row(this);
    }

    private void jbNextActionPerformed(java.awt.event.ActionEvent evt) {
        FilmActor_Utils.show_next_row(this);
    }

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

```
        FilmActor_Utils.jcbFilm_handler(this, this.jcbFilmID);
    }

    private void jcbTitleActionPerformed(java.awt.event.ActionEvent evt) {
        FilmActor_Utils.jcbFilm_handler(this, this.jcbTitle);
    }

    private void jtFilmActorMousePressed(java.awt.event.MouseEvent evt) {
        FilmActor_Utils.mouse_pressed_handler(this);
    }

    private void jbChartActionPerformed(java.awt.event.ActionEvent evt) {
        Charts_FilmActor frm1 = new Charts_FilmActor();
        frm1.setLocationRelativeTo(null);
        frm1.setTitle("SIX DISTRIBUTIONS IN FILM-ACTOR TABLE");
        frm1.setVisible(true);
        FilmActor_Utils.jbchart_handler(frm1);
    }

    private void jbDeleteActionPerformed(java.awt.event.ActionEvent evt) {
        FilmActor_Utils.delete_handler(this);
    }

    private void jbInsertActionPerformed(java.awt.event.ActionEvent evt) {
        FilmActor_Utils.insert_handler(this);
    }

    private void jbEditActionPerformed(java.awt.event.ActionEvent evt) {
        FilmActor_Utils.edit_handler(this);
    }

    private void jcbActorIDActionPerformed(java.awt.event.ActionEvent evt) {
        FilmActor_Utils.jcbActor_handler(this, this.jcbActorID, this.jcbFilm
    }

    private void jcbActorNameActionPerformed(java.awt.event.ActionEvent evt)
        FilmActor_Utils.jcbActor_handler(this, this.jcbActorName, this.jcbFil
    }

    private void jbFilmFormActionPerformed(java.awt.event.ActionEvent evt) {
        FilmForm frm = new FilmForm();
        frm.setVisible(true);
    }

    private void jbActorFormActionPerformed(java.awt.event.ActionEvent evt) {
        ActorForm frm = new ActorForm();
        frm.setVisible(true);
    }

    private void jbLanguageFormActionPerformed(java.awt.event.ActionEvent evt
        LanguageForm frm = new LanguageForm();
        frm.setVisible(true);
```

```
    }

    public static void main(String args[]) {
        try {
            for (javax.swing.UIManager.LookAndFeelInfo info :
javax.swing.UIManager.getInstalledLookAndFeels()) {
                if ("Nimbus".equals(info.getName())) {
                    javax.swing.UIManager.setLookAndFeel(info.getClassName())
                    break;
                }
            }
        } catch (ClassNotFoundException ex) {

java.util.logging.Logger.getLogger(FilmActorForm.class.getName()).log(java.ut
null, ex);
        } catch (InstantiationException ex) {

java.util.logging.Logger.getLogger(FilmActorForm.class.getName()).log(java.ut
null, ex);
        } catch (IllegalAccessException ex) {

java.util.logging.Logger.getLogger(FilmActorForm.class.getName()).log(java.ut
null, ex);
        } catch (javax.swing.UnsupportedLookAndFeelException ex) {

java.util.logging.Logger.getLogger(FilmActorForm.class.getName()).log(java.ut
null, ex);
        }
        //</editor-fold>
        //</editor-fold>
        //</editor-fold>
        //</editor-fold>

        /* Create and display the form */
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new FilmActorForm().setVisible(true);
            }
        });
    }

    // Variables declaration - do not modify
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel10;
    private javax.swing.JLabel jLabel11;
    private javax.swing.JLabel jLabel12;
    private javax.swing.JLabel jLabel13;
    private javax.swing.JLabel jLabel14;
    private javax.swing.JLabel jLabel15;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JLabel jLabel3;
    private javax.swing.JLabel jLabel4;
    private javax.swing.JLabel jLabel5;
```

```
    private javax.swing.JLabel jLabel6;
    private javax.swing.JLabel jLabel7;
    private javax.swing.JLabel jLabel8;
    private javax.swing.JLabel jLabel9;
    private javax.swing.JScrollPane jScrollPane1;
    private javax.swing.JButton jbActorForm;
    private javax.swing.JButton jbChart;
    private javax.swing.JButton jbDelete;
    private javax.swing.JButton jbEdit;
    private javax.swing.JButton jbFilmForm;
    private javax.swing.JButton jbFirst;
    private javax.swing.JButton jbInsert;
    private javax.swing.JButton jbLanguageForm;
    private javax.swing.JButton jbLast;
    private javax.swing.JButton jbNext;
    private javax.swing.JButton jbPrev;
    private javax.swing.JComboBox<String> jcbActorID;
    private javax.swing.JComboBox<String> jcbActorName;
    private javax.swing.JComboBox<String> jcbFilmID;
    private javax.swing.JComboBox<String> jcbTitle;
    private javax.swing.JTable jtFilmActor;
    private javax.swing.JTextField jtfActorID;
    private javax.swing.JTextField jtfDescription;
    private javax.swing.JTextField jtfFilmID;
    private javax.swing.JTextField jtfLanguageName;
    private javax.swing.JTextField jtfLastUpdate;
    private javax.swing.JTextField jtfLength;
    private javax.swing.JTextField jtfOriginalLanguageName;
    private javax.swing.JTextField jtfRating;
    private javax.swing.JTextField jtfReleaseYear;
    private javax.swing.JTextField jtfRentalDuration;
    private javax.swing.JTextField jtfRentalRate;
    private javax.swing.JTextField jtfReplacementCost;
    private javax.swing.JTextField jtfSpecialFeatures;
    // End of variables declaration
}
```

FILM CATEGORY FORM
FILM CATEGORY FORM

CREATING AND POPULATING FILM_CATEGORY TABLE

CREATING AND POPULATING FILM_CATEGORY TABLE

Step 1	Create a new class named **Query_FilmCategory**. It defines several S queries related to the film_category table in the sakila database. The first four queries are simple SELECT statements: 1. **sql_min**: Selects the minimum **film_id** from the **film** table. 2. **sql_max**: Selects the maximum **film_id** from the **film** table.

3. **sql_id**: Selects all columns from the **film** table where the **film**
matches a given parameter.
4. **sql_title**: Selects all columns from the **film** table where the t
matches a given parameter.

The next three queries are more complex and involve joining multiple table
1. **sql_film_category_dist**: Selects the category name, average ren
rate, average replacement cost, average length, and number of fil
in each category. The results are ordered by the number of films
each category (in descending order) and then by category name
ascending order), and limited to the top 10 categories.
2. **sql_film_category_joint**: Selects several columns from both
film_category and film tables, as well as the **language** a
category tables (by joining on their respective IDs). This qu
returns all rows from the **film_category** table and the correspond
data from the other tables.
3. **sql_film_category**: Defines a SQL statement to create
film_category table with several constraints and a default charac
set.

The class also provides getter methods for each of these queries, which c
be called by other parts of the program to access the queries as strings.

```
1  package sakila;
2
3  public class Query_FilmCategory {
4      private static final String
5  sql_min = "SELECT MIN(film_id) FROM
6  film";
7      private static final String
8  sql_max = "SELECT MAX(film_id) FROM
9  film";
10     private static final String
11 sql_id = "SELECT * FROM film WHERE
12 film_id  = ?";
13     private static final String
14 sql_title = "SELECT * FROM film
15 WHERE title = ?";
16
17     private static final String
18 sql_film_category_dist = """
19         SELECT c.name AS
20 category_name,
21         AVG(f.rental_rate) AS
22 avg_rental_rate,
23         AVG(f.replacement_cost) AS
24 avg_replacement_cost,
25         AVG(f.length) AS
26 avg_length,
27         COUNT(*) AS Number
28         FROM film_category fc
```

```
        JOIN film f ON f.film_id =
fc.film_id
        JOIN category c ON
c.category_id = fc.category_id
        JOIN language l ON
l.language_id = f.language_id
        GROUP BY category_name
        ORDER BY Count(*) DESC,
category_name ASC
        LIMIT 10""";

    private static final String
sql_film_category_joint = """
        SELECT fc.film_id,
fc.category_id, fc.last_update,
            f.title, f.description,
f.release_year,
            f.language_id,
f.original_language_id,
f.rental_duration,
            f.rental_rate,
f.length, f.replacement_cost,
f.rating,
            f.special_features,
l.name AS language_name,
            c.name AS category_name
        FROM film_category fc
        JOIN film f ON f.film_id =
fc.film_id
        JOIN category c ON
c.category_id = fc.category_id
        JOIN language l ON
l.language_id = f.language_id""";

    private static final String
sql_film_category = """
        CREATE TABLE film_category
(
          film_id SMALLINT UNSIGNED
NOT NULL,
          category_id TINYINT
UNSIGNED NOT NULL,
          last_update TIMESTAMP NOT
NULL DEFAULT CURRENT_TIMESTAMP ON
UPDATE CURRENT_TIMESTAMP,
          PRIMARY KEY (film_id,
category_id),
          CONSTRAINT
fk_film_category_film FOREIGN KEY
(film_id) REFERENCES film (film_id)
ON DELETE RESTRICT ON UPDATE
CASCADE,
          CONSTRAINT
fk_film_category_category FOREIGN
```

```
KEY (category_id) REFERENCES
category (category_id) ON DELETE
RESTRICT ON UPDATE CASCADE
        ) ENGINE=InnoDB DEFAULT
CHARSET=utf8mb4;""";

    //Getter methods
    public static String
get_sql_min() {
        return sql_min;
    }

    public static String
get_sql_max() {
        return sql_max;
    }

    public static String
get_sql_id() {
        return sql_id;
    }

    public static String
get_sql_title() {
        return sql_title;
    }

    public static String
get_sql_film_category() {
        return sql_film_category;
    }

    public static String
get_sql_film_category_joint() {
        return
sql_film_category_joint;
    }

    public static String
get_sql_film_category_dist() {
        return
sql_film_category_dist;
    }
}
```

Step 2	Then, create a public class named **FilmCategory** with instance variables a getter/setter methods. The class has 16 instance variables, including the I of the category and film, a **Timestamp** representing the last update tir and other information related to the film and category such as ti

description, release year, language, rental duration, rental rate, leng
replacement cost, rating, and special features.

The class has three constructors, one with no arguments, one with th
arguments for the ID of the category, ID of the film, and **Timesta**
representing the last update time, and one with all 16 arguments. The lat
constructor calls the three-argument constructor to set the category ID, f
ID, and last update time, and then sets the other instance variables.

The class also has getter and setter methods for each instance variable,
well as **hashCode()**, **equals()**, and **toString()** methods that override
corresponding methods inherited from the **Object** class.

The **hashCode()** method calculates a hash value based on the film
category ID, and last update time, using the **Objects.hash()** method.

The **equals()** method compares the film ID, category ID, and last upd
time of two **FilmCategory** objects using the == operator for the IDs and
Objects.equals() method for the Timestamp representing the last upd
time.

The **toString()** method returns a string representation of the **FilmCateg**
object, including all of its instance variables.

```
1   package sakila;
2   import java.util.Objects;
3   import java.sql.Timestamp;
4
5   public class FilmCategory {
6       //16 Instance Variables
7       private int cat_id;
8       private int film_id;
9       private Timestamp last_update;
10
11      private String title;
12      private String description;
13      private int release_year;
14      private int language_id;
15      private int original_language_id;
16      private int rental_duration;
17      private double rental_rate;
18      private int length;
19      private double replacement_cost;
20      private String rating;
21      private String special_features;
22
23      private String lang_name;
24      private String cat_name;
25
26      //Default constructor
```

```
27      FilmCategory(){
28          this(1, 1, new
29      Timestamp(System.currentTimeMillis()));
30      }
31
32      //Three-params constructor
33      FilmCategory(int cat_id, int
34      film_id, Timestamp lu){
35          setCategoryID(cat_id);
36          setFilmID(film_id);
37          setLastUpdate(lu);
38      }
39
40      //Sixteen-params constructor
41      FilmCategory(int cat_id, int
42      film_id, String title, String
43      description, int year, int lang_id, int
44      ori_lang_id,
45                  int duration, double
46      rental_rate, int length, double cost,
47      String rating,
48                  String features, String
49      lang_name, String cat_name, Timestamp
50      lu){
51          this(cat_id, film_id, lu);
52          this.title = title;
53          this.description = description;
54          this.release_year = year;
55          this.language_id = lang_id;
56          this.original_language_id =
57      ori_lang_id;
58          this.rental_duration =
59      duration;
60          this.rental_rate = rental_rate;
61          this.length = length;
62          this.replacement_cost = cost;
63          this.rating = rating;
64          this.special_features =
65      features;
66          this.lang_name = lang_name;
67          this.cat_name = cat_name;
68      }
69
70      // Getter methods
71      public int getCategoryID() {return
72      cat_id;}
73      public int getFilmID() {return
74      film_id;}
75      public Timestamp getLastUpdate()
76      {return last_update;}
77
78      public String getTitle() {return
79      title;}
80
```

```
81      public String getDescription()
82  {return description;}
83      public int getReleaseYear() {return
84  release_year;}
85      public int getLanguageID() {return
86  language_id;}
87      public int getOriginalLanguageID()
88  {return original_language_id;}
89      public int getRentalDuration()
90  {return rental_duration;}
91      public double getRentalRate()
92  {return rental_rate;}
93      public int getLength() {return
94  length;}
95      public double getReplacementCost()
96  {return replacement_cost;}
97      public String getRating() {return
98  rating;}
99      public String getSpecialFeatures()
100 {return special_features;}
101     public String getLanguageName()
102 {return lang_name;}
103     public String getCategoryName()
104 {return cat_name;}
105
106     //Setter methods
107     public void setCategoryID(int id)
108 {
109         if (id <= 0) {
110             throw new
111 IllegalArgumentException("Category ID must be
112 greater than zero.");
113         }
114         this.cat_id = id;
115     }
116
117     public void setFilmID(int id) {
118         if (id <= 0) {
119             throw new
120 IllegalArgumentException("Category ID must be
121 greater than zero.");
122         }
123         this.film_id = id;
124     }
125 126127
128     public void setLastUpdate(Timestamp
129 date){
130         if (date == null) {
131         throw new
132 IllegalArgumentException("Date cannot
133 be null");
134         }
135         this.last_update = date;
```

```
136     }
137
        // Override the hashCode() method
        @Override
        public int hashCode() {
            return Objects.hash(film_id,
cat_id, last_update);
        }

        // Override the equals() method
        @Override
        public boolean equals(Object o) {
            if (this == o) return true;
            if (o == null || getClass() !=
o.getClass()) return false;
            FilmCategory fc =
(FilmCategory) o;
            return film_id == fc.film_id &&
                   cat_id == fc.cat_id &&

Objects.equals(last_update,
fc.last_update);
        }

        @Override
        public String toString(){
            return  "\nFilm ID
: " + this.getFilmID() +
                    "\nCategory ID
: " + this.getCategoryID() +
                    "\nCategory Name
: " + this.getCategoryName() +
                    "\nTitle
: " + this.getTitle() +
                    "\nDescription
: " + this.getDescription() +
                    "\nRelease Year
: " + this.getReleaseYear() +
                    "\nLanguage ID
: " + this.getLanguageID() +
                    "\nLanguage Name
: " + this.getLanguageName() +
                    "\nOriginal Language ID
: " + this.getOriginalLanguageID() +
                    "\nRental Duration
: " + this.getRentalDuration()+
                    "\nRental Rate
: " + this.getRentalRate()+
                    "\nLength
: " + this.getLength()+
                    "\nReplacement Cose
: " + this.getReplacementCost()+
```

```
                    "\nRating
: " + this.getRating()+
                    "\nSpecial Features
: " + this.getSpecialFeatures()+
                    "\nLast Update
: " + this.getLastUpdate();
    }
}
```

Step 3

Create a new public class named **FilmCategory_Utils**. It extends the **Util** class. It contains several methods for working with the **film_category** tabl

1. **create_film_category_table()**: This method creates **film_category** table in the Sakila database by executing the S query obtained from **Query_FilmCategory.get_sql_film_categor** using a **Statement** object. If successful, it displays a message dia indicating that the table has been created. If an SQL except occurs, it displays an error message.
2. **populate_film_category_table()**: This method populates **film_category** table in the Sakila database with two rows of data executing an SQL INSERT statement using two **PreparedStateme** objects. The first **PreparedStatement** object creates a n **FilmCategory** object using its default constructor and sets values of its fields as the parameters for the INSERT statement. T second **PreparedStatement** object creates a new **FilmCateg** object using its three-params constructor and sets the values of fields as the parameters for the INSERT statement. If an S exception occurs, it displays an error message.
3. **read_film_category_table()**: This method reads the content of **film_category** table in the Sakila database and joins it with **category**, **film**, and **language** tables to retrieve additional data executes an SQL query obtained fr **Query_FilmCategory.get_sql_film_category_joint()** using **Statement** object and reads the result set using a while loop. each row in the result set, it retrieves the values of the columns us the **ResultSet** methods and creates a new **FilmCategory** obj using its sixteen-params constructor, passing the retrieved values arguments. It then prints the **FilmCategory** object to the console an SQL exception occurs, it displays an error message.

```
1  package sakila;
2  import java.awt.Dimension;
3  import java.util.logging.Level;
4  import java.util.logging.Logger;
5  import java.sql.*;
6  import java.util.logging.Level;
7  import java.util.logging.Logger;
```

```java
import java.sql.*;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashMap;
import java.util.LinkedHashMap;
import java.util.List;
import java.util.Map;
import java.util.Objects;
import javax.swing.JComboBox;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.event.TableModelEvent;
import javax.swing.table.DefaultTableModel;
import javax.swing.table.TableModel;
import org.jfree.data.category.DefaultCategoryDataset;
import org.jfree.data.general.DefaultPieDataset;

public class FilmCategory_Utils extends Utility{
    public static final int FIRST_INDEX = 0;
    public static final int INVALID_INDEX = -1;

    private static int currentIndex = FIRST_INDEX;
    private static final String SQL_ID =
Query_FilmCategory.get_sql_film_category_joint() + " WHERE
fc.film_id = ?";

    //Creates film_category table
    public static void create_film_category_table() {
        try (Connection conn = getConnection()) {
            Statement stmt = conn.createStatement();

stmt.addBatch(Query_FilmCategory.get_sql_film_category());
            stmt.executeBatch();

            String message = String.format("Successfully creates film_category
table");
            JOptionPane.showMessageDialog(null, message,
            "INFORMATION",JOptionPane.INFORMATION_MESSAGE);

        } catch (SQLException ex) {
            JOptionPane.showMessageDialog(null, ex.getMessage(),
                "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    //Populates film_category table with some rows of data
    public static void populate_film_category_table(){
        try(Connection conn = getConnection()){
            String sql = """
                    INSERT INTO film_category(category_id, film_id,
last_update)
                    VALUES(?, ?, ?)""";

```

```
            PreparedStatement ps1 = conn.prepareStatement(sql);
            //Creates a new FilmCategory class with default
constructor
            FilmCategory obj1 = new FilmCategory();
            ps1.setInt(1,obj1.getCategoryID());
            ps1.setInt(2, obj1.getFilmID());
            ps1.setTimestamp(3, obj1.getLastUpdate());

            PreparedStatement ps2 = conn.prepareStatement(sql);
            //Creates a new FilmCategory class with three-params
constructor
            FilmCategory obj2 = new FilmCategory(2, 2, new
Timestamp(System.currentTimeMillis()));
            ps2.setInt(1,obj2.getCategoryID());
            ps2.setInt(2, obj2.getFilmID());
            ps2.setTimestamp(3, obj2.getLastUpdate());

            ps1.executeUpdate();
            ps2.executeUpdate();

        }catch(SQLException ex){
            JOptionPane.showMessageDialog(null, ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    //Reads the content of joined film_category, category, film, a
language tables
    public static void read_film_category_table(){
        try(Connection conn = getConnection()){
            Statement stmt = conn.createStatement();
            ResultSet rs =
stmt.executeQuery(Query_FilmCategory.get_sql_film_category_joint()
            while(rs.next()){
                int cat_id = rs.getInt("category_id");
                int film_id = rs.getInt("film_id");
                String title = rs.getString("title");
                String description = rs.getString("description");
                int year = rs.getInt("release_year");
                int lang_id = rs.getInt("language_id");
                int ori_lang_id = rs.getInt("original_language_id"
                int duration = rs.getInt("rental_duration");
                double rate = rs.getDouble("rental_rate");
                int length = rs.getInt("length");
                double cost = rs.getDouble("replacement_cost");
                String rating = rs.getString("rating");
                String features = rs.getString("special_features")
                String lang_name = rs.getString("language_name");
                String cat_name = rs.getString("category_name");
                Timestamp lu = rs.getTimestamp("last_update");

                //Creates an FilmCategory object using sixteen-
params constructor

```

```
116                 FilmCategory obj = new FilmCategory(cat_id, film_i
117 title, description, year, lang_id, ori_lang_id, duration, rate,
118 length, cost, rating, features, lang_name, cat_name, lu);
119                 System.out.println(obj);
120             }
                rs.close();
                stmt.close();

            }catch(SQLException ex){
                JOptionPane.showMessageDialog(null, ex.getMessage(),
                        "ERROR",JOptionPane.ERROR_MESSAGE);
            }
        }
}
```

Step 4

In the driver class, **Sakila.java**, invoke **create_film_category_table()**, **populate_film_category_table()**, and **read_film_category_table()** shown in line 36 – 38:

```
1  package sakila;
2
3  public class Sakila {
4      public static void main(String[] args) {
5  //         Utility.testConnection();
6  //         Actor_Utils.create_actor_table();
7  //         Actor_Utils.populate_actor_table();
8  //         Actor_Utils.read_actor_table();
9  //         ActorForm frm = new ActorForm();
10 //         frm.setVisible(true);
11
12 //         Language_Utils.create_language_table();
13 //
14 Language_Utils.populate_language_table();
15 //         Language_Utils.read_language_table();
16 //         LanguageForm frm = new LanguageForm();
17 //         frm.setVisible(true);
18
19 //         Category_Utils.create_category_table();
20 //
21 Category_Utils.populate_category_table();
22 //         Category_Utils.read_category_table();
23 //         CategoryForm frm = new CategoryForm();
24 //         frm.setVisible(true);
25
26 //         Film_Utils.create_film_table();
27 //         Film_Utils.populate_film_table();
28 //         Film_Utils.read_film_table();
29 //         FilmForm frm = new FilmForm();
30 //         frm.setVisible(true);
31
32
```

```
//
FilmActor_Utils.create_film_actor_table();
//
FilmActor_Utils.populate_film_actor_table();
//        FilmActor_Utils.read_film_actor_table();
//        FilmActorForm frm = new FilmActorForm();
//        frm.setVisible(true);


FilmCategory_Utils.create_film_category_table();

FilmCategory_Utils.populate_film_category_table();

FilmCategory_Utils.read_film_category_table();
    }
}
```

Run project to see the result in console:

```
Film ID              : 1
Category ID          : 1
Category Name        : Category xxx
Title                : NO TITLE
Description          : null
Release Year         : 2023
Language ID          : 1
Language Name        : Language xxxe
Original Language ID : 1
Rental Duration      : 3
Rental Rate          : 4.99
Length               : 100
Replacement Cose     : 19.99
Rating               : G
Special Features     : null
Last Update          : 2023-04-25 14:17:19.0

Film ID              : 2
Category ID          : 2
Category Name        : Ethnics
Title                : Batak Ethnic Movie
Description          : null
Release Year         : 2023
Language ID          : 1
Language Name        : Language xxxe
Original Language ID : 1
Rental Duration      : 3
```

```
Rental Rate          : 2.99
Length               : 150
Replacement Cose     : 13.99
Rating               : G
Special Features     : Trailers,Commentaries
Last Update          : 2023-04-25 14:17:19.0
```

DESIGNING GUI
DESIGNING GUI

Step 1	In the project, create a new **JFrame Form** and name it as **FilmCategoryForm.java**. In the Design tab, add fifteen **JLabel** to the form and set their corresponding **text** properties as FILM ID, TITLE, DESCRIPTION, RELEASE YEAR, RENTAL RATE, RENTAL DURATION, LENGTH, REPLACEMENT COST, RATING, LANGUAGE, ORIGINAL LANGUAGE, SPECIAL FEATURES, CAT. ID, CAT. NAME, and LAST UPDATE.
Step 2	Then, add thirteen **JTextField** to the form and set set their corresponding **Variable Name** as **jtfFilmID**, **jtfCategoryID**, **jtfDescription**, **jtfReleaseYear**, **jtfRentalRate**, **jtfLength**, **jtfRentalDuration**, **jtfRating**, **jtfReplacementCost**, **jtfLanguageName**, **jtfOriginalLanguageName**, **jtfSpecialFeatures**, and **jtfLastUpdate**.
Step 3	Then, add eleven **JButton** to the form and set their corresponding **Variable Name** as **jbFirst**, **jbPrev**, **jbNext**, **jbLast**, **jbEdit**, **jbInsert**, **jbDelete**, **jbChart**, **jbLanguageForm**, **jbCategoryForm**, and **jbFilmForm**. Set their corresponding **text** properties as FIRST, PREV, NEXT, LAST, EDIT, INSERT, DELETE, CHART, LANGUAGE FORM, CATEGORY FORM, and FILM FORM.
Step 4	Then, add four **JComboBox** to the form and set set their corresponding **Variable Name** as **jcbFilmID**, **jcbTitle** **jcbCategoryID**, and **jcbCategoryName**.
Step 5	Lastly, add a new **JTable** to the form set set its **Variable Name** as **jtFilmCategory**. Then, right-click on it, then

	choose **Table Contents...** and set the number of columns to 16 and the number of rows to 50.
Step 6	In the driver class, **Sakila.java**, create a new object of **FilmCategoryForm** class using its default constructor as shown in 39 – 40

```
1  package sakila;
2
3  public class Sakila {
4      public static void main(String[] args) {
5  //         Utility.testConnection();
6  //         Actor_Utils.create_actor_table();
7  //         Actor_Utils.populate_actor_table();
8  //         Actor_Utils.read_actor_table();
9  //         ActorForm frm = new ActorForm();
10 //         frm.setVisible(true);
11
12 //         Language_Utils.create_language_table();
13 //
14 Language_Utils.populate_language_table();
15 //         Language_Utils.read_language_table();
16 //         LanguageForm frm = new LanguageForm();
17 //         frm.setVisible(true);
18
19 //         Category_Utils.create_category_table();
20 //
21 Category_Utils.populate_category_table();
22 //         Category_Utils.read_category_table();
23 //         CategoryForm frm = new CategoryForm();
24 //         frm.setVisible(true);
25
26 //         Film_Utils.create_film_table();
27 //         Film_Utils.populate_film_table();
28 //         Film_Utils.read_film_table();
29 //         FilmForm frm = new FilmForm();
30 //         frm.setVisible(true);
31
32 //
33 FilmActor_Utils.create_film_actor_table();
34 //
35 FilmActor_Utils.populate_film_actor_table();
36 //         FilmActor_Utils.read_film_actor_table();
37 //         FilmActorForm frm = new FilmActorForm();
38 //         frm.setVisible(true);
39
40 //
41 FilmCategory_Utils.create_film_category_table();
42 //
   FilmCategory_Utils.populate_film_category_table();
```

```
//
FilmCategory_Utils.read_film_category_table();
        FilmCategoryForm frm = new
FilmCategoryForm();
        frm.setVisible(true);
    }
}
```

Step 8

In **FilmCategoryForm**'s constructor, invoke **setLookAndFeel()** to set the look and feel of the form as shown in line 17.

```
1  package sakila;
2
3  import java.awt.Toolkit;
4  import java.awt.event.ActionEvent;
5  import
6  java.awt.event.ActionListener;
7  import javax.swing.JButton;
8  import javax.swing.JComboBox;
9  import javax.swing.JMenuItem;
10 import javax.swing.JOptionPane;
11 import javax.swing.JPopupMenu;
12 import javax.swing.JTable;
13 import javax.swing.JTextField;
14
15 public class FilmCategoryForm
16 extends javax.swing.JFrame {
17     public FilmCategoryForm() {
18         initComponents();
19
20 Utility.setLookAndFeel(this);
    }
    //...
}
```

Run the project to see the film category form as shown in Figure 7.1.

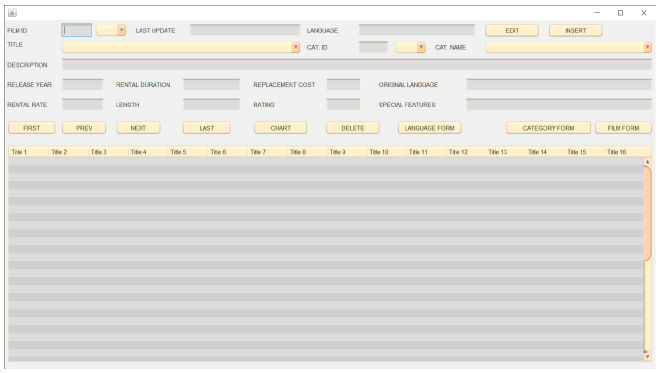


Figure 7.1 The layout of film category form

Step 9

In **FilmCategoryForm.java**, define **getter()** method for every object in the form and place them outside and beneath its default constructor:

```
1     //Getter methods for JTextField
2  variable instances
3     public JTextField
4  getJTFFilmID(){return
5  this.jtfFilmID;}
6     public JTextField
7  getJTFCategoryID(){return
8  this.jtfCategoryID;}
9     public JTextField
10 getJTFDescription(){return
11 this.jtfDescription;}
12    public JTextField
13 getJTFLastUpdate(){return
14 this.jtfLastUpdate;}
15    public JTextField
16 getJTFLanguageName(){return
17 this.jtfLanguageName;}
18    public JTextField
19 getJTFReleaseYear(){return
20 this.jtfReleaseYear;}
21    public JTextField
22 getJTFRentalDuration(){return
23 this.jtfRentalDuration;}
24    public JTextField
25 getJTFOriginalLanguageName(){return
26 this.jtfOriginalLanguageName;}
27    public JTextField
28 getJTFRentalRate(){return
29 this.jtfRentalRate;}
30    public JTextField
31 getJTFLength(){return
32 this.jtfLength;}
33    public JTextField
34 getJTFReplacementCost(){return
```

```
this.jtfReplacementCost;}
    public JTextField
getJTFSpecialFeatures(){return
this.jtfSpecialFeatures;}
    public JTextField
getJTFRating(){return
this.jtfRating;}

    //Getter methods for JComboBox
variable instances
    public JComboBox getJCBFilmID()
{return this.jcbFilmID;}
    public JComboBox
getJCBCategoryID(){return
this.jcbCategoryID;}
    public JComboBox getJCBTitle()
{return this.jcbTitle;}
    public JComboBox
getJCBCategoryName(){return
this.jcbCategoryName;}

    //Getter methods for JTable
variable instance
    public JTable
getJTFilmCategory(){return
this.jtFilmCategory;}

    //Getter methods for JButton
variable instances
    public JButton getJBEdit()
{return this.jbEdit;}
    public JButton getJBInsert()
{return this.jbInsert;}
    public JButton getJBDelete()
{return this.jbDelete;}
    public JButton getJBChart()
{return this.jbChart;}
    public JButton getJBFirst()
{return this.jbFirst;}
    public JButton getJBPrev()
{return this.jbPrev;}
    public JButton getJBNext()
{return this.jbNext;}
    public JButton getJBLast()
{return this.jbLast;}
```

POPULATING TABLE AND COMBOBOXES
POPULATING TABLE AND COMBOBOXES

Step 1

In **FilmCategory_Utils.java**, add two new methods: **get_film_cat_list()** a

The first method, **get_film_cat_list()**, takes three parameters: a **Fi** statement, and an item string. It returns an **ArrayList** of **FilmCategory** c empty **ArrayList** and initializes a **Connection** object to null. It then attem database by calling the **getConnection()** method and creates a **Pre** provided SQL statement. If the item parameter is not equal to "none", prepared statement to item.

The method then executes the query by calling **executeQuery()** on the p **ResultSet**. It then iterates over each row in the result set and creates a n sixteen-params constructor. The **FilmCategory** object is then added to the

If there is a **SQLException**, the method displays an error message dialog

The second method, **show_table_film_cat()**, takes two parameters: a **I ArrayList** of **FilmCategory** objects. The method creates a new **Default** and columns. It then creates a header string array containing **set_column_header()** to set the column headers on the **JTable** in th method then sets the model on the **JTable**.

The method then iterates over each **FilmCategory** object in the **Arra** column in the row object. The row object is then added to the table model.

```
1     private static ArrayList<FilmCategory>
2  get_film_cat_list(FilmCategoryForm frm, String sql, String item){
3          ArrayList<FilmCategory> list = new ArrayList<>();
4          Connection conv = null;
5
6          try(Connection conn = getConnection();
7              PreparedStatement ps = conn.prepareStatement(sql)){
8              if (item.equalsIgnoreCase("none")==false) {
9                   ps.setString(1,item);
10             }
11             ResultSet rs = ps.executeQuery();
12
13             FilmCategory obj;
14             while(rs.next()){
15                 //Using sixteen-params constructor
16                 obj = new FilmCategory(rs.getInt("category_id"),
17 rs.getInt("film_id"),
18                     rs.getString("title"),
19                     rs.getString("description"),
20 rs.getInt("release_year"),
21                     rs.getInt("language_id"),
22 rs.getInt("original_language_id"),
23                     rs.getInt("rental_duration"),
24 rs.getDouble("rental_rate"),
```

```
                    rs.getInt("length"),
rs.getDouble("replacement_cost"),
                    rs.getString("rating"),
rs.getString("special_features"),
                    rs.getString("language_name"),
rs.getString("category_name"),
                        rs.getTimestamp("last_update"));

                list.add(obj);
            }
        }catch (SQLException ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
                "ERROR",JOptionPane.ERROR_MESSAGE);
        }
        return list;
    }

    private static void show_table_film_cat(FilmCategoryForm frm,
ArrayList<FilmCategory> list) throws SQLException{
        DefaultTableModel model = new DefaultTableModel(0,0);

        String header[] = {"Category ID", "Category Name", "Film ID
"Title", "Description", "Release Year", "Language ID", "Original
Language", "Rental Duration", "Rental Rate", "Length", "Replacement
Cost", "Rating", "Special Features", "Language Name", "Last Update"


model.setColumnIdentifiers(set_column_header(frm.getJTFilmCategory(
header));
        frm.getJTFilmCategory().setModel(model);

        Object[] row = new Object[16];

        for(int i=0; i<list.size(); i++){
            row[0] = list.get(i).getCategoryID();
            row[1] = list.get(i).getCategoryName();
            row[2] = list.get(i).getFilmID();
            row[3] = list.get(i).getTitle();
            row[4] = list.get(i).getDescription();
            row[5] = list.get(i).getReleaseYear();
            row[6] = list.get(i).getLanguageID();
            row[7] = list.get(i).getOriginalLanguageID();
            row[8] = list.get(i).getRentalDuration();
            row[9] = list.get(i).getRentalRate();
            row[10] = list.get(i).getLength();
            row[11] = list.get(i).getReplacementCost();
            row[12] = list.get(i).getRating();
            row[13] = list.get(i).getSpecialFeatures();
            row[14] = list.get(i).getLanguageName();
            row[15] = list.get(i).getLastUpdate();

            model.addRow(row);
        }
    }
```

Step 2

In **FilmCategory_Utils.java**, define **refresh_controls()** method. This me on the **FilmCategoryForm**. It sets the location and title of the form, pop film and category tables joined together with a specific SQL query, and data from the **film** and **category** tables.

The method calls the following methods:

- **table_renderer(frm.getJTFilmCategory())**: This method sets tl alternating colors for the rows.
- **get_film_cat_list(frm, Query_FilmCategory.get_sql_film_cate f.film_id", "none")**: This method returns an **ArrayList** of **FilmC** from the film and category tables joined together with a specific S based on the arguments passed to the method. The **ArrayList** is tl the **FilmCategoryForm** using the **show_table_film_cat(frm, list)**
- **populate_combobox(sql_id, frm.getJCBFilmID(), frm)**: This r for the film ID with data from the film table. The method takes a S it to retrieve the data from the database. The retrieved data is ther
- **populate_combobox(sql_title, frm.getJCBTitle(), frm)**: This r for the film title with data from the **film** table. The method takes uses it to retrieve the data from the database. The retrieved **JComboBox**.
- **populate_combobox(sql_cat_id, frm.getJCBCategoryID(), fr JComboBox** for the category ID with data from the **category** table an argument and uses it to retrieve the data from the database. populate the **JComboBox**.
- **populate_combobox(sql_cat_name, frm.getJCBCategoryNam** the **JComboBox** for the category name with data from the **categ** query as an argument and uses it to retrieve the data from the d used to populate the **JComboBox**.

If there is an **SQLException**, the method displays an error message dialog

```
1     public static void
2  refresh_controls(FilmCategoryForm frm){
3          frm.setLocationRelativeTo(null);
4          frm.setTitle("FILM ACTOR FORM");
5
6          //Shows the content of film table and
7  populates combobox
8          try{
9              //Makes alternating color for table
10 rows
11
12 table_renderer(frm.getJTFilmCategory());
13
14             //Populates table
```

```
            ArrayList<FilmCategory> list =
get_film_cat_list(frm,
Query_FilmCategory.get_sql_film_category_joint()
+ " ORDER BY f.film_id", "none");
            show_table_film_cat(frm, list);

            //Populates jcbFilmID
            String sql_id = "SELECT film_id FROM
film ORDER BY film_id";
            populate_combobox(sql_id,
frm.getJCBFilmID(), frm);

            //Populates getJCBTitle()
            String sql_title = "SELECT DISTINCT
title FROM film ORDER BY title";
            populate_combobox(sql_title,
frm.getJCBTitle(), frm);

            //Populates jcbCategoryID
            String sql_cat_id = "SELECT
category_id FROM category ORDER BY category_id";
            populate_combobox(sql_cat_id,
frm.getJCBCategoryID(), frm);

            //Populates jcbCategoryName
            String sql_cat_name = "SELECT DISTINCT name
FROM category ORDER BY name";
            populate_combobox(sql_cat_name,
frm.getJCBCategoryName(), frm);

        }catch (SQLException ex){
            JOptionPane.showMessageDialog(frm,
ex.getMessage(),
            "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }
```

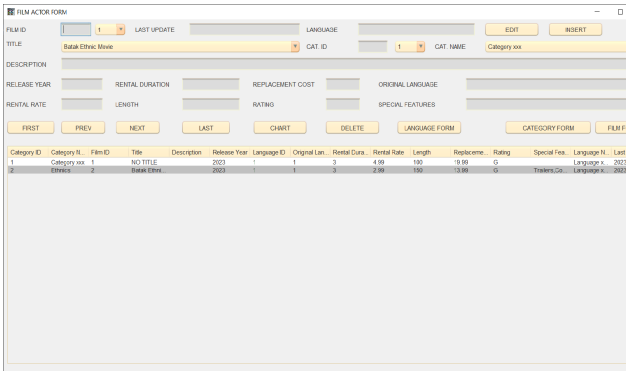


Figure 7.2 The content of joined **film_category**, **film**, **category**, and **jtFilmCategory**

Step 3

In **FilmCategoryForm**'s default constructor, the **FilmCategor_Util** initializes and populates the controls in the **FilmCategoryForm** with **refresh_controls()** method from the **FilmCategory_Utils** class.

The **this.setIconImage()** method sets the icon of the **FilmCat** **this.setDefaultCloseOperation(this.HIDE_ON_CLOSE)** method sets **FilmCategoryForm** to hide the form instead of exiting the application wh

```
1  package sakila;
2  import java.awt.Toolkit;
3  import java.awt.event.ActionEvent;
4  import java.awt.event.ActionListener;
5  import java.io.IOException;
6  import java.text.ParseException;
7  import java.util.logging.Level;
8  import java.util.logging.Logger;
9  import javax.swing.JButton;
10 import javax.swing.JComboBox;
11 import javax.swing.JLabel;
12 import javax.swing.JTable;
13 import javax.swing.JTextField;
14
15 public class FilmCategoryForm extends javax.swing.JFrame {
16     public FilmCategoryForm() {
17         initComponents();
18         Utility.setLookAndFeel(this);
19         FilmCategory_Utils.refresh_controls(this);
20
21  this.setIconImage(Toolkit.getDefaultToolkit().getImage(getClass().
22 ;
23         this.setDefaultCloseOperation(this.HIDE_ON_CLOSE);
24     }
25     //...
26 }
```

Step 4

Run the project to see the content of joined **film_category**, **film**, **category** **jtFilmCategory** as shown in Figure 7.2.

If you use the data from **Sakila** MySQL database available in the intern joined **film_category**, **film**, **category**, and **language** tables displayed in **j** 7.3.

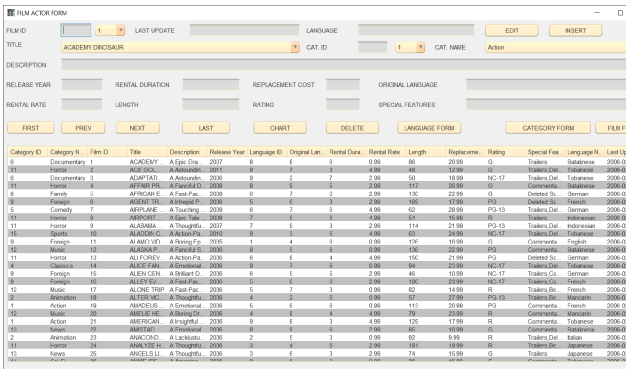


Figure 7.3 The the content of joined **film_actor**, **film**, **actor**, and **lan**
database available in the internet displayed in

DISPLAYING AND NAVIGATING DATA ROW BY ROW
DISPLAYING AND NAVIGATING DATA ROW BY ROW

Step 1

In **FilmCategory_Utils** class, define two new methods named **cl**
display_film_data().

The **clear_controls()** method is used to reset the values of various text fi
in the **FilmCategoryForm** object passed as a parameter. This method is c
found in the result set of a query in the **display_film_data()** method. Th
that the form fields are empty when there is no data to display.

The **display_film_data()** method displays the film data obtained from the
fields of the **FilmCategoryForm** object passed as a parameter. The SQL
data is executed using the **Connection** object obtained from the **getCo**
The **PreparedStatement** object is used to set the parameter value, and t
using the **executeQuery()** method of the **PreparedStatement** object.

The **ResultSet** object obtained from the query is processed using a do-
the data row by row. The values of the different fields in the **FilmCategor**
using the **setText()** method of the relevant text field. The **find_coml**
method is used to determine the item selected in the combo boxes.

If no row is found in the result set, the **clear_controls()** method is call
fields to empty strings. If an SQL exception occurs, an error message is
JOptionPane.showMessageDialog() method.

```
1      private static void clear_controls(FilmCategoryForm frm){
2          frm.getJTFFilmID().setText("");
3          frm.getJTFDescription().setText("");
4          frm.getJTFReleaseYear().setText("");
```

```
        frm.getJTFLanguageName().setText("");
        frm.getJTFOriginalLanguageName().setText("");
        frm.getJTFRentalDuration().setText("");
        frm.getJTFRentalRate().setText("");
        frm.getJTFLength().setText("");
        frm.getJTFReplacementCost().setText("");
        frm.getJTFSpecialFeatures().setText("");
        frm.getJTFLastUpdate().setText("");
    }

   //Displays film data result row by row
    private static <T> void display_film_data(FilmCategoryForm frm,
item){
        try(Connection conn = getConnection()){
            PreparedStatement ps = conn.prepareStatement(sql);
            ps.setObject(1,item);
            ResultSet rs = ps.executeQuery();

            if (!rs.next()) {
                // no row found, clear the form fields
                clear_controls(frm);
                return;
            }

            do{
                frm.getJTFFilmID().setText(String.valueOf(rs.getInt
                frm.getJTFDescription().setText(rs.getString("descr

frm.getJTFReleaseYear().setText(String.valueOf(rs.getInt("release_y
                frm.getJTFLanguageName().setText(rs.getString("lang

frm.getJTFRentalDuration().setText(String.valueOf(rs.getInt("rental

frm.getJTFRentalRate().setText(String.valueOf(rs.getDouble("rental_
                frm.getJTFLength().setText(String.valueOf(rs.getInt

frm.getJTFReplacementCost().setText(String.valueOf(rs.getDouble("re
                frm.getJTFSpecialFeatures().setText(rs.getString("s
                frm.getJTFRating().setText(rs.getString("rating"));

frm.getJTFLastUpdate().setText(String.valueOf(rs.getDate("last_upda

                // Determines item selected from jcbFilmID
                find_combo_value_selected(frm.getJCBFilmID(), rs.ge

                // Determines item selected from jcbTitle
                find_combo_value_selected(frm.getJCBTitle(), rs.get

               // Finds original language name
                int ori_lang_id = rs.getInt("original_language_id")
                if(ori_lang_id != 0) {
                    Object lang_name = get_val_from_database("langu
"language_id", ori_lang_id);
```

```
59
60  frm.getJTFOriginalLanguageName().setText(String.valueOf(lang_name))
61
62                  }
63
64              }while(rs.next());
65
66              rs.close();
67              ps.close();
68          }catch(SQLException ex){
69              JOptionPane.showMessageDialog(frm, ex.getMessage(),
70                      "ERROR",JOptionPane.ERROR_MESSAGE);
71          }
72      }
73
74
75
76
```

Step 2

In the same class, define another method named **jcbFilm_handler()**. This selection of items in the film ID and title combo boxes in the **FilmCategor**

First, it gets the selected item from the combo box and determines if it i combo box. Based on this, it constructs the SQL query to retrieve the database using the appropriate identifier.

Next, it calls the **get_film_cat_list()** method to retrieve the film categorie selected film, and then displays this information in the **jtFilmCateg** **show_table_film_cat()** method.

Finally, it calls the **display_film_data()** method to retrieve the film data u SQL query and displays it in the appropriate text fields in the GUI. If an this process, a message dialog box is displayed to inform the user.

```
1   public static void
2   jcbFilm_handler(FilmCategoryForm frm,
3   JComboBox<String> jcb) {
4           Object item = jcb.getSelectedItem();
5           String sql = "";
6           if (jcb.equals(frm.getJCBFilmID())) {
7               sql =
8   Query_Film.get_sql_film_joint() + " WHERE
9   f.film_id = ?";
10
11              try{
12                  //Filters jtFilmCategory
13                  ArrayList<FilmCategory> list =
14  get_film_cat_list(frm,
15  Query_FilmCategory.get_sql_film_category_joint()
```

```
+ " WHERE fc.film_id = ?",
String.valueOf(item));
                show_table_film_cat(frm, list);
            }catch (SQLException ex){

JOptionPane.showMessageDialog(frm,
ex.getMessage(),

 "ERROR",JOptionPane.ERROR_MESSAGE);
            }
        } else if
(jcb.equals(frm.getJCBTitle())) {
            sql =
Query_Film.get_sql_film_joint() + " WHERE
f.title = ?";

            try{
                //Filters jtFilmCategory
                ArrayList<FilmCategory> list =
get_film_cat_list(frm,
Query_FilmCategory.get_sql_film_category_joint()
+ " WHERE f.title = ?", String.valueOf(item));
                show_table_film_cat(frm, list);
            }catch (SQLException ex){

JOptionPane.showMessageDialog(frm,
ex.getMessage(),

 "ERROR",JOptionPane.ERROR_MESSAGE);
            }
        }

        display_film_data(frm, sql, item);
    }
```

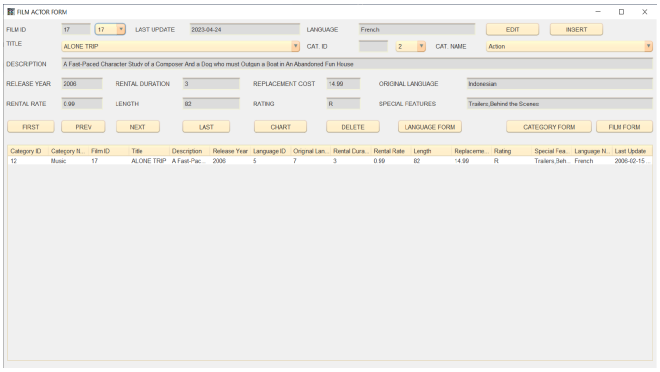


Figure 7.4 Displaying row by row the content of **film** ta

Step 3	In **FilmCategoryForm**, double click on **jcbFilmID** and **jcbTitle** combobc corresponding event handler as follows:

```
    private void
jcbFilmIDActionPerformed(java.awt.event.ActionEvent
evt) {
        FilmCategory_Utils.jcbFilm_handler(this,
this.jcbFilmID);
    }

    private void
jcbTitleActionPerformed(java.awt.event.ActionEvent
evt) {
        FilmCategory_Utils.jcbFilm_handler(this,
this.jcbTitle);
    }
```

These two methods are event handlers for the two combo boxes in the f
jcbTitle. They call the **jcbFilm_handler()** method in the **FilmCategory**
the form and the respective combo box as parameters. This method is res
and displaying the data based on the selected item from the combo box.

Step 4

Run the project. Choose one of items in **jcbFilmID** and/or **jcbTitle** combo
row the content of **film** table as shown in Figure 7.4.

Step 5

In **FilmCategory_Utils** class, define a new method named **display_categ**
static method that takes three parameters: a **FilmCategoryForm** object,
and two items of any type T. The method is responsible for displaying
categories in the **FilmCategoryForm**.

Inside the method, a connection to the database is established, and a **Pre**
created using the provided SQL query string. The two items are then set
PreparedStatement, and the query is executed using **executeQuery()** m

If the **ResultSet** returned from the query is empty (no rows found), the
text fields in the **FilmCategoryForm** object and returns. Otherwise,
through each row in the **ResultSet** using a while loop. For each row,
corresponding text fields and determines the selected items in two
(**jcbCategoryID** and **jcbCategoryName**) based on the values of th
category_name columns in the **ResultSet**.

Finally, the method closes the **ResultSet** and **PreparedStatement**
SQLException that may occur during the process by displaying an
JOptionPane.

```
    private static <T> void
display_category_data(FilmCategoryForm frm, String sql, T
item1, T item2) {
        try(Connection conn = getConnection()){
```

```
            PreparedStatement ps =
conn.prepareStatement(sql);
            ps.setObject(1,item1);
            ps.setObject(2,item2);
            ResultSet rs = ps.executeQuery();

            if (!rs.next()) {
                // no row found, clear the form fields
                frm.getJTFLastUpdate().setText("");
                return;
            }

            do {

frm.getJTFCategoryID().setText(rs.getString("category_id"));

frm.getJTFLastUpdate().setText(rs.getString("last_update"));

                // Determines item selected from
jcbCategoryID

find_combo_value_selected(frm.getJCBCategoryID(),
rs.getInt("category_id"));

                // Determines item selected from
jcbCategoryName

find_combo_value_selected(frm.getJCBCategoryName(),
rs.getString("category_name"));
            } while(rs.next());

            rs.close();
            ps.close();
        } catch(SQLException ex){
            JOptionPane.showMessageDialog(frm,
ex.getMessage(), "ERROR", JOptionPane.ERROR_MESSAGE);
        }
    }
```

Step 6

In the same class, define another method named **jcbCategory_hand**
jcbCategory_handler() is a similar implementation to **jcbFilm_handl**
selection of categories from the **JComboBoxes** in **FilmCategoryForm**.
FilmCategoryForm frm and two **JComboBoxes jcb1** and **jcb2**.

First, it retrieves the selected items from both **JComboBoxes item1**
checks which **JComboBox** triggered the event by comparing it to the **JC**
frm. Depending on which **JComboBox** triggered the event, it sets the ap
string to fetch the data from the database.

Finally, it calls the **display_category_data()** method, passing in the object, the SQL query string, and both selected items as parameters retrieves the data from the database using the SQL query and displays it o

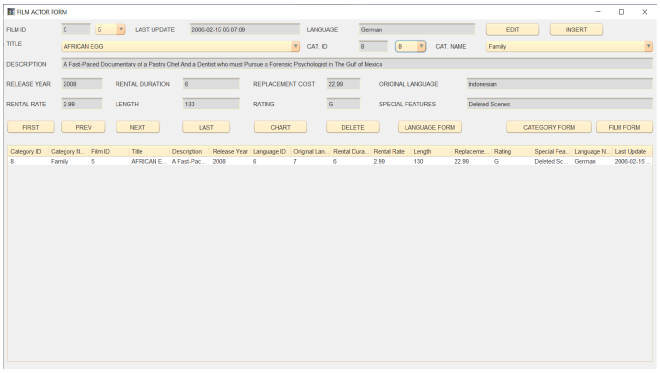
```
    public static void
jcbCategory_handler(FilmCategoryForm frm,
JComboBox<String> jcb1, JComboBox<String> jcb2)
{
        Object item1 = jcb1.getSelectedItem();
        Object item2 = jcb2.getSelectedItem();
        String sql = "";
        if (jcb1.equals(frm.getJCBCategoryID()))
{
            sql =
Query_FilmCategory.get_sql_film_category_joint()
+ " WHERE fc.category_id = ? AND fc.film_id =
?";
        } else if
(jcb1.equals(frm.getJCBCategoryName())) {
            sql =
Query_FilmCategory.get_sql_film_category_joint()
+ " WHERE c.name = ? AND fc.film_id = ?";
        }

        display_category_data(frm, sql, item1,
item2);
    }
```

Step 7

In **FilmCategoryForm**, double click on **jcbCategoryID** and **jcbCategor** and define their corresponding event handler as follows:

```
    private void
jcbCategoryIDActionPerformed(java.awt.event.ActionEvent
evt) {
        FilmCategory_Utils.jcbCategory_handler(this,
this.jcbCategoryID, this.jcbFilmID);
    }

    private void
jcbCategoryNameActionPerformed(java.awt.event.ActionEvent
evt) {
        FilmCategory_Utils.jcbCategory_handler(this,
this.jcbCategoryName, this.jcbFilmID);
    }
```

These methods are event handlers for when the user selects an item ir category name combo boxes. The **jcbCategoryIDActionPerformed(** **jcbCategory_handler()** method from the **FilmCategory_Utils** clas **FilmCategoryForm instance (this)**, the **jcbCategoryID** combo box,

	combo box as arguments. The **jcbCategoryNameActionPerformed()** m thing, but with the **jcbCategoryName** combo box instead. The **jcbCategory_handler()** method takes the **FilmCategoryForm** instan (**jcb1** and **jcb2**), and uses their selected items to construct a SQL query to selected category and film. It then calls the **display_category_data()** me **FilmCategoryForm** instance, the constructed SQL query, and the selecte combo boxes as arguments.
Step 8	Run the project. Choose **film_id** using **jcbFilmID** and/or **jcbTitle** comb **film_id = 5**). Then, choose available **category_id** relating to ch **jcbCategoryID** and/or **jcbCategoryName** comboboxes (in this case, **c** shown in Figure 7.5.  **Figure 7.5** Displaying row by row the content of **category**
Step 9	Define four navigating methods in **FilmCategory_Utils** class. Thes navigating between rows of data in the form. **show_first_row()** displays the first row of data by getting the item at t **JComboBox** for film IDs, passing it to the **display_film_data()** m currentIndex to the first index. **show_last_row()** displays the last row of data by getting the item at t **JComboBox** for film IDs, passing it to the **display_film_data()** m **currentIndex** to the last index. **show_prev_row()** displays the previous row of data by decrementing checking if it is less than the first index. If so, it sets **currentIndex** to returns. Otherwise, it gets the item at the current index of the **JComboBo** it to the **display_film_data()** method, and displays the data. **show_next_row()** displays the next row of data by incrementing **current** if it is greater than the last index. If so, it sets **currentIndex** to the las Otherwise, it gets the item at the current index of the **JComboBox** for film **display_film_data()** method, and displays the data.

```
    public static void show_first_row(FilmCategoryForm frm)
{
        String item =
String.valueOf(frm.getJCBFilmID().getItemAt(FIRST_INDEX));
        display_film_data(frm, SQL_ID, item);
        currentIndex = FIRST_INDEX;
    }

    public static void show_last_row(FilmCategoryForm frm){
        int endIndex = frm.getJCBFilmID().getItemCount() -
1;
        String item =
String.valueOf(frm.getJCBFilmID().getItemAt(endIndex));
        display_film_data(frm, SQL_ID, item);
        currentIndex = endIndex;
    }

    public static void show_prev_row(FilmCategoryForm frm){
        currentIndex--;
        if(currentIndex < FIRST_INDEX){
            currentIndex = FIRST_INDEX;
            return;
        }
        String item =
String.valueOf(frm.getJCBFilmID().getItemAt(currentIndex));
        display_film_data(frm, SQL_ID, item);
    }

    public static void show_next_row(FilmCategoryForm frm){
        int endIndex = frm.getJCBFilmID().getItemCount() -
1;
        currentIndex++;
        if(currentIndex > endIndex){
            currentIndex = endIndex;
            return;
        }
        String item =
String.valueOf(frm.getJCBFilmID().getItemAt(currentIndex));
        display_film_data(frm, SQL_ID, item);
    }
```

Step 10

Then in **FilmCategoryForm**, double click on each navigation butt corresponding event handler:

```
    private void
jbFirstActionPerformed(java.awt.event.ActionEvent
evt) {
       FilmCategory_Utils.show_first_row(this);
    }
```

```
6
7       private void
8   jbLastActionPerformed(java.awt.event.ActionEvent
9   evt) {
10          FilmCategory_Utils.show_last_row(this);
11      }
12
13      private void
14  jbPrevActionPerformed(java.awt.event.ActionEvent
15  evt) {
            FilmCategory_Utils.show_prev_row(this);
        }

        private void
    jbNextActionPerformed(java.awt.event.ActionEvent
    evt) {
            FilmCategory_Utils.show_next_row(this);
        }
```

These are event handler methods for the "First", "Last", "Previous", and " user interface. When the user clicks on one of these buttons, the corresp **FilmCategory_Utils** is called to show the first, last, previous, or next row category table. These methods use the currentIndex field to keep track being displayed and the **display_film_data()** method to show the data for

Step 11

Run the project. Click on one or more navigation buttons to see the resul 7.6.

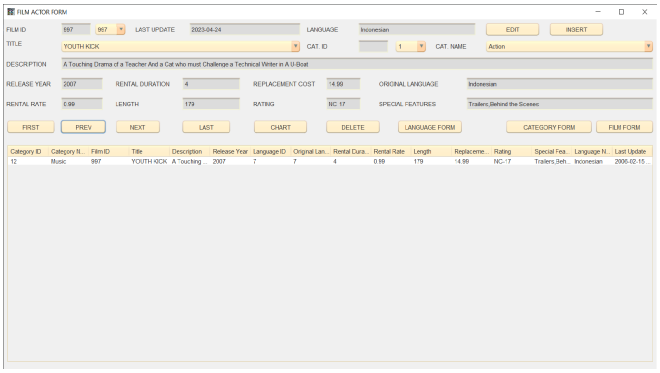


Figure 7.6 User clicks on one or more navigation buttons on film c

Step 12

Define **mouse_pressed_handler()** method in **FilmCategory_Utils** class. the mouse press event in the **JTable** of the **FilmCategoryForm**. It gets th from the **JTable** and checks if a row is actually selected. If not, it displa user and returns. If a row is selected, it retrieves the values of the first a the selected row, which represent the category ID and film ID, respective a SQL query using these values to display the corresponding film cate

display_category_data() method. If an **SQLException** occurs, it logs the message dialog with the error message and stack trace.

```
    public static void mouse_pressed_handler(FilmCategoryForm frm)
        Objects.requireNonNull(frm, "frm must not be null");

        int selectedIndex = frm.getJTFilmCategory().getSelectedRow(
        if (selectedIndex == -1) {
            JOptionPane.showMessageDialog(frm, "Please select a row to view its d
                "No row selected", JOptionPane.INFORMATION_MESSAGE)
            return;
        }

        try (Connection conn = getConnection()) {
            String item1 =
String.valueOf(frm.getJTFilmCategory().getModel().getValueAt(select
0));
            String item2 =
String.valueOf(frm.getJTFilmCategory().getModel().getValueAt(select
2));

            // Displays film category data
            String sql = Query_FilmCategory.get_sql_film_category_j
" WHERE fc.category_id = ? AND fc.film_id = ?";
            display_category_data(frm, sql, item1, item2);

        } catch (SQLException ex) {

Logger.getLogger(FilmCategoryForm.class.getName()).log(Level.SEVERE
displaying film category data", ex);
            String message = "Error displaying film category data:
ex.getMessage();
            String stackTrace = Arrays.toString(ex.getStackTrace())
            JOptionPane.showMessageDialog(frm, message + "\n\n" +
stackTrace, "ERROR", JOptionPane.ERROR_MESSAGE);
        }
    }
```

Step 13

Right click on **jtFilmCategory**. Then, choose **Events > Mouse > mouse** event handler:

```
    private void
jtFilmCategoryMousePressed(java.awt.event.MouseEvent
evt) {

 FilmCategory_Utils.mouse_pressed_handler(this);
    }
```

Step 14	Run the project. Choose **film_id** using **jcbFilmID** and/or **jcbTitle** comb **film_id = 998**). Then, double click on any row in **jtFilmCategory** tab corresponding row in **film_category** table displayed in textfields and com Figure 7.7.

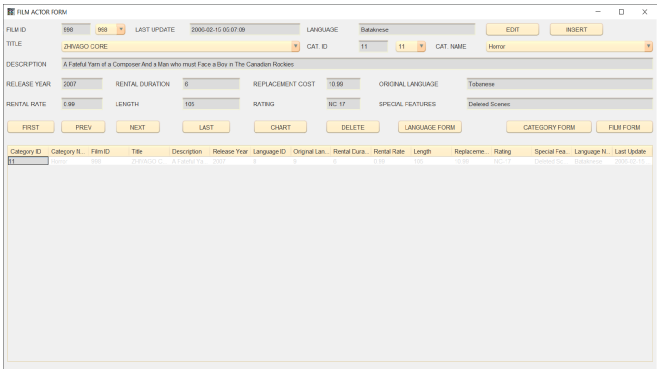


Figure 7.7 User double-clicks on any row in **jtFilmCateg**

UPDATING RECORD
UPDATING RECORD

Step 1	In **FilmCategory_Utils** class, define a new method name **update_row_by_film_cat_id()**. It updates a row of data in **film_catego** table. The method takes in four parameters: a **FilmCategoryForm** objec which is used to display error messages to the user; the ID of the film ar the category that the row corresponds to, and the new category ID that th row should be updated with. The method first establishes a connection to the database usir **getConnection()** method, then performs a SELECT query to check if th specified **film_id** exists in the **film** table. If the **film_id** is not found, a error message is displayed using a **JOptionPane**. If the **film_id** is foun the method creates a new **FilmCategory** object using the three-paramete constructor, where the first parameter is the new category ID, the secor parameter is the film ID, and the third parameter is a new **Timestam** object that represents the current time. The method then creates a **PreparedStatement** object for the UPDAT query, and sets the parameters of the query using the **obj** object create earlier, and the **film_id** and **category_id** parameters passed into th method. The **PreparedStatement** object then executes the UPDATE que using the **executeUpdate()** method.

Finally, the method closes the **ResultSet**, **PreparedStatement**, ar **Connection** objects used in the method. If there is a **SQLException** or **NumberFormatException** thrown during the execution of the method, a error message is displayed using a **JOptionPane**.

```
    //Updates row of data in film_category tabel by film_id and
category_id
    public static void update_row_by_film_cat_id(FilmCategoryForm
frm, int film_id, int cat_id, int cat_id_new) throws SQLException{
        Connection conn = getConnection();
        ResultSet rs = null;
        String query_id = "SELECT film_id FROM film WHERE film_id =
?";
        String update_query = """
            UPDATE film_category SET film_id = ?, category_id = ?
            WHERE film_id = ? AND category_id = ?""";
        try(PreparedStatement idPs = conn.prepareStatement(query_id

ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_READ_ONLY);
        PreparedStatement updatePS =
conn.prepareStatement(update_query,
                ResultSet.TYPE_SCROLL_SENSITIVE,
ResultSet.CONCUR_UPDATABLE))
        {
            idPs.setInt(1,film_id);
            if(!idPs.execute()){
                String message = "Can't find film_id " + film_id;

                JOptionPane.showMessageDialog(frm, message,
                    "ERROR",JOptionPane.ERROR_MESSAGE);
            } else{
                rs = idPs.getResultSet();
                rs.next();

                //Creates a FilmCategory object using three-params
constructor
                FilmCategory obj = new FilmCategory(cat_id, film_id
new Timestamp(System.currentTimeMillis()));
                updatePS.setInt(1, obj.getFilmID());
                updatePS.setInt(2, cat_id_new);
                updatePS.setInt(3, obj.getFilmID());
                updatePS.setInt(4, obj.getCategoryID());

                updatePS.executeUpdate();
                rs.close();
                updatePS.close();
                idPs.close();
                conn.close();
            }
        }catch(SQLException ex){

Logger.getLogger(FilmCategoryForm.class.getName()).log(Level.SEVERE
```

```
"Error updating film-category data", ex);
            String message = "Error updating film-category data: "
ex.getMessage();
            String stackTrace = Arrays.toString(ex.getStackTrace())
            JOptionPane.showMessageDialog(null, message + "\n\n" +
stackTrace, "ERROR", JOptionPane.ERROR_MESSAGE);
        }catch(java.lang.NumberFormatException ex){

Logger.getLogger(FilmCategoryForm.class.getName()).log(Level.SEVERE
"Invalid Input", ex);
            String message = "Invalid Input: " + ex.getMessage();
            String stackTrace = Arrays.toString(ex.getStackTrace())
            JOptionPane.showMessageDialog(null, message + "\n\n" +
stackTrace, "ERROR", JOptionPane.ERROR_MESSAGE);
        }
    }
```

Step 2

Then in the same class, define a new method **read_inputs()**. It reads inp data from a **FilmCategoryForm** object and returns a **HashMap** containi the input values. The input data consists of three fields: film ID, catego ID, and new category ID.

The method first initializes a new **HashMap** to store the input values. then reads the film ID from the selected item in a **JComboBox** componer and the category ID and new category ID from **JTextFields**.

The method then validates the user input. It first checks that the film ID is positive integer. If the film ID is not a valid positive integer, the meth throws a **NumberFormatException** or **IllegalArgumentException** a displays an error message dialog.

The method then checks that the category ID is not null or empty. If t category ID is null or empty, the method throws **IllegalArgumentException** and displays an error message dialog.

Finally, the method checks that the new category ID is not null or empty. the new category ID is null or empty, the method throws **IllegalArgumentException** and displays an error message dialog.

If all input data is valid, the method stores the input values in the **HashM** and returns it.

```
    private static HashMap<String, String>
read_inputs(FilmCategoryForm frm) {
        HashMap<String, String> input_data = new
HashMap<>();

```

```
        String film_id =
String.valueOf(frm.getJCBFilmID().getSelectedItem());
        String cat_id = frm.getJTFCategoryID().getText();
        String cat_id_new =
String.valueOf(frm.getJCBCategoryID().getSelectedItem());

        // Validate user input
        int film_id_int = 0;
        try {
            film_id_int = Integer.parseInt(film_id);
            if (film_id_int <= 0) {
                throw new IllegalArgumentException("Film ID cannot
be negative or zero");
            }
        } catch (NumberFormatException ex) {
            JOptionPane.showMessageDialog(frm, "Invalid
Film ID: " + film_id,
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        } catch (IllegalArgumentException ex) {
            JOptionPane.showMessageDialog(frm,
ex.getMessage(),
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        }

        if (cat_id == null) {
            JOptionPane.showMessageDialog(frm, "Category
ID cannot be empty",
            "Error", JOptionPane.ERROR_MESSAGE);
            throw new IllegalArgumentException("Category
ID cannot be empty");
        }

        if (cat_id_new == null || cat_id_new.isEmpty()) {
            JOptionPane.showMessageDialog(frm, "New
Category ID cannot be empty",
            "Error", JOptionPane.ERROR_MESSAGE);
            throw new IllegalArgumentException("Ner Category ID
cannot be empty");
        }

        input_data.put("film_id", film_id);
        input_data.put("cat_id", cat_id);
        input_data.put("cat_id_new", cat_id_new);

        return input_data;
    }
```

Step 3	Still in the same class, define another method named **edit_actual()**. updates a row in a database table with new values for the film category. Tl

method takes a **FilmCategoryForm** object as input.

The method first calls the **read_inputs()** method to read the input valu for film ID, category ID, and new category ID from the **FilmCategoryFor** object. It then converts the input values to integers using tl **Integer.parseInt()** method.

The method then calls the **update_row_by_film_cat_id()** method with tl **FilmCategoryForm** object, film ID, category ID, and new category ID arguments. This method updates the corresponding row in the databa table with the new values.

After updating the database, the method calls the **refresh_controls** method to refresh all objects on the form.

If the database update operation encounters an **SQLException**, the metho catches the exception and displays an error message dialog with tl exception message.

```
1     private static void
2  edit_actual(FilmCategoryForm frm){
3          try{
4              HashMap<String, String> input_data
5  = read_inputs(frm);
6              int film_id =
7  Integer.parseInt(input_data.get("film_id"));
8              int cat_id =
9  Integer.parseInt(input_data.get("cat_id"));
10             int cat_id_new =
11 Integer.parseInt(input_data.get("cat_id_new"));
12
13             update_row_by_film_cat_id(frm,
14 film_id, cat_id, cat_id_new);
15
16             //Refreshes all objects on form
17             refresh_controls(frm);

          }catch(SQLException ex){
              JOptionPane.showMessageDialog(frm,
ex.getMessage(),

 "ERROR",JOptionPane.ERROR_MESSAGE);
          }
    }
```

Step 4

Lastly, define two new methods named **enable_controls()** an **edit_handler()**. It handles the edit functionality in the **FilmCategoryFor** object. The method takes a **FilmCategoryForm** object as input.

The method first checks the text of the "Edit" button on the form. If the te is "EDIT", the method changes the text to "CONFIRM", indicating that tl user wants to confirm the editing action. The method then calls tl **enable_controls()** method with the boolean value "false" and tl **FilmCategoryForm** object as arguments to disable all controls on the forr

If the text of the "Edit" button is "CONFIRM", the method changes the te back to "EDIT". The method then calls the **edit_actual()** method with tl **FilmCategoryForm** object as an argument to perform the actual editir operation in the database table. After editing, the method calls tl **enable_controls()** method with the boolean value "true" and tl **FilmCategoryForm** object as arguments to enable all controls on the forn

```
1     private static void enable_controls(boolean
2  state, FilmCategoryForm frm){
3          frm.getJBFirst().setEnabled(state);
4          frm.getJBPrev().setEnabled(state);
5          frm.getJBNext().setEnabled(state);
6          frm.getJBLast().setEnabled(state);
7          frm.getJBInsert().setEnabled(state);
8          frm.getJBDelete().setEnabled(state);
9          frm.getJTFFilmID().setEnabled(state);
10         frm.getJTFDescription().setEnabled(state);
11         frm.getJTFReleaseYear().setEnabled(state);
12         frm.getJTFLanguageName().setEnabled(state);
13
14 frm.getJTFOriginalLanguageName().setEnabled(state);
15
16 frm.getJTFRentalDuration().setEnabled(state);
17         frm.getJTFRentalRate().setEnabled(state);
18         frm.getJTFLength().setEnabled(state);
19
20 frm.getJTFReplacementCost().setEnabled(state);
21
22 frm.getJTFSpecialFeatures().setEnabled(state);
23         frm.getJTFLastUpdate().setEnabled(state);
24     }
25
26     public static void
27 edit_handler(FilmCategoryForm frm){
28
29  if(frm.getJBEdit().getText().equals("EDIT")){
30             frm.getJBEdit().setText("CONFIRM");
31
32             // Disables controls
33             enable_controls(false, frm);
34         }
35
36         else {
37             frm.getJBEdit().setText("EDIT");
```

```
38                  // Actual editing
                    edit_actual(frm);

                    //Enables controls
                    enable_controls(true, frm);
                }
            }
```

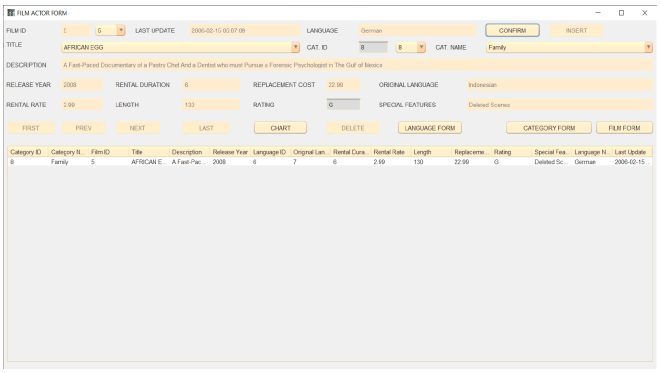


Figure 7.8 The film category form is in editing state

Step 5

In **FilmCategoryForm.java**, double click on EDIT button to create its eve listener:

```
1     private void
2 jbEditActionPerformed(java.awt.event.ActionEvent
3 evt) {
          FilmCategory_Utils.edit_handler(this);
      }
```

Step 6

Run the project. Choose **film_id** using **jcbFilmID** or **jcbTitle** combobox (this case, **film_id = 5**). Then, choose available **category_id** relating chosen **film_id** using **jcbCategoryID** or **jcbCategoryName** (in this cas **category_id = 8**). Then, click on EDIT button as shown in Figure 7.8.

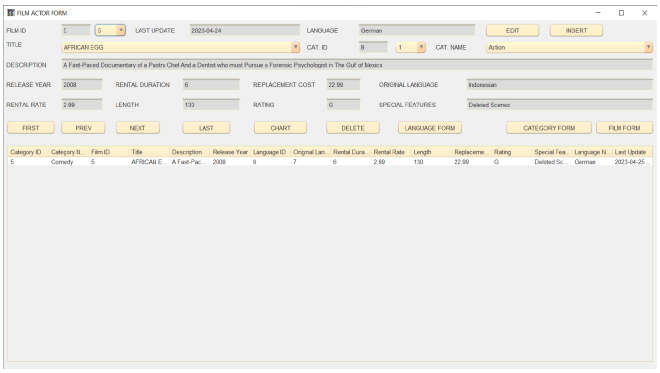


Figure 7.9 The edited row had been saved into database

Then, choose new **category_id** using **jcbCategoryID** or **jcbCategoryNam** (in this case, **category_id = 5**). Click on CONFIRM button. The edited rc had been saved into **film_category** table as shown in Figure 7.9.

INSERTING NEW RECORD

INSERTING NEW RECORD

Step 1

In **FilmCategory_Utils** class, define a method named **insert_row()**. This method inserts a new row into the **film_category** table in the database. It reads the input data from the form using the **read_inputs()** method and then creates a new **FilmCategory** object using the three-parameter constructor with **cat_id_new**, **film_id**, and the current timestamp. It then sets the values of the parameters in the SQL insert statement and executes the statement using the **executeUpdate()** method of the **PreparedStatement** object.

If an **SQLException** is caught, it is logged and an error message is displayed to the user using **JOptionPane.showMessageDialog**.

```
1   //Inserts new row into film_category table
2   private static void insert_row(FilmCategoryForm frm) throws
3   SQLException{
4           HashMap<String, String> input_data = read_inputs(frm);
5           int film_id =
6   Integer.parseInt(input_data.get("film_id"));
7           int cat_id_new =
8   Integer.parseInt(input_data.get("cat_id_new"));
9
10          // SQL insert statement
11          String sql = """
12              INSERT INTO film_category(film_id, category_id)
13                  VALUES(?, ?)""";
14
15          try(Connection conn = getConnection();
```

```
16          PreparedStatement pstmt = conn.prepareStatement(sql)){
17
18              //Creates a FilmCategory object three-params
19  constructor
20              FilmCategory obj = new FilmCategory(cat_id_new,
21  film_id, new Timestamp(System.currentTimeMillis()));
22              pstmt.setInt(1,obj.getFilmID());
23              pstmt.setInt(2,obj.getCategoryID());
24
25              //Executes the sql insert statement
26              pstmt.executeUpdate();
27          } catch (SQLException ex) {
28
29  Logger.getLogger(FilmActorForm.class.getName()).log(Level.SEVERE,
    "Database error", ex);
                JOptionPane.showMessageDialog(frm, "Error: Database
    error\n" + ex.getMessage());
            }
        }
```

Step 2

Still in **FilmCategory_Utils.java**, define **insert_actual()** and **insert_handler()** methods. The **insert_actual()** method inserts a new row into **film_category** table using the information entered in a form (**FilmCategoryForm**), and then refreshes the table and related combo boxes. If there is a SQL exception, it displays an error message dialog with the exception message.

The **insert_handler()** method is an event handler for a button (**frm.getJBInsert()**). When the button is first clicked and its label is "INSERT", it changes the button label to "CONFIRM", disables the "Edit" button (**frm.getJBEdit()**), disables the controls in the form, and enables the "Insert" button. When the button is clicked again and its label is "CONFIRM", it changes the label back to "INSERT", inserts the data into the database using the **insert_actual()** method, enables the "Edit" button, and enables the controls in the form.

```
1       private static void
2   insert_actual(FilmCategoryForm frm){
3           try{
4               insert_row(frm);
5
6               //Refreshes table and comboboxes
7               refresh_controls(frm);
8
9           }catch(SQLException ex){
10              JOptionPane.showMessageDialog(frm,
11  ex.getMessage(),
12
```

```
13
14  "ERROR",JOptionPane.ERROR_MESSAGE);
15          }
16      }
17
18      public static void
19  insert_handler(FilmCategoryForm frm){
20
21   if(frm.getJBInsert().getText().equals("INSERT")
22  ){
23
24  frm.getJBInsert().setText("CONFIRM");
25
26              //Disables jbEdit
27              frm.getJBEdit().setEnabled(false);
28
29              // Disables controls
30              enable_controls(false, frm);
31
32              // Enables
33              frm.getJBInsert().setEnabled(true);
34          }
35
36          else {
37              frm.getJBInsert().setText("INSERT");
38
39              // Actual insertion
40              insert_actual(frm);

                //Enables jbEdit
                frm.getJBEdit().setEnabled(true);

                //Enables controls
                enable_controls(true, frm);
            }
        }
```

Step 3

In **FilmCategoryForm.java**, double click on INSERT button to create its event listener:

```
1      private void
2  jbInsertActionPerformed(java.awt.event.ActionEvent
3  evt) {
           FilmCategory_Utils.insert_handler(this);
       }
```

Step 4

Run the project. Choose **film_id** using **jcbFilmID** or **jcbTitle** combobox (in this case, **film_id = 3**). Click on INSERT button. You will see the

state of film category form when insertion is in progress as shown in Figure 7.10.

Then, choose **category_id** that is not available relating to chosen **film_id** using **jcbCategoryID** or **jcbCategoryName** (in this case, **category_id = 3**). Then, click CONFIRM button to save the new record into **film_category** table as shown in Figure 7.11.

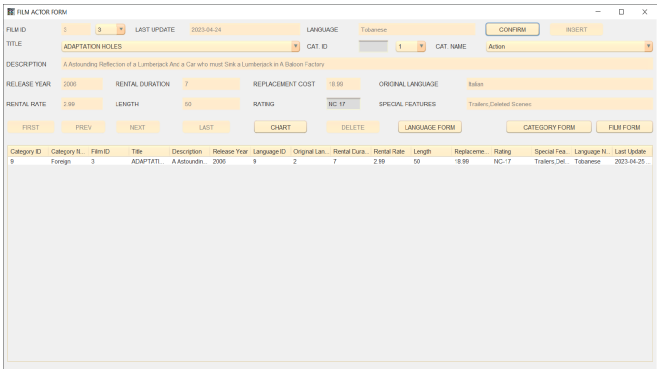


Figure 7.10 When user clicks on INSERT button, the film category form will be in state of insertion

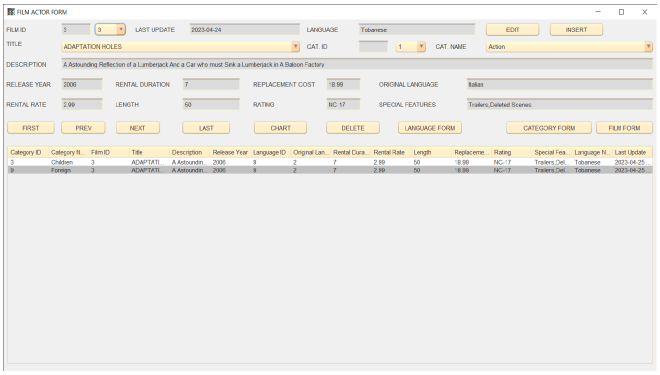


Figure 7.11 The new data had been saved into **film_category** table

DELETING RECORD
DELETING RECORD

Step 1 Then in **FilmCategory_Utils** class, define **delete_handler()** method.

```
1      public static void delete_handler(FilmCategoryForm frm){
2          int dialogButton = JOptionPane.YES_NO_OPTION;
3          int film_id =
4  Integer.parseInt(String.valueOf(frm.getJCBFilmID().getSelectedItem(
5
```

```
        int cat_id =
Integer.parseInt(String.valueOf(frm.getJCBCategoryID().getSelectedI

        String message = String.format("Are you sure you want to de
row (Film ID: %d, Category ID: %d)", film_id, cat_id);
        int answer = JOptionPane.showConfirmDialog(frm, message, "D
ROW OF DATA", dialogButton);

        if(answer == JOptionPane.YES_OPTION){
            String query = """
                DELETE FROM film_category WHERE film_id = ? AND category_id = ?"""
            try(Connection conn = getConnection();
                PreparedStatement ps = conn.prepareStatement(query)
                // Use PreparedStatement to avoid SQL injection att
                ps.setInt(1, film_id);
                ps.setInt(2, cat_id);
                ps.executeUpdate();

                // Refresh table and comboboxes
                refresh_controls(frm);

            } catch (SQLException ex){
                JOptionPane.showMessageDialog(frm, ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
            }
        }
    }
```

Step 2

In **FilmCategoryForm.java**, double click on DELETE button to generate listener:

```
    private void
jbDeleteActionPerformed(java.awt.event.ActionEvent
evt) {
        FilmCategory_Utils.delete_handler(this);
    }
```

Step 3

Run the project. Choose **film_id** using **jcbFilmID** or **jcbFilm** combobox ar **category_id** using **jcbCategoryID** or **jcbCategoryName** combobox Then DELETE button. The corresponding row of data had been deleted from dat

PLOTTING CHART
PLOTTING CHART

Step 1	Create a new **JFrame** and save it as **Charts_FilmCategory.java**.
Step 2	In **Charts_FilmCategory.java**, add four **JPanels** and set thei corresponding **Variable Name** as **jPanel1**, **jPanel2**, **jPanel3**, an **jPanel4**. Then, add getter method for each object as follows:
Step 3	In **FilmCategory_Utils** class, define four new methods. These are fou private static methods that draw different types of charts based on filn categories. The first method **draw_pie_chart_film_by_category()** draws a pie char that represents the distribution of films by category name. It takes tw parameters, **frm** and **jp**, where **frm** is an instance of th **Charts_FilmCategory** class, and **jp** is a **JPanel** object. The second metho **draw_bar_chart_category_by_avg_replacement_cost()** draws a ba chart that represents the distribution of film categories by averag replacement cost. It takes the same parameters as the previous method. The third method **draw_bar_chart_category_by_avg_rental_rate()** draw a bar chart that represents the distribution of film categories by averag rental rate. It also takes the same parameters as the previous methods. The fourth method **draw_bar_chart_category_by_avg_length()** draws bar chart that represents the distribution of film categories by averag

```
1       //Getter method for jPanel1
2       public JPanel getJPanel1(){
3           return this.jPanel1;
4       }
5
6       //Getter method for jPanel2
7       public JPanel getJPanel2(){
8           return this.jPanel2;
9       }
10
11      //Getter method for jPanel3
12      public JPanel getJPanel3(){
13          return this.jPanel3;
14      }
15
16      //Getter method for jPanel4
17      public JPanel getJPanel4(){
18          return this.jPanel4;
19      }
```

length. It takes the same parameters as the previous methods.

All of these methods set the preferred size of the **jp JPanel** object, create a dataset using the **create_pie_dataset()** or **create_bar_dataset(** methods, and then draw a chart using the **draw_piechart_with_dataset(** or **draw_barchart_with_dataset()** methods.

```
    private static void
draw_pie_chart_film_by_category(Charts_FilmCategory frm, JPanel jp)
{
        jp.setPreferredSize(new Dimension(jp.getWidth(),
jp.getHeight()));
        DefaultPieDataset dataset =
create_pie_dataset(Query_FilmCategory.get_sql_film_category_dist(),
"Number", "category_name");

        //Draws piechart film distribution by category name
        draw_piechart_with_dataset(frm, jp, dataset, "TOP 10 FILM
DISTRIBUTION BY CATEGORY NAME");
    }

    private static void
draw_bar_chart_category_by_avg_replacement_cost(Charts_FilmCategory
frm, JPanel jp){
        jp.setPreferredSize(new Dimension(jp.getWidth(),
jp.getHeight()));

        DefaultCategoryDataset dataset =
create_bar_dataset(Query_FilmCategory.get_sql_film_category_dist(),
"avg_replacement_cost", "category_name");

        //Draws barchart category distribution by average
replacement cost
        draw_barchart_with_dataset(frm, jp, dataset, "THE CATEGORY
DISTRIBUTION BY AVERAGE REPLACEMENT COST", "CATEGORY NAME", "AVG
REPLACEMENT COST");
    }

    private static void
draw_bar_chart_category_by_avg_rental_rate(Charts_FilmCategory frm,
JPanel jp){
        jp.setPreferredSize(new Dimension(jp.getWidth(),
jp.getHeight()));

        DefaultCategoryDataset dataset =
create_bar_dataset(Query_FilmCategory.get_sql_film_category_dist(),
"avg_rental_rate", "category_name");

        //Draws barchart category distribution by average rental
rate
        draw_barchart_with_dataset(frm, jp, dataset, "THE CATEGORY
DISTRIBUTION BY AVERAGE RENTAL RATE", "CATEGORY NAME", "AVG RENTAL
```

```
RATE");
    }

    private static void
draw_bar_chart_category_by_avg_length(Charts_FilmCategory frm,
JPanel jp){
        jp.setPreferredSize(new Dimension(jp.getWidth(),
jp.getHeight()));

        DefaultCategoryDataset dataset =
create_bar_dataset(Query_FilmCategory.get_sql_film_category_dist(),
"avg_length", "category_name");

        //Draws barchart category distribution by average length
        draw_barchart_with_dataset(frm, jp, dataset, "THE CATEGORY
DISTRIBUTION BY LENGTH", "CATEGORY NAME", "AVG LENGTH");
    }
```

Step 4

In **FilmCategory_Utils** class, define a new method name **jbchart_handler()**.

```
    public static void
jbchart_handler(Charts_FilmCategory frm){
        //Draws piechart film distribution by
category name
        draw_pie_chart_film_by_category(frm,
frm.getJPanel1());

        //Draws barchart category distribution by
average replacement cost

draw_bar_chart_category_by_avg_replacement_cost(frm,
frm.getJPanel2());

        //Draws barchart category distribution by
average rental rate

draw_bar_chart_category_by_avg_rental_rate(frm,
frm.getJPanel3());

        //Draws barchart category distribution by
average length
        draw_bar_chart_category_by_avg_length(frm,
frm.getJPanel4());
    }
```

Step 5

In **FilmCategoryForm**, double click on **jbChart** button to define its even listener:

```
    private void
jbChartActionPerformed(java.awt.event.ActionEvent
evt) {
        Charts_FilmCategory frm1 = new
Charts_FilmCategory();
        frm1.setLocationRelativeTo(null);
        frm1.setTitle("FOUR DISTRIBUTIONS IN
FILM-CATEGORY TABLE");
        frm1.setVisible(true);
        FilmCategory_Utils.jbchart_handler(frm1);
    }
```

Step 6

Run the project. Click on CHART button on the form. You will see the fou charts displayed on the panels as shown in Figure 7.12.

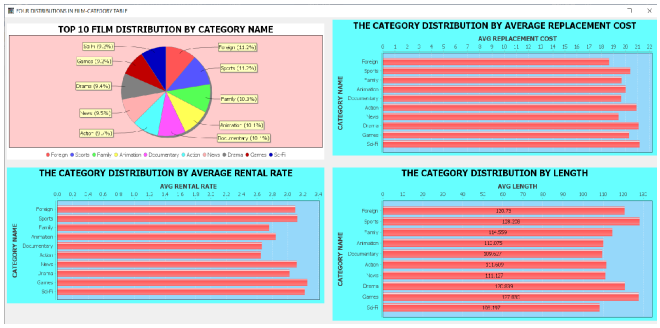


Figure 7.12 The top 10 film distribution by category name, the category distribution by average replacement cost, the category distribution by average rental rate, and the category distribution by length

This is the full version of **FilmCategory_Utils.java**:

```
package sakila;
import java.awt.Dimension;
import java.util.logging.Level;
import java.util.logging.Logger;
import java.sql.*;
import java.util.logging.Level;
import java.util.logging.Logger;
import java.sql.*;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashMap;
import java.util.LinkedHashMap;
import java.util.List;
import java.util.Map;
import java.util.Objects;
```

```
import javax.swing.JComboBox;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.event.TableModelEvent;
import javax.swing.table.DefaultTableModel;
import javax.swing.table.TableModel;
import org.jfree.data.category.DefaultCategoryDataset;
import org.jfree.data.general.DefaultPieDataset;

public class FilmCategory_Utils extends Utility{
    public static final int FIRST_INDEX = 0;
    public static final int INVALID_INDEX = -1;

    private static int currentIndex = FIRST_INDEX;
    private static final String SQL_ID =
Query_FilmCategory.get_sql_film_category_joint() + " WHERE
fc.film_id = ?";

    //Creates film_category table
    public static void create_film_category_table() {
        try (Connection conn = getConnection()) {
            Statement stmt = conn.createStatement();

stmt.addBatch(Query_FilmCategory.get_sql_film_category());
            stmt.executeBatch();

            String message = String.format("Successfully
creates film_category table");
            JOptionPane.showMessageDialog(null, message,

 "INFORMATION",JOptionPane.INFORMATION_MESSAGE);

        } catch (SQLException ex) {
            JOptionPane.showMessageDialog(null,
ex.getMessage(),
                "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    //Populates film_category table with some rows of data
    public static void populate_film_category_table(){
        try(Connection conn = getConnection()){
            String sql = """
                  INSERT INTO film_category(category_id,
film_id, last_update)
                  VALUES(?, ?, ?)""";

            PreparedStatement ps1 =
conn.prepareStatement(sql);
            //Creates a new FilmCategory class with
default constructor
            FilmCategory obj1 = new FilmCategory();
            ps1.setInt(1,obj1.getCategoryID());
```

```
            ps1.setInt(2, obj1.getFilmID());
            ps1.setTimestamp(3, obj1.getLastUpdate());

            PreparedStatement ps2 =
conn.prepareStatement(sql);
            //Creates a new FilmCategory class with three-
params constructor
            FilmCategory obj2 = new FilmCategory(2, 2, new
Timestamp(System.currentTimeMillis()));
            ps2.setInt(1,obj2.getCategoryID());
            ps2.setInt(2, obj2.getFilmID());
            ps2.setTimestamp(3, obj2.getLastUpdate());

            ps1.executeUpdate();
            ps2.executeUpdate();

        }catch(SQLException ex){
            JOptionPane.showMessageDialog(null,
ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
```

```
        }
    }

    //Reads the content of joined film_category, category, film, and language
    public static void read_film_category_table(){
        try(Connection conn = getConnection()){
            Statement stmt = conn.createStatement();
            ResultSet rs =
stmt.executeQuery(Query_FilmCategory.get_sql_film_category_joint());
            while(rs.next()){
                int cat_id = rs.getInt("category_id");
                int film_id = rs.getInt("film_id");
                String title = rs.getString("title");
                String description = rs.getString("description");
                int year = rs.getInt("release_year");
                int lang_id = rs.getInt("language_id");
                int ori_lang_id = rs.getInt("original_language_id");
                int duration = rs.getInt("rental_duration");
                double rate = rs.getDouble("rental_rate");
                int length = rs.getInt("length");
                double cost = rs.getDouble("replacement_cost");
                String rating = rs.getString("rating");
                String features = rs.getString("special_features");
                String lang_name = rs.getString("language_name");
                String cat_name = rs.getString("category_name");
                Timestamp lu = rs.getTimestamp("last_update");

                //Creates an FilmCategory object using sixteen-params constru
                FilmCategory obj = new FilmCategory(cat_id, film_id, title,
description, year, lang_id, ori_lang_id,
                    duration, rate, length, cost, rating, features, lang_name
cat_name, lu);
                System.out.println(obj);
            }
            rs.close();
            stmt.close();

        }catch(SQLException ex){
            JOptionPane.showMessageDialog(null, ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    private static ArrayList<FilmCategory> get_film_cat_list(FilmCategoryForm
String sql, String item){
        ArrayList<FilmCategory> list = new ArrayList<>();
        Connection conv = null;

        try(Connection conn = getConnection();
            PreparedStatement ps = conn.prepareStatement(sql)){
            if (item.equalsIgnoreCase("none")==false) {
                 ps.setString(1,item);
            }
```

```
            ResultSet rs = ps.executeQuery();

            FilmCategory obj;
            while(rs.next()){
                //Using sixteen-params constructor
                obj = new FilmCategory(rs.getInt("category_id"), rs.getInt("
                    rs.getString("title"),
                    rs.getString("description"), rs.getInt("release_year"),
                    rs.getInt("language_id"), rs.getInt("original_language_id
                    rs.getInt("rental_duration"), rs.getDouble("rental_rate"),
                    rs.getInt("length"), rs.getDouble("replacement_cost"),
                    rs.getString("rating"), rs.getString("special_features"),
                    rs.getString("language_name"), rs.getString("category_nam
                        rs.getTimestamp("last_update"));

                list.add(obj);
            }
        }catch (SQLException ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
                "ERROR",JOptionPane.ERROR_MESSAGE);
        }
        return list;
    }

    private static void show_table_film_cat(FilmCategoryForm frm,
ArrayList<FilmCategory> list) throws SQLException{
        DefaultTableModel model = new DefaultTableModel(0,0);

        String header[] = {"Category ID", "Category Name", "Film ID", "Title",
"Description", "Release Year",
            "Language ID", "Original Language", "Rental Duration", "Rental Rate",
"Length",
            "Replacement Cost", "Rating", "Special Features", "Language Name",
Update"};

        model.setColumnIdentifiers(set_column_header(frm.getJTFilmCategory(),
header));
        frm.getJTFilmCategory().setModel(model);

        Object[] row = new Object[16];

        for(int i=0; i<list.size(); i++){
            row[0] = list.get(i).getCategoryID();
            row[1] = list.get(i).getCategoryName();
            row[2] = list.get(i).getFilmID();
            row[3] = list.get(i).getTitle();
            row[4] = list.get(i).getDescription();
            row[5] = list.get(i).getReleaseYear();
            row[6] = list.get(i).getLanguageID();
            row[7] = list.get(i).getOriginalLanguageID();
            row[8] = list.get(i).getRentalDuration();
            row[9] = list.get(i).getRentalRate();
            row[10] = list.get(i).getLength();
            row[11] = list.get(i).getReplacementCost();
```

```
            row[12] = list.get(i).getRating();
            row[13] = list.get(i).getSpecialFeatures();
            row[14] = list.get(i).getLanguageName();
            row[15] = list.get(i).getLastUpdate();

            model.addRow(row);
        }
    }

    public static void refresh_controls(FilmCategoryForm frm){
        frm.setLocationRelativeTo(null);
        frm.setTitle("FILM ACTOR FORM");

        //Shows the content of film table and populates combobox
        try{
            //Makes alternating color for table rows
            table_renderer(frm.getJTFilmCategory());

            //Populates table
            ArrayList<FilmCategory> list = get_film_cat_list(frm,
Query_FilmCategory.get_sql_film_category_joint() + " ORDER BY f.film_id", "no
            show_table_film_cat(frm, list);

            //Populates jcbFilmID
            String sql_id = "SELECT film_id FROM film ORDER BY film_id";
            populate_combobox(sql_id, frm.getJCBFilmID(), frm);

            //Populates getJCBTitle()
            String sql_title = "SELECT DISTINCT title FROM film ORDER BY titl
            populate_combobox(sql_title, frm.getJCBTitle(), frm);

            //Populates jcbCategoryID
            String sql_cat_id = "SELECT category_id FROM category ORDER BY
category_id";
            populate_combobox(sql_cat_id, frm.getJCBCategoryID(), frm);

            //Populates jcbCategoryName
            String sql_cat_name = "SELECT DISTINCT name FROM category ORDER B
            populate_combobox(sql_cat_name, frm.getJCBCategoryName(), frm);

        }catch (SQLException ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
            "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    private static void clear_controls(FilmCategoryForm frm){
        frm.getJTFFilmID().setText("");
        frm.getJTFDescription().setText("");
        frm.getJTFReleaseYear().setText("");
        frm.getJTFLanguageName().setText("");
        frm.getJTFOriginalLanguageName().setText("");
        frm.getJTFRentalDuration().setText("");
```

```
        frm.getJTFRentalRate().setText("");
        frm.getJTFLength().setText("");
        frm.getJTFReplacementCost().setText("");
        frm.getJTFSpecialFeatures().setText("");
        frm.getJTFLastUpdate().setText("");
    }

   //Displays film data result row by row
    private static <T> void display_film_data(FilmCategoryForm frm, String sc
item){
        try(Connection conn = getConnection()){
            PreparedStatement ps = conn.prepareStatement(sql);
            ps.setObject(1,item);
            ResultSet rs = ps.executeQuery();

            if (!rs.next()) {
                // no row found, clear the form fields
                clear_controls(frm);
                return;
            }

            do{
                frm.getJTFFilmID().setText(String.valueOf(rs.getInt("film_id"
                frm.getJTFDescription().setText(rs.getString("description"));

frm.getJTFReleaseYear().setText(String.valueOf(rs.getInt("release_year")));
                frm.getJTFLanguageName().setText(rs.getString("language_name"

frm.getJTFRentalDuration().setText(String.valueOf(rs.getInt("rental_duration"

frm.getJTFRentalRate().setText(String.valueOf(rs.getDouble("rental_rate")));
                frm.getJTFLength().setText(String.valueOf(rs.getInt("length"

frm.getJTFReplacementCost().setText(String.valueOf(rs.getDouble("replacement_
                frm.getJTFSpecialFeatures().setText(rs.getString("special_fea
                frm.getJTFRating().setText(rs.getString("rating"));

frm.getJTFLastUpdate().setText(String.valueOf(rs.getDate("last_update")));

                // Determines item selected from jcbFilmID
                find_combo_value_selected(frm.getJCBFilmID(), rs.getInt("filn

                // Determines item selected from jcbTitle
                find_combo_value_selected(frm.getJCBTitle(), rs.getString("ti

               // Finds original language name
                int ori_lang_id = rs.getInt("original_language_id");
                if(ori_lang_id != 0) {
                    Object lang_name = get_val_from_database("language", "nam
"language_id", ori_lang_id);

frm.getJTFOriginalLanguageName().setText(String.valueOf(lang_name));
```

```
                }

            }while(rs.next());

            rs.close();
            ps.close();
        }catch(SQLException ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    public static void jcbFilm_handler(FilmCategoryForm frm, JComboBox<String
        Object item = jcb.getSelectedItem();
        String sql = "";
        if (jcb.equals(frm.getJCBFilmID())) {
            sql = Query_Film.get_sql_film_joint() + " WHERE f.film_id = ?";

            try{
                //Filters jtFilmCategory
                ArrayList<FilmCategory> list = get_film_cat_list(frm,
Query_FilmCategory.get_sql_film_category_joint() + " WHERE fc.film_id = ?",
String.valueOf(item));
                show_table_film_cat(frm, list);
            }catch (SQLException ex){
                JOptionPane.showMessageDialog(frm, ex.getMessage(),
                "ERROR",JOptionPane.ERROR_MESSAGE);
            }
        } else if (jcb.equals(frm.getJCBTitle())) {
            sql = Query_Film.get_sql_film_joint() + " WHERE f.title = ?";

            try{
                //Filters jtFilmCategory
                ArrayList<FilmCategory> list = get_film_cat_list(frm,
Query_FilmCategory.get_sql_film_category_joint() + " WHERE f.title = ?",
String.valueOf(item));
                show_table_film_cat(frm, list);
            }catch (SQLException ex){
                JOptionPane.showMessageDialog(frm, ex.getMessage(),
                "ERROR",JOptionPane.ERROR_MESSAGE);
            }
        }

        display_film_data(frm, sql, item);
    }

    private static <T> void display_category_data(FilmCategoryForm frm, Strir
item1, T item2) {
        try(Connection conn = getConnection()){
            PreparedStatement ps = conn.prepareStatement(sql);
            ps.setObject(1,item1);
            ps.setObject(2,item2);
            ResultSet rs = ps.executeQuery();
```

```
            if (!rs.next()) {
                // no row found, clear the form fields
                frm.getJTFLastUpdate().setText("");
                return;
            }

            do {
                frm.getJTFCategoryID().setText(rs.getString("category_id"));
                frm.getJTFLastUpdate().setText(rs.getString("last_update"));

                // Determines item selected from jcbCategoryID
                find_combo_value_selected(frm.getJCBCategoryID(),
rs.getInt("category_id"));

                // Determines item selected from jcbCategoryName
                find_combo_value_selected(frm.getJCBCategoryName(),
rs.getString("category_name"));
            } while(rs.next());

            rs.close();
            ps.close();
        } catch(SQLException ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(), "ERROR",
JOptionPane.ERROR_MESSAGE);
        }
    }

    public static void jcbCategory_handler(FilmCategoryForm frm, JComboBox<St
jcb1, JComboBox<String> jcb2) {
        Object item1 = jcb1.getSelectedItem();
        Object item2 = jcb2.getSelectedItem();
        String sql = "";
        if (jcb1.equals(frm.getJCBCategoryID())) {
            sql = Query_FilmCategory.get_sql_film_category_joint() + " WHERE
fc.category_id = ? AND fc.film_id = ?";
        } else if (jcb1.equals(frm.getJCBCategoryName())) {
            sql = Query_FilmCategory.get_sql_film_category_joint() + " WHERE
? AND fc.film_id = ?";
        }

        display_category_data(frm, sql, item1, item2);
    }

    public static void show_first_row(FilmCategoryForm frm){
        String item = String.valueOf(frm.getJCBFilmID().getItemAt(FIRST_INDE
        display_film_data(frm, SQL_ID, item);
        currentIndex = FIRST_INDEX;
    }

    public static void show_last_row(FilmCategoryForm frm){
        int endIndex = frm.getJCBFilmID().getItemCount() - 1;
        String item = String.valueOf(frm.getJCBFilmID().getItemAt(endIndex));
```

```
        display_film_data(frm, SQL_ID, item);
        currentIndex = endIndex;
    }

    public static void show_prev_row(FilmCategoryForm frm){
        currentIndex--;
        if(currentIndex < FIRST_INDEX){
            currentIndex = FIRST_INDEX;
            return;
        }
        String item = String.valueOf(frm.getJCBFilmID().getItemAt(currentInde
        display_film_data(frm, SQL_ID, item);
    }

    public static void show_next_row(FilmCategoryForm frm){
        int endIndex = frm.getJCBFilmID().getItemCount() - 1;
        currentIndex++;
        if(currentIndex > endIndex){
            currentIndex = endIndex;
            return;
        }
        String item = String.valueOf(frm.getJCBFilmID().getItemAt(currentInde
        display_film_data(frm, SQL_ID, item);
    }

    public static void mouse_pressed_handler(FilmCategoryForm frm) {
        Objects.requireNonNull(frm, "frm must not be null");

        int selectedIndex = frm.getJTFilmCategory().getSelectedRow();
        if (selectedIndex == -1) {
            JOptionPane.showMessageDialog(frm, "Please select a row to view 
data.",
                "No row selected", JOptionPane.INFORMATION_MESSAGE);
            return;
        }

        try (Connection conn = getConnection()) {
            String item1 =
String.valueOf(frm.getJTFilmCategory().getModel().getValueAt(selectedIndex, 
            String item2 =
String.valueOf(frm.getJTFilmCategory().getModel().getValueAt(selectedIndex, 

            // Displays film category data
            String sql = Query_FilmCategory.get_sql_film_category_joint() + "
fc.category_id = ? AND fc.film_id = ?";
            display_category_data(frm, sql, item1, item2);

        } catch (SQLException ex) {
            Logger.getLogger(FilmCategoryForm.class.getName()).log(Level.SEVE
"Error displaying film category data", ex);
            String message = "Error displaying film category data: " +
ex.getMessage();
            String stackTrace = Arrays.toString(ex.getStackTrace());
```

```
            JOptionPane.showMessageDialog(frm, message + "\n\n" + stackTrace,
JOptionPane.ERROR_MESSAGE);
        }
    }

    //Updates row of data in film_category tabel by film_id and category_id
    public static void update_row_by_film_cat_id(FilmCategoryForm frm, int fi
int cat_id, int cat_id_new) throws SQLException{
        Connection conn = getConnection();
        ResultSet rs = null;
        String query_id = "SELECT film_id FROM film WHERE film_id = ?";
        String update_query = """
            UPDATE film_category SET film_id = ?, category_id = ?
            WHERE film_id = ? AND category_id = ?""";
        try(PreparedStatement idPs = conn.prepareStatement(query_id,
                    ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_READ_ONL
        PreparedStatement updatePS = conn.prepareStatement(update_query,
                ResultSet.TYPE_SCROLL_SENSITIVE, ResultSet.CONCUR_UPDATABLE))
        {
            idPs.setInt(1,film_id);
            if(!idPs.execute()){
                String message = "Can't find film_id " + film_id;

                JOptionPane.showMessageDialog(frm, message,
                    "ERROR",JOptionPane.ERROR_MESSAGE);
            } else{
                rs = idPs.getResultSet();
                rs.next();

                //Creates a FilmCategory object using three-params constructo
                FilmCategory obj = new FilmCategory(cat_id, film_id, new
Timestamp(System.currentTimeMillis()));
                updatePS.setInt(1, obj.getFilmID());
                updatePS.setInt(2, cat_id_new);
                updatePS.setInt(3, obj.getFilmID());
                updatePS.setInt(4, obj.getCategoryID());

                updatePS.executeUpdate();
                rs.close();
                updatePS.close();
                idPs.close();
                conn.close();
            }
        }catch(SQLException ex){
            Logger.getLogger(FilmCategoryForm.class.getName()).log(Level.SEVE
"Error updating film-category data", ex);
            String message = "Error updating film-category data: " + ex.getMe
            String stackTrace = Arrays.toString(ex.getStackTrace());
            JOptionPane.showMessageDialog(null, message + "\n\n" + stackTrace
"ERROR", JOptionPane.ERROR_MESSAGE);
        }catch(java.lang.NumberFormatException ex){
            Logger.getLogger(FilmCategoryForm.class.getName()).log(Level.SEVE
"Invalid Input", ex);
            String message = "Invalid Input: " + ex.getMessage();
```

```
            String stackTrace = Arrays.toString(ex.getStackTrace());
            JOptionPane.showMessageDialog(null, message + "\n\n" + stackTrace
"ERROR", JOptionPane.ERROR_MESSAGE);
        }
    }

    private static HashMap<String, String> read_inputs(FilmCategoryForm frm)
        HashMap<String, String> input_data = new HashMap<>();
        String film_id = String.valueOf(frm.getJCBFilmID().getSelectedItem()
        String cat_id = frm.getJTFCategoryID().getText();
        String cat_id_new = String.valueOf(frm.getJCBCategoryID().getSelecte

        // Validate user input
        int film_id_int = 0;
        try {
            film_id_int = Integer.parseInt(film_id);
            if (film_id_int <= 0) {
                throw new IllegalArgumentException("Film ID cannot be negati
zero");
            }
        } catch (NumberFormatException ex) {
            JOptionPane.showMessageDialog(frm, "Invalid Film ID: " + film_id,
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        } catch (IllegalArgumentException ex) {
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        }

        if (cat_id == null) {
            JOptionPane.showMessageDialog(frm, "Category ID cannot be empty",
            "Error", JOptionPane.ERROR_MESSAGE);
            throw new IllegalArgumentException("Category ID cannot be empty")
        }

        if (cat_id_new == null || cat_id_new.isEmpty()) {
            JOptionPane.showMessageDialog(frm, "New Category ID cannot be emp
            "Error", JOptionPane.ERROR_MESSAGE);
            throw new IllegalArgumentException("Ner Category ID cannot be emp
        }

        input_data.put("film_id", film_id);
        input_data.put("cat_id", cat_id);
        input_data.put("cat_id_new", cat_id_new);

        return input_data;
    }

    private static void edit_actual(FilmCategoryForm frm){
        try{
            HashMap<String, String> input_data = read_inputs(frm);
            int film_id = Integer.parseInt(input_data.get("film_id"));
```

```
            int cat_id = Integer.parseInt(input_data.get("cat_id"));
            int cat_id_new = Integer.parseInt(input_data.get("cat_id_new"));

            update_row_by_film_cat_id(frm, film_id, cat_id, cat_id_new);

            //Refreshes all objects on form
            refresh_controls(frm);

        }catch(SQLException ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    private static void enable_controls(boolean state, FilmCategoryForm frm){
        frm.getJBFirst().setEnabled(state);
        frm.getJBPrev().setEnabled(state);
        frm.getJBNext().setEnabled(state);
        frm.getJBLast().setEnabled(state);
        frm.getJBInsert().setEnabled(state);
        frm.getJBDelete().setEnabled(state);
        frm.getJTFFilmID().setEnabled(state);
        frm.getJTFDescription().setEnabled(state);
        frm.getJTFReleaseYear().setEnabled(state);
        frm.getJTFLanguageName().setEnabled(state);
        frm.getJTFOriginalLanguageName().setEnabled(state);
        frm.getJTFRentalDuration().setEnabled(state);
        frm.getJTFRentalRate().setEnabled(state);
        frm.getJTFLength().setEnabled(state);
        frm.getJTFReplacementCost().setEnabled(state);
        frm.getJTFSpecialFeatures().setEnabled(state);
        frm.getJTFLastUpdate().setEnabled(state);
    }

    public static void edit_handler(FilmCategoryForm frm){
        if(frm.getJBEdit().getText().equals("EDIT")){
            frm.getJBEdit().setText("CONFIRM");

            // Disables controls
            enable_controls(false, frm);
        }

        else {
            frm.getJBEdit().setText("EDIT");

            // Actual editing
            edit_actual(frm);

            //Enables controls
            enable_controls(true, frm);
        }
    }

    //Inserts new row into film_category table
```

```
    private static void insert_row(FilmCategoryForm frm) throws SQLException{
        HashMap<String, String> input_data = read_inputs(frm);
        int film_id = Integer.parseInt(input_data.get("film_id"));
        int cat_id_new = Integer.parseInt(input_data.get("cat_id_new"));

        // SQL insert statement
        String sql = """
            INSERT INTO film_category(film_id, category_id)
                VALUES(?, ?)""";

        try(Connection conn = getConnection();
         PreparedStatement pstmt = conn.prepareStatement(sql)){

            //Creates a FilmCategory object three-params constructor
            FilmCategory obj = new FilmCategory(cat_id_new, film_id, new
Timestamp(System.currentTimeMillis()));
            pstmt.setInt(1,obj.getFilmID());
            pstmt.setInt(2,obj.getCategoryID());

            //Executes the sql insert statement
            pstmt.executeUpdate();
        } catch (SQLException ex) {
            Logger.getLogger(FilmActorForm.class.getName()).log(Level.SEVERE,
"Database error", ex);
            JOptionPane.showMessageDialog(frm, "Error: Database error\n" +
ex.getMessage());
        }
    }

    private static void insert_actual(FilmCategoryForm frm){
        try{
            insert_row(frm);

            //Refreshes table and comboboxes
            refresh_controls(frm);

        }catch(SQLException ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    public static void insert_handler(FilmCategoryForm frm){
        if(frm.getJBInsert().getText().equals("INSERT")){
            frm.getJBInsert().setText("CONFIRM");

            //Disables jbEdit
            frm.getJBEdit().setEnabled(false);

            // Disables controls
            enable_controls(false, frm);

            // Enables
```

```
            frm.getJBInsert().setEnabled(true);
        }

        else {
            frm.getJBInsert().setText("INSERT");

            // Actual insertion
            insert_actual(frm);

            //Enables jbEdit
            frm.getJBEdit().setEnabled(true);

            //Enables controls
            enable_controls(true, frm);
        }
    }

    public static void delete_handler(FilmCategoryForm frm){
        int dialogButton = JOptionPane.YES_NO_OPTION;
        int film_id =
Integer.parseInt(String.valueOf(frm.getJCBFilmID().getSelectedItem()));
        int cat_id =
Integer.parseInt(String.valueOf(frm.getJCBCategoryID().getSelectedItem()));

        String message = String.format("Are you sure you want to delete the r
ID: %d, Category ID: %d)", film_id, cat_id);
        int answer = JOptionPane.showConfirmDialog(frm, message, "DELETING RO
DATA", dialogButton);

        if(answer == JOptionPane.YES_OPTION){
            String query = """
                DELETE FROM film_category WHERE film_id = ? AND category_id =
            try(Connection conn = getConnection();
                PreparedStatement ps = conn.prepareStatement(query)){
                // Use PreparedStatement to avoid SQL injection attacks
                ps.setInt(1, film_id);
                ps.setInt(2, cat_id);
                ps.executeUpdate();

                // Refresh table and comboboxes
                refresh_controls(frm);

            } catch (SQLException ex){
                JOptionPane.showMessageDialog(frm, ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
            }
        }
    }

    private static void draw_pie_chart_film_by_category(Charts_FilmCategory f
JPanel jp){
        jp.setPreferredSize(new Dimension(jp.getWidth(), jp.getHeight()));
```

```
        DefaultPieDataset dataset =
create_pie_dataset(Query_FilmCategory.get_sql_film_category_dist(), "Number",
"category_name");

        //Draws piechart film distribution by category name
        draw_piechart_with_dataset(frm, jp, dataset, "TOP 10 FILM DISTRIBUTIO
CATEGORY NAME");
    }

    private static void
draw_bar_chart_category_by_avg_replacement_cost(Charts_FilmCategory frm, JPar
        jp.setPreferredSize(new Dimension(jp.getWidth(), jp.getHeight()));

        DefaultCategoryDataset dataset =
create_bar_dataset(Query_FilmCategory.get_sql_film_category_dist(),
"avg_replacement_cost", "category_name");

        //Draws barchart category distribution by average replacement cost
        draw_barchart_with_dataset(frm, jp, dataset, "THE CATEGORY DISTRIBUTI
AVERAGE REPLACEMENT COST", "CATEGORY NAME", "AVG REPLACEMENT COST");
    }

    private static void draw_bar_chart_category_by_avg_rental_rate(Charts_Fil
frm, JPanel jp){
        jp.setPreferredSize(new Dimension(jp.getWidth(), jp.getHeight()));

        DefaultCategoryDataset dataset =
create_bar_dataset(Query_FilmCategory.get_sql_film_category_dist(), "avg_rent
"category_name");

        //Draws barchart category distribution by average rental rate
        draw_barchart_with_dataset(frm, jp, dataset, "THE CATEGORY DISTRIBUTI
AVERAGE RENTAL RATE", "CATEGORY NAME", "AVG RENTAL RATE");
    }

    private static void draw_bar_chart_category_by_avg_length(Charts_FilmCate
JPanel jp){
        jp.setPreferredSize(new Dimension(jp.getWidth(), jp.getHeight()));

        DefaultCategoryDataset dataset =
create_bar_dataset(Query_FilmCategory.get_sql_film_category_dist(), "avg_leng
"category_name");

        //Draws barchart category distribution by average length
        draw_barchart_with_dataset(frm, jp, dataset, "THE CATEGORY DISTRIBUTI
LENGTH", "CATEGORY NAME", "AVG LENGTH");
    }

    public static void jbchart_handler(Charts_FilmCategory frm){
        //Draws piechart film distribution by category name
        draw_pie_chart_film_by_category(frm, frm.getJPanel1());

        //Draws barchart category distribution by average replacement cost
```

```
        draw_bar_chart_category_by_avg_replacement_cost(frm, frm.getJPanel2()
        //Draws barchart category distribution by average rental rate
        draw_bar_chart_category_by_avg_rental_rate(frm, frm.getJPanel3());
        //Draws barchart category distribution by average length
        draw_bar_chart_category_by_avg_length(frm, frm.getJPanel4());
    }
}
```

This is the full version of **FilmCategoryForm.java**:

```
package sakila;
import java.awt.Toolkit;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.*;
import javax.swing.event.TableModelEvent;
import javax.swing.event.TableModelListener;

public class FilmCategoryForm extends javax.swing.JFrame {

    public FilmCategoryForm() {
        initComponents();
        Utility.setLookAndFeel(this);
        FilmCategory_Utils.refresh_controls(this);

 this.setIconImage(Toolkit.getDefaultToolkit().getImage(getClass().getResourc
;
//        this.setDefaultCloseOperation(this.HIDE_ON_CLOSE);
    }

    //Getter methods for JTextField variable instances
    public JTextField getJTFFilmID(){return this.jtfFilmID;}
    public JTextField getJTFCategoryID(){return this.jtfCategoryID;}
    public JTextField getJTFDescription(){return this.jtfDescription;}
    public JTextField getJTFLastUpdate(){return this.jtfLastUpdate;}
    public JTextField getJTFLanguageName(){return this.jtfLanguageName;}
    public JTextField getJTFReleaseYear(){return this.jtfReleaseYear;}
    public JTextField getJTFRentalDuration(){return this.jtfRentalDuration;}
    public JTextField getJTFOriginalLanguageName(){return this.jtfOriginalLar
    public JTextField getJTFRentalRate(){return this.jtfRentalRate;}
    public JTextField getJTFLength(){return this.jtfLength;}
    public JTextField getJTFReplacementCost(){return this.jtfReplacementCost;
    public JTextField getJTFSpecialFeatures(){return this.jtfSpecialFeatures;
    public JTextField getJTFRating(){return this.jtfRating;}

    //Getter methods for JComboBox variable instances
    public JComboBox getJCBFilmID(){return this.jcbFilmID;}
    public JComboBox getJCBCategoryID(){return this.jcbCategoryID;}
    public JComboBox getJCBTitle(){return this.jcbTitle;}
    public JComboBox getJCBCategoryName(){return this.jcbCategoryName;}
```

```
    //Getter methods for JTable variable instance
    public JTable getJTFilmCategory(){return this.jtFilmCategory;}

    //Getter methods for JButton variable instances
    public JButton getJBEdit(){return this.jbEdit;}
    public JButton getJBInsert(){return this.jbInsert;}
    public JButton getJBDelete(){return this.jbDelete;}
    public JButton getJBChart(){return this.jbChart;}
    public JButton getJBFirst(){return this.jbFirst;}
    public JButton getJBPrev(){return this.jbPrev;}
    public JButton getJBNext(){return this.jbNext;}
    public JButton getJBLast(){return this.jbLast;}

    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">
    private void initComponents() {
        //...
        pack();
    }// </editor-fold>

    private void jbFirstActionPerformed(java.awt.event.ActionEvent evt) {
       FilmCategory_Utils.show_first_row(this);
    }

    private void jbLastActionPerformed(java.awt.event.ActionEvent evt) {
        FilmCategory_Utils.show_last_row(this);
    }

    private void jbPrevActionPerformed(java.awt.event.ActionEvent evt) {
        FilmCategory_Utils.show_prev_row(this);
    }

    private void jbNextActionPerformed(java.awt.event.ActionEvent evt) {
        FilmCategory_Utils.show_next_row(this);
    }

    private void jcbFilmIDActionPerformed(java.awt.event.ActionEvent evt) {
        FilmCategory_Utils.jcbFilm_handler(this, this.jcbFilmID);
    }

    private void jcbTitleActionPerformed(java.awt.event.ActionEvent evt) {
        FilmCategory_Utils.jcbFilm_handler(this, this.jcbTitle);
    }

    private void jtFilmCategoryMousePressed(java.awt.event.MouseEvent evt) {
        FilmCategory_Utils.mouse_pressed_handler(this);
    }

    private void jbChartActionPerformed(java.awt.event.ActionEvent evt) {
        Charts_FilmCategory frm1 = new Charts_FilmCategory();
        frm1.setLocationRelativeTo(null);
```

```
        frm1.setTitle("FOUR DISTRIBUTIONS IN FILM-CATEGORY TABLE");
        frm1.setVisible(true);
        FilmCategory_Utils.jbchart_handler(frm1);
    }

    private void jbDeleteActionPerformed(java.awt.event.ActionEvent evt) {
        FilmCategory_Utils.delete_handler(this);
    }

    private void jbInsertActionPerformed(java.awt.event.ActionEvent evt) {
        FilmCategory_Utils.insert_handler(this);
    }

    private void jbEditActionPerformed(java.awt.event.ActionEvent evt) {
        FilmCategory_Utils.edit_handler(this);
    }

    private void jcbCategoryIDActionPerformed(java.awt.event.ActionEvent evt)
        FilmCategory_Utils.jcbCategory_handler(this, this.jcbCategoryID, this
    }

    private void jcbCategoryNameActionPerformed(java.awt.event.ActionEvent ev
        FilmCategory_Utils.jcbCategory_handler(this, this.jcbCategoryName, th
    }

    private void jbFilmFormActionPerformed(java.awt.event.ActionEvent evt) {
        FilmForm frm = new FilmForm();
        frm.setVisible(true);
    }

    private void jbCategoryFormActionPerformed(java.awt.event.ActionEvent evt
        CategoryForm frm = new CategoryForm();
        frm.setVisible(true);
    }

    private void jbLanguageFormActionPerformed(java.awt.event.ActionEvent evt
        LanguageForm frm = new LanguageForm();
        frm.setVisible(true);
    }

    public static void main(String args[]) {
        try {
            for (javax.swing.UIManager.LookAndFeelInfo info :
javax.swing.UIManager.getInstalledLookAndFeels()) {
                if ("Nimbus".equals(info.getName())) {
                    javax.swing.UIManager.setLookAndFeel(info.getClassName())
                    break;
                }
            }
        } catch (ClassNotFoundException ex) {

java.util.logging.Logger.getLogger(FilmCategoryForm.class.getName()).log(java
```

```
null, ex);
        } catch (InstantiationException ex) {

java.util.logging.Logger.getLogger(FilmCategoryForm.class.getName()).log(java
null, ex);
        } catch (IllegalAccessException ex) {

java.util.logging.Logger.getLogger(FilmCategoryForm.class.getName()).log(java
null, ex);
        } catch (javax.swing.UnsupportedLookAndFeelException ex) {

java.util.logging.Logger.getLogger(FilmCategoryForm.class.getName()).log(java
null, ex);
        }

        /* Create and display the form */
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new FilmCategoryForm().setVisible(true);
            }
        });
    }

    // Variables declaration - do not modify
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel10;
    private javax.swing.JLabel jLabel11;
    private javax.swing.JLabel jLabel12;
    private javax.swing.JLabel jLabel13;
    private javax.swing.JLabel jLabel14;
    private javax.swing.JLabel jLabel15;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JLabel jLabel3;
    private javax.swing.JLabel jLabel4;
    private javax.swing.JLabel jLabel5;
    private javax.swing.JLabel jLabel6;
    private javax.swing.JLabel jLabel7;
    private javax.swing.JLabel jLabel8;
    private javax.swing.JLabel jLabel9;
    private javax.swing.JScrollPane jScrollPane1;
    private javax.swing.JButton jbCategoryForm;
    private javax.swing.JButton jbChart;
    private javax.swing.JButton jbDelete;
    private javax.swing.JButton jbEdit;
    private javax.swing.JButton jbFilmForm;
    private javax.swing.JButton jbFirst;
    private javax.swing.JButton jbInsert;
    private javax.swing.JButton jbLanguageForm;
    private javax.swing.JButton jbLast;
    private javax.swing.JButton jbNext;
    private javax.swing.JButton jbPrev;
    private javax.swing.JComboBox<String> jcbCategoryID;
    private javax.swing.JComboBox<String> jcbCategoryName;
```

```
    private javax.swing.JComboBox<String> jcbFilmID;
    private javax.swing.JComboBox<String> jcbTitle;
    private javax.swing.JTable jtFilmCategory;
    private javax.swing.JTextField jtfCategoryID;
    private javax.swing.JTextField jtfDescription;
    private javax.swing.JTextField jtfFilmID;
    private javax.swing.JTextField jtfLanguageName;
    private javax.swing.JTextField jtfLastUpdate;
    private javax.swing.JTextField jtfLength;
    private javax.swing.JTextField jtfOriginalLanguageName;
    private javax.swing.JTextField jtfRating;
    private javax.swing.JTextField jtfReleaseYear;
    private javax.swing.JTextField jtfRentalDuration;
    private javax.swing.JTextField jtfRentalRate;
    private javax.swing.JTextField jtfReplacementCost;
    private javax.swing.JTextField jtfSpecialFeatures;
    // End of variables declaration
}
```

COUNTRY FORM

COUNTRY FORM

CREATING AND POPULATING COUNTRY TABLE

CREATING AND POPULATING COUNTRY TABLE

Step 1	Create a new class named **Query_Country**. It includes several SQL queries and a SQL table creation statement related to a database schema for a movie rental company called sakila. The SQL queries are defined as private static final strings, which means that they can be accessed by other classes in the same package but cannot be modified. The queries are: • **sql_min**: Returns the minimum value of the **country_id** column from the **country** table. • **sql_max**: Returns the maximum value of the **country_id** column from the **country** table. • **sql_id**: Returns all columns from the **country** table where the **country_id** matches a given value (specified using a prepared statement). • **sql_name**: Returns all columns from the **country** table where the **country** matches a given value (specified using a prepared statement). The SQL table creation statement is also defined as a private static final string and is named **sql_country**. It creates a new table called country with three columns: **country_id**, **country**, and **last_update**. The class also includes several getter methods that return the SQL queries and table creation statement as strings. These getter methods are: • **get_sql_min()**: Returns the **sql_min** query as a string.

- **get_sql_max()**: Returns the **sql_max** query as a string.
- **get_sql_id()**: Returns the **sql_id** query as a string.
- **get_sql_country()**: Returns the **sql_country** table creation statement as a string.
- **get_sql_name()**: Returns the **sql_name** query as a string.

```
1  package sakila;
2
3  public class Query_Country {
4      private static final String
5  sql_min = "SELECT MIN(country_id)
6  FROM country";
7      private static final String
8  sql_max = "SELECT MAX(country_id)
9  FROM country";
10     private static final String sql_id =
11 "SELECT * FROM country WHERE country_id  = ?";
12     private static final String sql_name =
13 "SELECT * FROM country WHERE country = ?";
14
15
16     private static final String
17 sql_country = """
18         CREATE TABLE country (
19           country_id SMALLINT
20 UNSIGNED NOT NULL AUTO_INCREMENT,
21           country VARCHAR(50) NOT
22 NULL,
23           last_update TIMESTAMP NOT
24 NULL DEFAULT CURRENT_TIMESTAMP ON
25 UPDATE CURRENT_TIMESTAMP,
26           PRIMARY KEY  (country_id)
27         ) ENGINE=InnoDB DEFAULT
28 CHARSET=utf8mb4;""";
29
30     //Getter methods
31     public static String
32 get_sql_min() {
33         return sql_min;
34     }
35
36     public static String
37 get_sql_max() {
38         return sql_max;
     }

     public static String
 get_sql_id() {
         return sql_id;
```

```
    }

    public static String
get_sql_country() {
        return sql_country;
    }

    public static String
get_sql_name() {
        return sql_name;
    }
}
```

Step 2

Then, create a public class named **Country**. It represents a country entity in the database. It has three instance variables: **country_id** of type **int**, **country** of type **String**, and **last_update** of type **Timestamp**.

It has three constructors:

1. A default constructor that initializes the instance variables with default values.
2. A two-parameter constructor that sets the **country** and **last_update** fields.
3. A three-parameter constructor that sets the **country_id**, **country**, and **last_update** fields.

It has getter and setter methods for each instance variable to get and set the values of the variables respectively.

It overrides the **equals()** and **hashCode()** methods to compare the values of the instance variables. It also overrides the **toString()** method to return a string representation of the object.

```
1
2
3
4
5
6
```

	7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60

```
61
62
63
64
65
```

```java
package sakila;
import java.util.Objects;
import java.sql.Timestamp;

public class Country {
    private int country_id;
    private String country;
    private Timestamp last_update;

    //Default constructor
    Country(){
        this(1, "Country xxx", new
Timestamp(System.currentTimeMillis()));
    }

    //Two-params constructor
    Country(String name, Timestamp lu){
        setCountry(name);
        setLastUpdate(lu);
    }

    //Three-params constructor
    Country(int id, String name,
Timestamp lu){
        this(name, lu);
        setCountryID(id);
    }

    // Getter methods
    public int getCountryID() {return
country_id;}
    public String getCountry() {return
country;}
    public Timestamp getLastUpdate()
{return last_update;}

    //Setter methods
    public void setCountryID(int id) {
        if (id <= 0) {
            throw new
IllegalArgumentException("Country ID must be greater
than zero.");
        }
        this.country_id = id;
    }

    public void setCountry(String name)
{
        if (name == null ||
name.trim().isEmpty()) {
            throw new
IllegalArgumentException("Country cannot be null or
```

```
empty");
        }
        if (name.length() > 50) {
            throw new
IllegalArgumentException("Country
cannot be longer than 50 characters");
        }
        this.country = name;
    }

    public void setLastUpdate(Timestamp
date){
        if (date == null) {
        throw new
IllegalArgumentException("Date cannot
be null");
        }
        this.last_update = date;
    }

    // Override the hashCode() method
    @Override
    public int hashCode() {
        return Objects.hash(country_id,
country, last_update);
    }

    // Override the equals() method
    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() !=
o.getClass()) return false;
        Country ct = (Country) o;
        return country_id ==
ct.country_id &&
                Objects.equals(country,
ct.country) &&

Objects.equals(last_update,
ct.last_update);
    }

    @Override
    public String toString(){
        return "\nCountry ID   : " +
getCountryID() +
                "\nCountry      : " +
getCountry() +
                "\nLast Update  : " +
getLastUpdate();
    }
}
```

Step 3	Create a new public class na extends the **Utility** class. It con manipulating data in the **coun** database. The class extends the **Utility** **getConnection()** method to es the Sakila database. The clas constants, such as FIRST_INDE and a private static field **current** The class has three main method 1. **create_country_table()**: the **country** table in t executing a SQL qu **Query_Country** class. If this method will not creat successfully created, a displayed to inform the us 2. **populate_country_table** some sample data into creating **Country** **PreparedStatement** to Two **Country** objects are default constructor and th parameter constructor. B into the **country** table. 3. **read_country_table()**: T data from the **countr** **Country** objects using t **Country** objects are ther using the **toString()** met Overall, the **Country_Utils** functionality for creating, popu **country** table in the Sakila datab 1 `package sakila;` 2 `import java.util.logging.Log` 3 `import java.sql.*;` 4 `import java.util.ArrayList;` 5 `import java.util.Arrays;` 6 `import java.util.HashMap;` 7 `import java.util.Objects;` 8 `import javax.swing.JComboBo` 9 `import javax.swing.JOptionPa`

```
10 import javax.swing.event.Tab
11 import javax.swing.table.Def
12 import javax.swing.table.Tab
13
14 public class Country_Utils e
15     public static final int
16     public static final int
17
18     private static int curre
19 FIRST_INDEX;
20     private static final Str
21 Query_Country.get_sql_id();
22
23     //Creates country table
24     public static void creat
25         try (Connection conn
26 {
27             Statement stmt =
28 conn.createStatement();
29
30 stmt.addBatch(Query_Country.
31             stmt.executeBatc
32
33             String message =
34 String.format("Successfully
35 table");
36             JOptionPane.show
37 message,
38
39  "INFORMATION",JOptionPane.I
40 ;
41
42         } catch (SQLExceptio
43             JOptionPane.show
44 ex.getMessage(),
45
46  "ERROR",JOptionPane.ERROR_M
47         }
48     }
49
50     //Populates country tabl
51 data
52     public static void popul
53 {
54         try(Connection conn
55             String sql = """
56                 INSERT INTO
57 country, last_update)
58                 VALUES(?, ?,
59
60             //Creates a new
61 default constructor
62
63
```

```
64              PreparedStatemer
65  conn.prepareStatement(sql);
66              Country obj1 = n
67              ps1.setInt(1,obj
68              ps1.setString(2,
69
70  ps1.setTimestamp(3,obj1.getL
71
72              // Creates a new
73  three-params constructor
74              PreparedStatemer
75  conn.prepareStatement(sql);
76              Country obj2 = n
77  State", new
78  Timestamp(System.currentTime
79              ps2.setInt(1,obj
80              ps2.setString(2,
81
82  ps2.setTimestamp(3,obj2.getL
83
84              ps1.executeUpdat
85              ps2.executeUpdat
86
87          }catch(SQLException
88              JOptionPane.show
89  ex.getMessage(),
90
91   "ERROR",JOptionPane.ERROR_M
          }
      }

      //Reads the content of c
      public static void read_
          try(Connection conn
              Statement stmt =
  conn.createStatement();
              ResultSet rs =
  stmt.executeQuery("SELECT *

              while(rs.next())
                  int ct_id =
  rs.getInt("country_id");
                  String count
  rs.getString("country");
                  Timestamp lu
  rs.getTimestamp("last_update

                  //Creates a
  using three-params construct
                  Country obj
  Country(ct_id, country, lu);
                  System.out.p
              }
```

```
            rs.close();
            stmt.close();

        }catch(SQLException
            JOptionPane.show
ex.getMessage(),

 "ERROR",JOptionPane.ERROR_M
        }
    }
```

Step 4

In the driver class, **create_country_table()**, **pop** and **read_country_table()** as sh

```
1  package sakila;
2
3  public class Sakila {
4      public static void main(
5  //         Utility.testConnec
6  //         Actor_Utils.create
7  //         Actor_Utils.popula
8  //         Actor_Utils.read_a
9  //         ActorForm frm = ne
10 //         frm.setVisible(tru
11
12 //         Language_Utils.cre
13 //
14 Language_Utils.populate_lang
15 //         Language_Utils.rea
16 //         LanguageForm frm =
17 //         frm.setVisible(tru
18
19 //         Category_Utils.cre
20 //
21 Category_Utils.populate_cate
22 //         Category_Utils.rea
23 //         CategoryForm frm =
24 //         frm.setVisible(tru
25
26 //         Film_Utils.create_
27 //         Film_Utils.populat
28 //         Film_Utils.read_fi
29 //         FilmForm frm = new
30 //         frm.setVisible(tru
31
32 //
33 FilmActor_Utils.create_film_
34 //
35 FilmActor_Utils.populate_fil
36
```

```
37 //        FilmActor_Utils.re
38 //        FilmActorForm frm
39 //        frm.setVisible(tru
40
41 //
42 FilmCategory_Utils.create_fi
43 //
44 FilmCategory_Utils.populate_
45 //
46 FilmCategory_Utils.read_film
   //        FilmCategoryForm f
   FilmCategoryForm();
   //        frm.setVisible(tru

           Country_Utils.create
           Country_Utils.popula
           Country_Utils.read_c
       }
   }
```

Run project to see the result in co

```
Country ID   : 1
Country      : Country xxx
Last Update  : 2023-04-26 18:25:

Country ID   : 2
Country      : Toba State
Last Update  : 2023-04-26 18:25:
```

DESIGNING GUI
DESIGNING GUI

Step 1	In the project, create a new **JFrame Form** and name it as **CountryForm.java**. In the Design tab, add three **JLabels** to the form and set their corresponding **text** properties as COUNTRY ID, COUNTRY, and LAST UPDATE.
Step 2	Then, add three **JTextField** to the form and set their corresponding **Variable Name** as **jtfCountryID**, **jtfCountry**, and **jtfLastUpdate**.
Step 3	Then, add seven **JButton** to the form and set their corresponding **Variable Name** as **jbFirst**, **jbPrev**, **jbNext**, **jbLast**, **jbEdit**, **jbInsert**, and **jbDelete**. Set

	their corresponding **text** properties as FIRST, PREV, NEXT, LAST, EDIT, INSERT, and DELETE.
Step 4	Then, add two **JComboBoxes** to the form and set their corresponding **Variable Name** as **jcbCountryID** and **jcbCountry**.
Step 5	Lastly, add a new **JTable** to the form set set its **Variable Name** as **jtCountry**. Then, right-click on it, then choose **Table Contents...** and set the number of columns to 3 and the number of rows to 25.
Step 6	In the driver class, **Sakila.java**, create a new object of **CountryForm** class using its default constructor as shown in 45 – 46:

```
1  package sakila;
2
3  public class Sakila {
4      public static void main(String[] args) {
5  //         Utility.testConnection();
6  //         Actor_Utils.create_actor_table();
7  //         Actor_Utils.populate_actor_table();
8  //         Actor_Utils.read_actor_table();
9  //         ActorForm frm = new ActorForm();
10 //         frm.setVisible(true);
11
12 //         Language_Utils.create_language_table();
13 //
14 Language_Utils.populate_language_table();
15 //         Language_Utils.read_language_table();
16 //         LanguageForm frm = new LanguageForm();
17 //         frm.setVisible(true);
18
19 //         Category_Utils.create_category_table();
20 //
21 Category_Utils.populate_category_table();
22 //         Category_Utils.read_category_table();
23 //         CategoryForm frm = new CategoryForm();
24 //         frm.setVisible(true);
25
26 //         Film_Utils.create_film_table();
27 //         Film_Utils.populate_film_table();
28 //         Film_Utils.read_film_table();
29 //         FilmForm frm = new FilmForm();
30 //         frm.setVisible(true);
31
32 //
33 FilmActor_Utils.create_film_actor_table();
34
```

```
35  //
36  FilmActor_Utils.populate_film_actor_table();
37  //        FilmActor_Utils.read_film_actor_table();
38  //        FilmActorForm frm = new FilmActorForm();
39  //        frm.setVisible(true);
40
41  //
42  FilmCategory_Utils.create_film_category_table();
43  //
44  FilmCategory_Utils.populate_film_category_table();
45  //
46  FilmCategory_Utils.read_film_category_table();
47  //        FilmCategoryForm frm = new
48  FilmCategoryForm();
    //        frm.setVisible(true);

    //        Country_Utils.create_country_table();
    //        Country_Utils.populate_country_table();
    //        Country_Utils.read_country_table();
            CountryForm frm = new CountryForm();
            frm.setVisible(true);
        }
    }
```

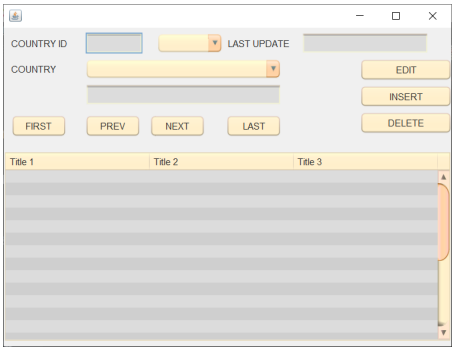


Figure 8.1 The layout of country form

Step 8

In **CountryForm**'s constructor, invoke **setLookAndFeel()** to set the look and feel of the form as shown in line 17.

```
1  package sakila;
2
3  import java.awt.Toolkit;
4  import java.awt.event.ActionEvent;
5  import
6  java.awt.event.ActionListener;
7  import javax.swing.JButton;
8  import javax.swing.JComboBox;
9  import javax.swing.JMenuItem;
```

```
import javax.swing.JOptionPane;
import javax.swing.JPopupMenu;
import javax.swing.JTable;
import javax.swing.JTextField;

public class CountryForm extends
javax.swing.JFrame {
    public CountryForm() {
        initComponents();

Utility.setLookAndFeel(this);
    }
    //...
}
```

Run the project to see the country form as shown in Figure 8.1.

Step 9

In **CountryForm.java**, define **getter()** method for every object in the form and place them outside and beneath its default constructor:

```
    //Getter method for
jtfCountryID
    public JTextField
getJTFCountryID(){
        return this.jtfCountryID;
    }

    //Getter method for
jtfLastUpdate
    public JTextField
getJTFLastUpdate(){
        return this.jtfLastUpdate;
    }

    //Getter method for jtfCountry
    public JTextField
getJTFCountry(){
        return this.jtfCountry;
    }

    //Getter method for jtCountry
    public JTable getJTCountry(){
        return this.jtCountry;
    }

    //Getter method for
jcbCountryID
    public JComboBox
getJCBCountryID(){
```

```
30          return this.jcbCountryID;
31      }
32
33      //Getter method for jcbCountry
34      public JComboBox
35  getJCBCountry(){
36          return this.jcbCountry;
37      }
38
39      //Getter method for jbEdit
40      public JButton getJBEdit(){
41          return this.jbEdit;
42      }
43
44      //Getter method for jbInsert
45      public JButton getJBInsert(){
46          return this.jbInsert;
47      }
48
49      //Getter method for jbDelete
50      public JButton getJBDelete(){
51          return this.jbDelete;
52      }
53
54      //Getter method for jbFirst
55      public JButton getJBFirst(){
56          return this.jbFirst;
57      }
58
59      //Getter method for jbPrev
60      public JButton getJBPrev(){
61          return this.jbPrev;
62      }
63
64      //Getter method for jbNext
        public JButton getJBNext(){
            return this.jbNext;
        }

        //Getter method for jbLast
        public JButton getJBLast(){
            return this.jbLast;
        }
```

POPULATING TABLE AND COMBOBOXES

POPULATING TABLE AND COMBOBOXES

Step 1

In **Country_Utils.java**, add two new methods: **get_country_list()** and **sh** methods used to populate a JTable in the **CountryForm** class with data fr

get_country_list() takes in a **CountryForm** object, a SQL query strin **ArrayList** of **Country** objects. It establishes a connection to the data **PreparedStatement**, and then loops through the **ResultSet** and creates result set. The **Country** objects are added to the **ArrayList** and then the m

show_table_country() takes in a **CountryForm** object and an **ArrayI DefaultTableModel** with three columns for country ID, country name, a column headers using the **set_column_header()** method of the **CountryF JTable** in the **CountryForm** object to the new model. It then loops throu and adds a row to the model for each object, using the **Object[]** array row

```
1     private static ArrayList<Country>
2  get_country_list(CountryForm frm, String sql, String item){
3          ArrayList<Country> list = new ArrayList<>();
4          Connection conv = null;
5
6          try(Connection conn = getConnection();
7              PreparedStatement ps = conn.prepareStatement(sql)){
8              if (item.equalsIgnoreCase("none")==false) {
9                   ps.setString(1,item);
10             }
11             ResultSet rs = ps.executeQuery();
12
13             Country obj;
14             while(rs.next()){
15                 //Using three-params constructor
16                 obj = new Country(rs.getInt("country_id"),
17                     rs.getString("country"),
18                     rs.getTimestamp("last_update"));
19
20                 list.add(obj);
21             }
22         }catch (SQLException ex){
23             JOptionPane.showMessageDialog(frm, ex.getMessage(),
24                 "ERROR",JOptionPane.ERROR_MESSAGE);
25         }
26         return list;
27     }
28
29     private static void show_table_country(CountryForm frm,
30 ArrayList<Country> list) throws SQLException{
31         DefaultTableModel model = new DefaultTableModel(0,0);
32
33         String header[] = {"Country ID", "Country", "Last
34 Update"};
35
36
37 model.setColumnIdentifiers(set_column_header(frm.getJTCountry(),
```

```
header));
        frm.getJTCountry().setModel(model);

        Object[] row = new Object[3];

        for(int i=0; i<list.size(); i++){
            row[0] = list.get(i).getCountryID();
            row[1] = list.get(i).getCountry();
            row[2] = list.get(i).getLastUpdate();

            model.addRow(row);
        }
    }
```

Step 2

In **Country_Utils.java**, define **refresh_controls()** method. It is respon controls in the **CountryForm** object. Here's what this method does:

1. Sets the location and title of the **CountryForm** object.
2. Calls the **get_country_list()** method to retrieve the content of the store it in an **ArrayList** of **Country** objects.
3. Calls the **show_table_country()** method to populate the **JTal ArrayList** of **Country** objects.
4. Calls the **populate_combobox()** method to populate the **JCBCo** controls with the list of country IDs and country names, respectivel
5. If an SQL exception occurs while executing any of these tasks, an e

```
    public static void
refresh_controls(CountryForm frm){

frm.setLocationRelativeTo(null);
        frm.setTitle("COUNTRY
FORM");

        //Shows the content of
country table and populates
combobox
        try{
            //Makes alternating
color for table rows

table_renderer(frm.getJTCountry());

            //Populates table
            ArrayList<Country> list
= get_country_list(frm, "SELECT *
FROM country", "none");
            show_table_country(frm,
list);
```

```
            //Populates
jcbCountryID
            String sql_id = "SELECT
country_id FROM country ORDER BY
country_id";

populate_combobox(sql_id,
frm.getJCBCountryID(), frm);



            //Populates
getJCBCountry()
            String sql_ct = "SELECT
DISTINCT country FROM country ORDER
BY country";



populate_combobox(sql_ct,
frm.getJCBCountry(), frm);

        }catch (SQLException ex){

JOptionPane.showMessageDialog(frm,
ex.getMessage(),

 "ERROR",JOptionPane.ERROR_MESSAGE)
;
        }
    }
```

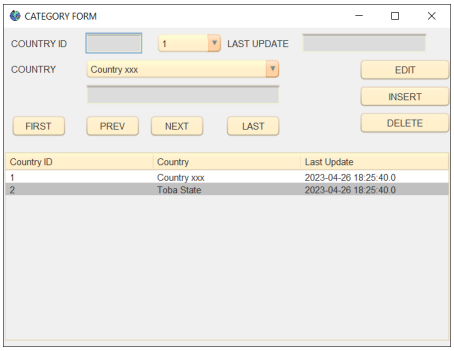


Figure 8.2 The content of **country** table displaye

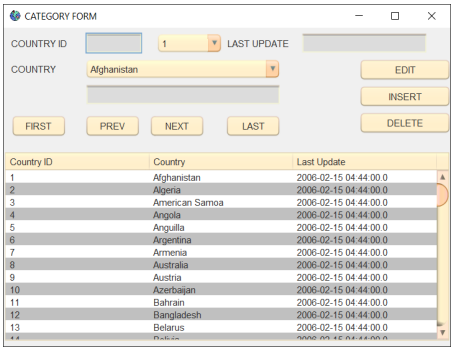


Figure 8.3 The the content of **country** table in original **Sakila** database

jtCountry

Step 3

In **CountryForm**'s default constructor, the **Country_Utils.refresh_c** populates the controls in the **CountryForm** with data from a database from the **Country_Utils** class.

The **this.setIconImage()** method sets the icon of the **C** **this.setDefaultCloseOperation(this.HIDE_ON_CLOSE)** method sets th to hide the form instead of exiting the application when the close button is

```
1  package sakila;
2  import java.awt.Toolkit;
3  import java.awt.event.ActionEvent;
4  import java.awt.event.ActionListener;
5  import java.io.IOException;
6  import java.text.ParseException;
7  import java.util.logging.Level;
8  import java.util.logging.Logger;
9  import javax.swing.JButton;
10 import javax.swing.JComboBox;
11 import javax.swing.JLabel;
12 import javax.swing.JTable;
13 import javax.swing.JTextField;
14
15 public class CountryForm extends javax.swing.JFrame {
16     public CountryForm() {
17         initComponents();
18         Utility.setLookAndFeel(this);
19         Country_Utils.refresh_controls(this);
20
21  this.setIconImage(Toolkit.getDefaultToolkit().getImage(getClass().
22 ;
23         this.setDefaultCloseOperation(this.HIDE_ON_CLOSE);
24     }
25     //...
26 }
```

Step 4	Run the project to see the content of **country** table displayed in **jtCountry** If you use the data from **Sakila** MySQL database available in the inte **country** table displayed in **jtCountry** as shown in Figure 8.3.

DISPLAYING AND NAVIGATING DATA ROW BY ROW

DISPLAYING AND NAVIGATING DATA ROW BY ROW

Step 1

In **Country_Utils** class, define two new methods named **clear_co** **display_country_data()**. These are two Java methods used in conjuncti other to display country data in the form.

The **clear_controls()** method is used to clear the form fields in the form l text of the text fields to an empty string. The method takes a reference to as a parameter and is called when no rows are found in the **Resu** **display_country_data()** method.

The **display_country_data()** method is used to display country data rov GUI form. The method takes three parameters: a reference to the GUI f query string, and a generic item. The method establishes a database prepares a **PreparedStatement** object with the SQL query string, and ex the item parameter. The resulting **ResultSet** is then looped through to ex data for each row.

If no rows are found in the **ResultSet**, the **clear_controls()** method is c the form fields. Otherwise, the extracted data is displayed in the form fi **find_combo_value_selected()** method is called twice to determine the s in two **JComboBoxes** based on the extracted country ID and name.

If an **SQLException** occurs, the method displays an error messa **JOptionPane** dialog.

```
1     private static void clear_controls(CountryForm frm){
2         frm.getJTFCountryID().setText("");
3         frm.getJTFCountry().setText("");
4         frm.getJTFLastUpdate().setText("");
5     }
6
7    //Displays country data result row by row
8     private static <T> void display_country_data(CountryForm frm, S
9 item){
10        try(Connection conn = getConnection()){
11            PreparedStatement ps = conn.prepareStatement(sql);
```

```
            ps.setObject(1,item);
            ResultSet rs = ps.executeQuery();

            if (!rs.next()) {
                // no row found, clear the form fields
                clear_controls(frm);
                return;
            }

            do{

frm.getJTFCountryID().setText(String.valueOf(rs.getInt("country_id"
                frm.getJTFCountry().setText(rs.getString("country")

frm.getJTFLastUpdate().setText(String.valueOf(rs.getTimestamp("last

                // Determines item selected from jcbCountryID
                find_combo_value_selected(frm.getJCBCountryID(),
rs.getInt("country_id"));

                // Determines item selected from jcbCountry
                find_combo_value_selected(frm.getJCBCountry(),
rs.getString("country"));
            }while(rs.next());

            rs.close();
            ps.close();
```

```
        }catch(SQLException ex){

JOptionPane.showMessageDialog(frm,
ex.getMessage(),

 "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }
```

Step 2

In the same class, define another r
handles the selection event of two **J**
jcbCountry. The method takes two
and the **JComboBox** that triggered

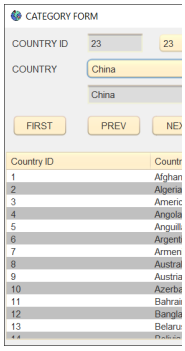
The method first gets the selecte
getSelectedItem() method. It then
event by comparing the reference c
the two **JComboBoxes** in the GUI fc

Based on which **JComboBox** trigger
query string to retrieve country
Query_Country class. The query str

Finally, the method calls the **displa**
country data in the GUI form by p
query string, and the selected item a

```
1   public static void
2  jcbCountry_handler(CountryForm
3  JComboBox<String> jcb) {
4          Object item =
5  jcb.getSelectedItem();
6          String sql = "";
7          if
8  (jcb.equals(frm.getJCBCountryI
9  {
10             sql =
11 Query_Country.get_sql_id();
12         } else if
   (jcb.equals(frm.getJCBCountry(
               sql =
   Query_Country.get_sql_name();
           }

           display_country_data(f
   sql, item);
       }
```

Step 3	In **CountryForm**, double click comboboxes and define their corresp
	
	1 private void jcbCountryIDActionPerformed(java
	2 evt) {
	3 Country_Utils.jcbCountry
	4 this.jcbCountryID);
	5 }
	6
	7 private void jcbCountryActionPerformed(java.a
	evt) {
	Country_Utils.jcbCountry
	this.jcbCountry);
	}
	
	These are event handler meth
	JComboBoxes in GUI form: **jcbCo**
	are private and take a single p
	represents the event.
	
	When an event occurs, each metho
	method of the **Country_Utils** clas
	country data in the GUI form. The
	parameters: a reference to the GUI
	the event. The reference to the GUI
	which refers to the current instance

Step 4	Run the project. Choose one of ite comboboxes to see row by row th Figure 8.4.  **Figure 8.4** Displaying row b
Step 5	Define four navigating methods ir

methods in the **Country_Utils** clas
rows of **country** table and display th

The **show_first_row()** method displa
first item in the **jcbCountryID** JCo
the **display_country_data()** metho
the value of the FIRST_INDEX const

The **show_last_row()** method displa
last item in the **jcbCountryID** JCo
the **display_country_data()** metho
the index of the last item in the **jcbC**

The **show_prev_row()** method dis
decrementing the **currentIndex** v
index in the **jcbCountryID JComb**
FIRST_INDEX constant, the method

The **show_next_row()** method di
incrementing the **currentIndex** v
index in the **jcbCountryID** JCombo
index of the last item in the **jcbCo**
without displaying any data.

```
1     public static void show_fi
2         String item =
3 String.valueOf(frm.getJCBCountr
4         display_country_data(fr
5         currentIndex = FIRST_IN
6     }
7
8     public static void show_las
9         int endIndex = frm.getJ
10 1;
11        String item =
12 String.valueOf(frm.getJCBCountr
13        display_country_data(fr
14        currentIndex = endIndex
15    }
16
17    public static void show_pre
18        currentIndex--;
19        if(currentIndex < FIRST
20            currentIndex = FIRS
21            return;
22        }
23        String item =
24 String.valueOf(frm.getJCBCountr
25        display_country_data(fr
26
```

```
27      }
28
29      public static void show_ne
30          int endIndex = frm.get
31  1;
32          currentIndex++;
33          if(currentIndex > endI
34              currentIndex = end
35              return;
36          }
            String item =
    String.valueOf(frm.getJCBCount
            display_country_data(f
        }
```

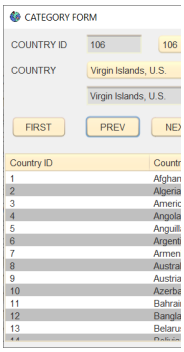
Step 6

Then in **CountryForm**, double click corresponding event handler:

```
1       private void
2   jbLastActionPerformed(java.awt.
3   evt) {
4           Country_Utils.show_last
5       }
6
7       private void
8   jbNextActionPerformed(java.awt.
9   evt) {
10          Country_Utils.show_next
11      }
12
13      private void
14  jbPrevActionPerformed(java.awt.
15  evt) {
            Country_Utils.show_prev
        }

        private void
    jbFirstActionPerformed(java.awt
    evt) {
            Country_Utils.show_firs
        }
```

These methods are event handlers corresponding methods in the **Cou** previous, or next row of country data

- **jbLastActionPerformed()** display the last row of count

	• **jbNextActionPerformed()** display the next row of count • **jbPrevActionPerformed()** display the previous row of c • **jbFirstActionPerformed()** display the first row of count Overall, these methods allow the use data using buttons in the GUI.
Step 7	Run the project. Click on one or mo shown in Figure 8.5.  **Figure 8.5** User clicks on one or r
Step 8	Define **mouse_pressed_handler()** method handles the mouse press ev first checks whether a row has been informing the user to select a row. value of the selected row from corresponding country data in the fo block to create a new database conn may be thrown. If an **SQLExcept** message and stack trace and displ user.

```
1     public static void mouse_pr
2         Objects.requireNonNull(
3
4         int selectedIndex = frm
5         if (selectedIndex == -1
6             JOptionPane.showMessa
7 data.",
8                 "No row selecte
9             return;
10        }
11
12        try (Connection conn =
```

```
13              String id =
14 String.valueOf(frm.getJTCountry
15 0));
16
17              // Displays country
18              display_country_dat
19
20          } catch (SQLException e
21
22 Logger.getLogger(CountryForm.cl
23 displaying country data", ex);
24              String message = "E
25 ex.getMessage();
26              String stackTrace =
27              JOptionPane.showMes
   stackTrace, "ERROR", JOptionPan
          }


      }
```

Step 9

Right click on **jtCountry**. Then, cho
Define its event handler:

```
1     private void
2  jtCountryMousePressed(java.awt.e
3  evt) {

   Country_Utils.mouse_pressed_han
      }
```

Step 10

Run the project. Double click on any
corresponding row in **country** table
shown in Figure 8.6.

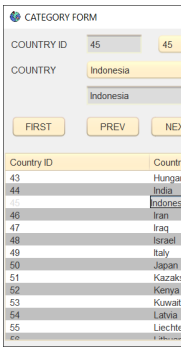


Figure 8.6 User double-

UPDATING RECORD
UPDATING RECORD

Step 1

In **Country_Utils** class, define a new method named **update_row_by_country_id()**. This method updates a row of data in the **country** table by the country ID. It takes a **CountryForm** object, an int **country_id**, and a String **country** as parameters.

First, it creates a connection to the database and prepares two SQL statements: one to check if the country ID exists in the database, and another to update the row with the new country name.

Then, it executes the **query_id** statement with the given **country_id** parameter to check if the country ID exists in the database. If it does not exist, it displays an error message using **JOptionPane.showMessageDialog()** and returns.

If the country ID exists, it creates a **Country** object using the three-parameter constructor and sets its country name to the given country parameter.

Then, it executes the **update_query** statement with the **Country** object's country name and country ID as parameters to update the row in the **country** table.

Finally, it closes the **ResultSet**, **PreparedStatement**, and **Connection** objects in a try-with-resources block.

```
1   //Updates row of data in country tabel by country_id
2   public static void update_row_by_country_id(CountryForm
3   frm, int country_id, String country) throws SQLException{
4           Connection conn = getConnection();
5           ResultSet rs = null;
6           String query_id = "SELECT country_id FROM country WHERE
7   country_id = ?";
8           String update_query = """
9                   UPDATE country SET country = ? WHERE country_id =
10  ?""";
11          try(PreparedStatement idPs =
12  conn.prepareStatement(query_id,
13
14  ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_READ_ONLY);
15          PreparedStatement updatePS =
16  conn.prepareStatement(update_query,
17                  ResultSet.TYPE_SCROLL_SENSITIVE,
18  ResultSet.CONCUR_UPDATABLE))
```

```
19          {
20              idPs.setInt(1,country_id);
21              if(!idPs.execute()){
22                  String message = "Can't find country_id " +
23 country_id;
24
25                  JOptionPane.showMessageDialog(frm, message,
26                      "ERROR",JOptionPane.ERROR_MESSAGE);
27              } else{
28                  rs = idPs.getResultSet();
29                  rs.next();
30
31                  //Creates a Country object using three-params
32 constructor
33                  Country obj = new Country(country_id, country,
34 new Timestamp(System.currentTimeMillis()));
35                  updatePS.setString(1, obj.getCountry());
36                  updatePS.setInt(2, obj.getCountryID());
37
38                  updatePS.executeUpdate();
39                  rs.close();
40                  updatePS.close();
41                  idPs.close();
42                  conn.close();
43              }
44          }catch(SQLException ex){
45
46 Logger.getLogger(CountryForm.class.getName()).log(Level.SEVERE,
47 "Error updating country data", ex);
48              String message = "Error updating country data: " +
49 ex.getMessage();
50              String stackTrace =
55 Arrays.toString(ex.getStackTrace());
              JOptionPane.showMessageDialog(null, message +
"\n\n" + stackTrace, "ERROR", JOptionPane.ERROR_MESSAGE);
          }catch(java.lang.NumberFormatException ex){

Logger.getLogger(CountryForm.class.getName()).log(Level.SEVERE,
"Invalid Input", ex);
              String message = "Invalid Input: " +
ex.getMessage();
              String stackTrace =
Arrays.toString(ex.getStackTrace());
              JOptionPane.showMessageDialog(null, message +
"\n\n" + stackTrace, "ERROR", JOptionPane.ERROR_MESSAGE);
          }
      }
```

Step 2	Then in the same class, define a new method **read_inputs()**. This method reads input data from the **CountryForm** and validates it

before returning a **HashMap** containing the input data.

First, it initializes an empty **HashMap** object to store the input data. It then reads the selected country ID from a **JComboBox** and the country name from a **JTextField**.

Next, it attempts to parse the country ID as an integer and validates that it is greater than zero. If it is not a valid integer or is less than or equal to zero, it throws a **NumberFormatException** or **IllegalArgumentException** and displays an error message using **JOptionPane**.

Finally, it checks that the country name is not null or empty. If it is null or empty, it throws an **IllegalArgumentException** and displays an error message using **JOptionPane**.

```
    private static HashMap<String, String>
read_inputs(CountryForm frm) {
        HashMap<String, String> input_data = new
HashMap<>();
        String country_id =
String.valueOf(frm.getJCBCountryID().getSelectedItem());
        String country = frm.getJTFCountry().getText();

        // Validate user input
        int country_id_int = 0;
        try {
            country_id_int =
Integer.parseInt(country_id);
            if (country_id_int <= 0) {
                throw new
IllegalArgumentException("Country ID cannot be negative
or zero");
            }
        } catch (NumberFormatException ex) {
            JOptionPane.showMessageDialog(frm, "Invalid
Country ID: " + country_id,
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        } catch (IllegalArgumentException ex) {
            JOptionPane.showMessageDialog(frm,
ex.getMessage(),
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        }

        if (country == null || country.isEmpty()) {
            JOptionPane.showMessageDialog(frm, "Country
cannot be empty",
            "Error", JOptionPane.ERROR_MESSAGE);

```

```
            throw new IllegalArgumentException("Country cannot be empty");
        }

        input_data.put("country_id", country_id);
        input_data.put("country", country);

        return input_data;
    }
```

Step 3

Still in the same class, define another method named **edit_actual()**. It reads the input data from the form using the **read_inputs()** method and then attempts to update the corresponding row in the country table using the **update_row_by_country_id()** method. If an SQL exception is thrown, it displays an error message to the user using a **JOptionPane**. If the update is successful, it calls the **refresh_controls()** method to refresh all objects on the form.

```
    private static void edit_actual(CountryForm frm){
        try{
            HashMap<String, String> input_data = read_inputs(frm);
            int country_id = Integer.parseInt(input_data.get("country_id"));
            String country = input_data.get("country");

            update_row_by_country_id(frm, country_id, country);

            //Refreshes all objects on form
            refresh_controls(frm);

        }catch(SQLException ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(),

 "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }
```

Step 4

Lastly, define two new methods named **enable_controls()** and **edit_handler()**. It is responsible for handling the edit button click event on the **CountryForm**. If the button text is "EDIT", it changes the button text to "CONFIRM" and disables the form controls. If the button text is "CONFIRM", it changes the button text back to "EDIT", calls

edit_actual() method to perform the actual editing of the selected row, and enables the form controls.

enable_controls() method is used to enable or disable form controls based on the given state parameter.

```
    private static void
enable_controls(boolean state, CountryForm
frm){
        frm.getJBFirst().setEnabled(state);
        frm.getJBPrev().setEnabled(state);
        frm.getJBNext().setEnabled(state);
        frm.getJBLast().setEnabled(state);
        frm.getJBInsert().setEnabled(state);
        frm.getJBDelete().setEnabled(state);

frm.getJTFCountryID().setEnabled(state);
    }

    public static void
edit_handler(CountryForm frm){

 if(frm.getJBEdit().getText().equals("EDIT")
){

frm.getJBEdit().setText("CONFIRM");

            // Disables controls
            enable_controls(false, frm);
        }

        else {
            frm.getJBEdit().setText("EDIT");

            // Actual editing
            edit_actual(frm);

            //Enables controls
            enable_controls(true, frm);
        }
    }
```

Step 5

Run the project. Choose **country_id** using **jcbCountryID** or **jcbCountry** combobox. Or, you can choose one of rows in **jtCountry** (in this case, **country_id = 45**). Then, click on EDIT button as shown in Figure 8.7.

Edit the country name. Then, click on CONFIRM button. The edited row had been saved into **country** table as shown in Figure 8.8.

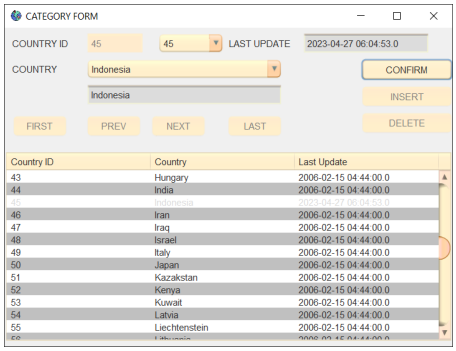


Figure 8.7 The country form is in editing state

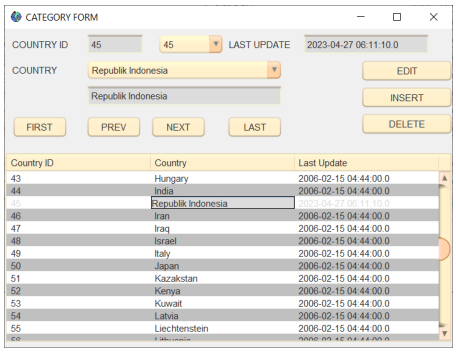


Figure 8.8 The edited row had been saved into database

UPDATING RECORD DIRECTLY ON JTABLE
UPDATING RECORD DIRECTLY ON JTABLE

Step 1

In **Country_Utility** class, define a new method **edit_database_from_jtable()**. This method is used to handle updates database when a user modifies a cell in the **JTable**. It first checks if the type is an update, and then gets the row and column index of the updated then retrieves the country ID and name from the corresponding cells JTable model.

After that, it tries to update the database by calling **update_row_by_country_id**() method with the retrieved country ID and If the update fails due to a duplicate entry, invalid SQL syntax, or a dat error, an appropriate error message is displayed to the user.

Finally, the method refreshes all objects on the form by callin **refresh_controls()** method.

```
1    public static void
```

```
edit_database_from_jtable(TableModelEvent e, CountryForm frm){
        if (e.getType() == TableModelEvent.UPDATE) {
            int row = e.getFirstRow();
            TableModel model = (TableModel)e.getSource();
            int country_id =
Integer.parseInt(String.valueOf(model.getValueAt(row, 0)));
            String name = (String) model.getValueAt(row, 1);

            try{
                update_row_by_country_id(frm, country_id,
name);

                //Refreshes all objects on form
                refresh_controls(frm);

            } catch (SQLIntegrityConstraintViolationException
ex) {

Logger.getLogger(CountryForm.class.getName()).log(Level.SEVERE,
"Duplicate entry", ex);
                JOptionPane.showMessageDialog(frm, "Error:
Duplicate entry\n" + ex.getMessage());
            } catch (SQLSyntaxErrorException ex) {

Logger.getLogger(CountryForm.class.getName()).log(Level.SEVERE,
"Invalid SQL syntax", ex);
                JOptionPane.showMessageDialog(frm, "Error:
Invalid SQL syntax\n" + ex.getMessage());
            } catch (SQLException ex) {

Logger.getLogger(CountryForm.class.getName()).log(Level.SEVERE,
"Database error", ex);
                JOptionPane.showMessageDialog(frm, "Error:
Database error\n" + ex.getMessage());
            }
        }
    }
```

Step 2

Create a new public class named **CountryTableModelListener**.It imple the **TableModelListener** interface. It listens for changes to a **TableMod** handles them by calling the **edit_database_from_jtable()** method **Country_Utils** class.

The constructor takes a **JTable** and a **CountryForm** object as arguments, are used to update the database and stop cell editing, respectively.

The **tableChanged()** method is the implementation of **TableModelListener** interface. When a change is detected in the **TableM** this method is called. It first calls the **edit_database_from_jtable()** met

update the database with the changes made in the **JTable**. It then che there is a cell editor active and stops it to ensure that the change committed before the user moves away from the edited cell.

```
1  package sakila;
2  import javax.swing.event.TableModelEvent;
3  import
4  javax.swing.event.TableModelListener;
5  import javax.swing.JTable;
6
7  public class CountryTableModelListener
8  implements TableModelListener {
9      private final JTable jt;
10     private final CountryForm frm;
11
12     public
13 CountryTableModelListener(JTable jt,
14 CountryForm frm) {
15         this.jt = jt;
16         this.frm = frm;
17     }
18
19     @Override
20     public void
21 tableChanged(TableModelEvent e) {
22
23 Country_Utils.edit_database_from_jtable(e,
   frm);

         if (jt.getCellEditor() != null) {

jt.getCellEditor().stopCellEditing();
         }
     }
}
```

Step 3

Right click on **jtCountry**. Then, choose **Events > Mouse > mouseCli** Define its event handler:

```
1      private void jtCountryMouseClicked(java.awt.event.MouseEvent evt
2          // instantiate CountryTableModelListener and add it as a lis
3 to the table model
4          CountryTableModelListener tableModelListener = new
5 CountryTableModelListener(this.getJTCountry(), this);
6
7  this.getJTCountry().getModel().addTableModelListener(tableModelList
   ;
      }
```

	When the user clicks on the JTable **jtCountry**, a **CountryTableModelListener** is instantiated and added as a listener table model. This will enable the **Country_Utils.edit_database_from_jt** method to be called whenever there is a change to the table's data. The **edit_database_from_jtable()** method updates the database based changes made by the user and then refreshes all objects on the form to the updated data. Additionally, it catches any database-related errors tha occur during the update process and displays an appropriate error mess the user. After the **edit_database_from_jtable()** method is called, **stopCellEditing()** method is called on any active cell editor for the table ensures that any changes made by the user are committed before the li updates the database.
Step 4	Run the project. Click on any cell in second column in any row in **jtCountr** you want to edit. Then, change it. Then, click any where outsid corresponding cell. The edited data had been saved into database.

INSERTING NEW RECORD
INSERTING NEW RECORD

Step 1	In **Country_Utils** class, define a method named **insert_row()**. This method inserts a new row into the **country** table in the database using the INSERT INTO SQL statement. The input data is obtained by calling the **read_inputs()** method, which returns a **HashMap** object containing the country name entered by the user. The SQL statement is prepared using a **PreparedStatement** object, which is created with the **getConnection()** method that establishes a connection to the database. The **setString()** method of the **PreparedStatement** object is then used to set the first parameter of the SQL statement with the country name obtained from the **Country** object. If the SQL statement is executed successfully, a new row is added to the **country** table with the country name entered by the user and the current timestamp. If an exception occurs during the execution of the SQL statement, an error message is displayed in a dialog box. 1 2 `//Inserts new row into country table`

```
    private static void insert_row(CountryForm frm) throws
SQLException{
        HashMap<String, String> input_data = read_inputs(frm);
        String country = input_data.get("country");

        // SQL insert statement
        String sql = """
            INSERT INTO country(country) VALUES(?)""";

        try(Connection conn = getConnection();
         PreparedStatement pstmt = conn.prepareStatement(sql)){

            //Creates a Country object two-params constructor
            Country obj = new Country(country, new
Timestamp(System.currentTimeMillis()));
            pstmt.setString(1,obj.getCountry());

            //Executes the sql insert statement
            pstmt.executeUpdate();
        } catch (SQLException ex) {

Logger.getLogger(CountryForm.class.getName()).log(Level.SEVERE,
"Database error", ex);
            JOptionPane.showMessageDialog(frm, "Error: Database
error\n" + ex.getMessage());
        }
    }
```

Step 2

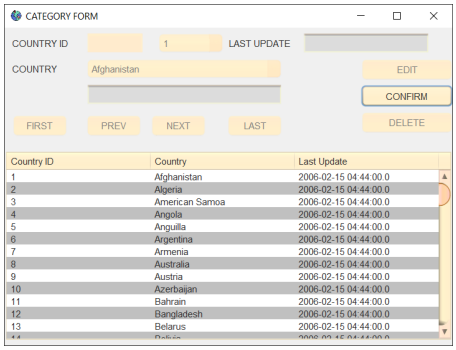
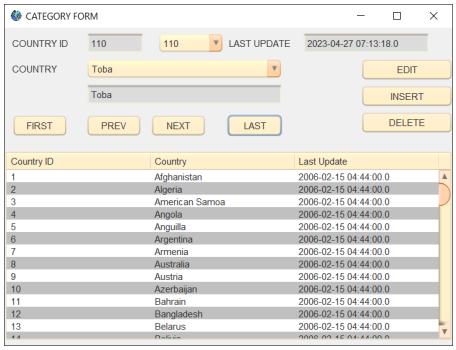
Still in **Country_Utils.java**, define **insert_actual()** and **insert_handler()** methods. The **insert_handler()** method handles the "INSERT" button's functionality in the **CountryForm**. If the button text is "INSERT", it changes the text to "CONFIRM", disables the **jbEdit** button, disables all controls except for the **JBInsert** button, and enables the **jbInsert** button. It then clears all controls, so the user can input new data.

If the button text is "CONFIRM", it calls the **insert_actual()** method, which inserts the data into the **country** table and refreshes the controls on the form. The method then enables the **jbEdit** button and enables all controls.

```
    private static void
insert_actual(CountryForm frm){
        try{
            insert_row(frm);

            //Refreshes table and comboboxes
            refresh_controls(frm);

```

```
        }catch(SQLException ex){
            JOptionPane.showMessageDialog(frm,
ex.getMessage(),

 "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    public static void
insert_handler(CountryForm frm){

 if(frm.getJBInsert().getText().equals("INSERT")
){

frm.getJBInsert().setText("CONFIRM");

            //Disables jbEdit
            frm.getJBEdit().setEnabled(false);

            // Disables controls
            enable_controls(false, frm);

frm.getJCBCountryID().setEnabled(false);

frm.getJCBCountry().setEnabled(false);

            // Clears controls
            clear_controls(frm);

            // Enables
            frm.getJBInsert().setEnabled(true);
        }

        else {
            frm.getJBInsert().setText("INSERT");

            // Actual insertion
            insert_actual(frm);

            //Enables jbEdit
            frm.getJBEdit().setEnabled(true);

            //Enables controls
            enable_controls(true, frm);

frm.getJCBCountryID().setEnabled(true);

frm.getJCBCountry().setEnabled(true);
        }
    }
```

Step 3	In **CountryForm.java**, double click on INSERT button to create its event listener:
	
	`private void jbInsertActionPerformed(java.awt.event.ActionEvent evt) {`
	`    Country_Utils.insert_handler(this);`
	`}`

Step 4	Run the project. Click on INSERT button. You will see the state of country form when insertion is in progress as shown in Figure 8.9. Then, type a country name. Then, click CONFIRM button to save the new record into **country** table as shown in Figure 8.10.  **Figure 8.9** When user clicks on INSERT button, the country form will be in state of insertion  **Figure 8.10** The new data had been saved into **country** table

DELETING RECORD

DELETING RECORD

Step 1

Then in **Country_Utils** class, define **delete_handler()** method. It **CountryForm** object as a parameter and handles the deletion of a row f **country** table. It first prompts the user with a confirmation dialog to ma they want to delete the row. If the user clicks "Yes", the method construct DELETE statement using a prepared statement to avoid SQL injection atta **country_id** of the row to be deleted is obtained from the **jcbCountryII** box of the **CountryForm** object. The **PreparedStatement** object is use the value of the **country_id** parameter in the SQL statement. The state then executed to delete the row from the database. If an exception occur the execution of the SQL statement, an error message dialog is displaye user. Finally, the **refresh_controls()** method is called to update the ta combo boxes with the changes made to the database.

```
    public static void delete_handler(CountryForm frm){
        int dialogButton = JOptionPane.YES_NO_OPTION;
        int ct_id =
Integer.parseInt(String.valueOf(frm.getJCBCountryID().getSelectedIt

        String message = String.format("Are you sure you want to de
the row Country ID: %d)", ct_id);
        int answer = JOptionPane.showConfirmDialog(frm, message, "D
ROW OF DATA", dialogButton);

        if(answer == JOptionPane.YES_OPTION){
            String query = """
                DELETE FROM country WHERE country_id = ?""";
            try(Connection conn = getConnection();
                PreparedStatement ps = conn.prepareStatement(query)
                // Use PreparedStatement to avoid SQL injection att
                ps.setInt(1, ct_id);
                ps.executeUpdate();

                // Refresh table and comboboxes
                refresh_controls(frm);

            } catch (SQLException ex){
                JOptionPane.showMessageDialog(frm, ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
            }
        }
    }
```

Step 2

In **CountryForm.java**, double click on DELETE button to generate it listener:

```
    private void
```

```
jbDeleteActionPerformed(java.awt.event.ActionEvent
evt) {
        Country_Utils.delete_handler(this);
    }
```

Step 3	Run the project. Choose **country_id** using **jcbCountryID** or **jcb** combobox. Then, Click on DELETE button. The corresponding row of d been deleted from database.

This is the full version of **Country_Utils.java**:

```
package sakila;
import java.util.logging.Level;
import java.util.logging.Logger;
import java.sql.*;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashMap;
import java.util.Objects;
import javax.swing.JComboBox;
import javax.swing.JOptionPane;
import javax.swing.event.TableModelEvent;
import javax.swing.table.DefaultTableModel;
import javax.swing.table.TableModel;

public class Country_Utils extends Utility{
    public static final int FIRST_INDEX = 0;
    public static final int INVALID_INDEX = -1;

    private static int currentIndex =
FIRST_INDEX;
    private static final String SQL_ID =
Query_Country.get_sql_id();

    //Creates country table
    public static void create_country_table() {
        try (Connection conn = getConnection())
{
            Statement stmt =
conn.createStatement();

stmt.addBatch(Query_Country.get_sql_country());
            stmt.executeBatch();
```

```
            String message = String.format("Successfully creates country
table");
            JOptionPane.showMessageDialog(null, message,
            "INFORMATION",JOptionPane.INFORMATION_MESSAGE);

        } catch (SQLException ex) {
            JOptionPane.showMessageDialog(null, ex.getMessage(),
                "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    //Populates country table with some rows of data
    public static void populate_country_table(){
        try(Connection conn = getConnection()){
            String sql = """
                INSERT INTO country(country_id, country, last_update)
                VALUES(?, ?, ?)""";

            //Creates a new Country class with default constructor
            PreparedStatement ps1 = conn.prepareStatement(sql);
            Country obj1 = new Country();
            ps1.setInt(1,obj1.getCountryID());
            ps1.setString(2,obj1.getCountry());
            ps1.setTimestamp(3,obj1.getLastUpdate());

            // Creates a new Country class with three-params constructor
            PreparedStatement ps2 = conn.prepareStatement(sql);
            Country obj2 = new Country(2, "Toba State", new
Timestamp(System.currentTimeMillis()));
            ps2.setInt(1,obj2.getCountryID());
            ps2.setString(2,obj2.getCountry());
            ps2.setTimestamp(3,obj2.getLastUpdate());

            ps1.executeUpdate();
            ps2.executeUpdate();

        }catch(SQLException ex){
            JOptionPane.showMessageDialog(null, ex.getMessage(),
                "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    //Reads the content of country table
    public static void read_country_table(){
        try(Connection conn = getConnection()){
            Statement stmt = conn.createStatement();
            ResultSet rs = stmt.executeQuery("SELECT * FROM country");

            while(rs.next()){
                int ct_id = rs.getInt("country_id");
                String country = rs.getString("country");
                Timestamp lu = rs.getTimestamp("last_update");
```

```
                //Creates a Country object using three-params constructor
                Country obj = new Country(ct_id, country, lu);
                System.out.println(obj);
            }
            rs.close();
            stmt.close();

        }catch(SQLException ex){
            JOptionPane.showMessageDialog(null, ex.getMessage(),
                "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    private static ArrayList<Country> get_country_list(CountryForm frm, Strir
sql, String item){
        ArrayList<Country> list = new ArrayList<>();
        Connection conv = null;

        try(Connection conn = getConnection();
            PreparedStatement ps = conn.prepareStatement(sql)){
            if (item.equalsIgnoreCase("none")==false) {
                 ps.setString(1,item);
            }
            ResultSet rs = ps.executeQuery();

            Country obj;
            while(rs.next()){
                //Using three-params constructor
                obj = new Country(rs.getInt("country_id"),
                    rs.getString("country"),
                    rs.getTimestamp("last_update"));

                list.add(obj);
            }
        }catch (SQLException ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
                "ERROR",JOptionPane.ERROR_MESSAGE);
        }
        return list;
    }

    private static void show_table_country(CountryForm frm, ArrayList<Country
list) throws SQLException{
        DefaultTableModel model = new DefaultTableModel(0,0);

        String header[] = {"Country ID", "Country", "Last Update"};

        model.setColumnIdentifiers(set_column_header(frm.getJTCountry(),
header));
        frm.getJTCountry().setModel(model);

        Object[] row = new Object[3];
```

```
        for(int i=0; i<list.size(); i++){
            row[0] = list.get(i).getCountryID();
            row[1] = list.get(i).getCountry();
            row[2] = list.get(i).getLastUpdate();

            model.addRow(row);
        }
    }

    public static void refresh_controls(CountryForm frm){
        frm.setLocationRelativeTo(null);
        frm.setTitle("COUNTRY FORM");

        //Shows the content of country table and populates combobox
        try{
            //Makes alternating color for table rows
            table_renderer(frm.getJTCountry());

            //Populates table
            ArrayList<Country> list = get_country_list(frm, "SELECT * FROM
country", "none");
            show_table_country(frm, list);

            //Populates jcbCountryID
            String sql_id = "SELECT country_id FROM country ORDER BY
country_id";
            populate_combobox(sql_id, frm.getJCBCountryID(), frm);


            //Populates getJCBCountry()
            String sql_ct = "SELECT DISTINCT country FROM country ORDER BY
country";

            populate_combobox(sql_ct, frm.getJCBCountry(), frm);

        }catch (SQLException ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
            "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    private static void clear_controls(CountryForm frm){
        frm.getJTFCountryID().setText("");
        frm.getJTFCountry().setText("");
        frm.getJTFLastUpdate().setText("");
    }

   //Displays country data result row by row
    private static <T> void display_country_data(CountryForm frm, String sql,
item){
        try(Connection conn = getConnection()){
            PreparedStatement ps = conn.prepareStatement(sql);
```

```
            ps.setObject(1,item);
            ResultSet rs = ps.executeQuery();

            if (!rs.next()) {
                // no row found, clear the form fields
                clear_controls(frm);
                return;
            }

            do{

frm.getJTFCountryID().setText(String.valueOf(rs.getInt("country_id")));
                frm.getJTFCountry().setText(rs.getString("country"));

frm.getJTFLastUpdate().setText(String.valueOf(rs.getTimestamp("last_update"))

                // Determines item selected from jcbCountryID
                find_combo_value_selected(frm.getJCBCountryID(),
rs.getInt("country_id"));

                // Determines item selected from jcbCountry
                find_combo_value_selected(frm.getJCBCountry(),
rs.getString("country"));
            }while(rs.next());

            rs.close();
            ps.close();
        }catch(SQLException ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    public static void jcbCountry_handler(CountryForm frm, JComboBox<String>
jcb) {
        Object item = jcb.getSelectedItem();
        String sql = "";
        if (jcb.equals(frm.getJCBCountryID())) {
            sql = Query_Country.get_sql_id();
        } else if (jcb.equals(frm.getJCBCountry())) {
            sql = Query_Country.get_sql_name();
        }

        display_country_data(frm, sql, item);
    }

    public static void show_first_row(CountryForm frm){
        String item =
String.valueOf(frm.getJCBCountryID().getItemAt(FIRST_INDEX));
        display_country_data(frm, SQL_ID, item);
        currentIndex = FIRST_INDEX;
    }
```

```
    public static void show_last_row(CountryForm frm){
        int endIndex = frm.getJCBCountryID().getItemCount() - 1;
        String item =
String.valueOf(frm.getJCBCountryID().getItemAt(endIndex));
        display_country_data(frm, SQL_ID, item);
        currentIndex = endIndex;
    }

    public static void show_prev_row(CountryForm frm){
        currentIndex--;
        if(currentIndex < FIRST_INDEX){
            currentIndex = FIRST_INDEX;
            return;
        }
        String item =
String.valueOf(frm.getJCBCountryID().getItemAt(currentIndex));
        display_country_data(frm, SQL_ID, item);
    }

    public static void show_next_row(CountryForm frm){
        int endIndex = frm.getJCBCountryID().getItemCount() - 1;
        currentIndex++;
        if(currentIndex > endIndex){
            currentIndex = endIndex;
            return;
        }
        String item =
String.valueOf(frm.getJCBCountryID().getItemAt(currentIndex));
        display_country_data(frm, SQL_ID, item);
    }
    public static void mouse_pressed_handler(CountryForm frm) {
        Objects.requireNonNull(frm, "frm must not be null");

        int selectedIndex = frm.getJTCountry().getSelectedRow();
        if (selectedIndex == -1) {
            JOptionPane.showMessageDialog(frm, "Please select a row to view 
data.",
                "No row selected", JOptionPane.INFORMATION_MESSAGE);
            return;
        }

        try (Connection conn = getConnection()) {
            String id =
String.valueOf(frm.getJTCountry().getModel().getValueAt(selectedIndex, 0));

            // Displays country data
            display_country_data(frm, Query_Country.get_sql_id(), id);

        } catch (SQLException ex) {
            Logger.getLogger(CountryForm.class.getName()).log(Level.SEVERE,
"Error displaying country data", ex);
            String message = "Error displaying country data: " +
ex.getMessage();
```

```
            String stackTrace = Arrays.toString(ex.getStackTrace());
            JOptionPane.showMessageDialog(frm, message + "\n\n" + stackTrace,
"ERROR", JOptionPane.ERROR_MESSAGE);
        }

    }

    //Updates row of data in country tabel by country_id
    public static void update_row_by_country_id(CountryForm frm, int
country_id, String country) throws SQLException{
        Connection conn = getConnection();
        ResultSet rs = null;
        String query_id = "SELECT country_id FROM country WHERE country_id =
?";
        String update_query = """
            UPDATE country SET country = ? WHERE country_id = ?""";
        try(PreparedStatement idPs = conn.prepareStatement(query_id,

ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_READ_ONLY);
        PreparedStatement updatePS = conn.prepareStatement(update_query,
                ResultSet.TYPE_SCROLL_SENSITIVE, ResultSet.CONCUR_UPDATABLE))
        {
            idPs.setInt(1,country_id);
            if(!idPs.execute()){
                String message = "Can't find country_id " + country_id;

                JOptionPane.showMessageDialog(frm, message,
                    "ERROR",JOptionPane.ERROR_MESSAGE);
            } else{
                rs = idPs.getResultSet();
                rs.next();

                //Creates a Country object using three-params constructor
                Country obj = new Country(country_id, country, new
Timestamp(System.currentTimeMillis()));
                updatePS.setString(1, obj.getCountry());
                updatePS.setInt(2, obj.getCountryID());

                updatePS.executeUpdate();
                rs.close();
                updatePS.close();
                idPs.close();
                conn.close();
            }
        }catch(SQLException ex){
            Logger.getLogger(CountryForm.class.getName()).log(Level.SEVERE,
"Error updating country data", ex);
            String message = "Error updating country data: " + ex.getMessage(
            String stackTrace = Arrays.toString(ex.getStackTrace());
            JOptionPane.showMessageDialog(null, message + "\n\n" + stackTrace
"ERROR", JOptionPane.ERROR_MESSAGE);
        }catch(java.lang.NumberFormatException ex){
```

```
            Logger.getLogger(CountryForm.class.getName()).log(Level.SEVERE,
"Invalid Input", ex);
            String message = "Invalid Input: " + ex.getMessage();
            String stackTrace = Arrays.toString(ex.getStackTrace());
            JOptionPane.showMessageDialog(null, message + "\n\n" + stackTrace
"ERROR", JOptionPane.ERROR_MESSAGE);
        }
    }

    private static HashMap<String, String> read_inputs(CountryForm frm) {
        HashMap<String, String> input_data = new HashMap<>();
        String country_id =
String.valueOf(frm.getJCBCountryID().getSelectedItem());
        String country = frm.getJTFCountry().getText();

        // Validate user input
        int country_id_int = 0;
        try {
            country_id_int = Integer.parseInt(country_id);
            if (country_id_int <= 0) {
                throw new IllegalArgumentException("CountryID cannot be
negative or zero");
            }
        } catch (NumberFormatException ex) {
            JOptionPane.showMessageDialog(frm, "Invalid Country ID: " +
country_id,
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        } catch (IllegalArgumentException ex) {
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        }

        if (country == null || country.isEmpty()) {
            JOptionPane.showMessageDialog(frm, "Country cannot be empty",
            "Error", JOptionPane.ERROR_MESSAGE);
            throw new IllegalArgumentException("Country cannot be empty");
        }

        input_data.put("country_id", country_id);
        input_data.put("country", country);

        return input_data;
    }

    private static void edit_actual(CountryForm frm){
        try{
            HashMap<String, String> input_data = read_inputs(frm);
            int country_id = Integer.parseInt(input_data.get("country_id"));
            String country = input_data.get("country");

            update_row_by_country_id(frm, country_id, country);
```

```
            //Refreshes all objects on form
            refresh_controls(frm);

        }catch(SQLException ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    private static void enable_controls(boolean state, CountryForm frm){
        frm.getJBFirst().setEnabled(state);
        frm.getJBPrev().setEnabled(state);
        frm.getJBNext().setEnabled(state);
        frm.getJBLast().setEnabled(state);
        frm.getJBInsert().setEnabled(state);
        frm.getJBDelete().setEnabled(state);
        frm.getJTFCountryID().setEnabled(state);
    }

    public static void edit_handler(CountryForm frm){
        if(frm.getJBEdit().getText().equals("EDIT")){
            frm.getJBEdit().setText("CONFIRM");

            // Disables controls
            enable_controls(false, frm);
        }

        else {
            frm.getJBEdit().setText("EDIT");

            // Actual editing
            edit_actual(frm);

            //Enables controls
            enable_controls(true, frm);
        }
    }

    public static void edit_database_from_jtable(TableModelEvent e, CountryFo
frm){
        if (e.getType() == TableModelEvent.UPDATE) {
            int row = e.getFirstRow();
            TableModel model = (TableModel)e.getSource();
            int country_id =
Integer.parseInt(String.valueOf(model.getValueAt(row, 0)));
            String name = (String) model.getValueAt(row, 1);

            try{
                update_row_by_country_id(frm, country_id, name);

                //Refreshes all objects on form
                refresh_controls(frm);
```

```
            } catch (SQLIntegrityConstraintViolationException ex) {
                Logger.getLogger(CountryForm.class.getName()).log(Level.SEVE
"Duplicate entry", ex);
                JOptionPane.showMessageDialog(frm, "Error: Duplicate entry\n'
ex.getMessage());
            } catch (SQLSyntaxErrorException ex) {
                Logger.getLogger(CountryForm.class.getName()).log(Level.SEVE
"Invalid SQL syntax", ex);
                JOptionPane.showMessageDialog(frm, "Error: Invalid SQL
syntax\n" + ex.getMessage());
            } catch (SQLException ex) {
                Logger.getLogger(CountryForm.class.getName()).log(Level.SEVE
"Database error", ex);
                JOptionPane.showMessageDialog(frm, "Error: Database error\n"
ex.getMessage());
            }
        }
    }

    //Inserts new row into country table
    private static void insert_row(CountryForm frm) throws SQLException{
        HashMap<String, String> input_data = read_inputs(frm);
        String country = input_data.get("country");

        // SQL insert statement
        String sql = """
            INSERT INTO country(country) VALUES(?)""";

        try(Connection conn = getConnection();
         PreparedStatement pstmt = conn.prepareStatement(sql)){

            //Creates a Country object two-params constructor
            Country obj = new Country(country, new
Timestamp(System.currentTimeMillis()));
            pstmt.setString(1,obj.getCountry());

            //Executes the sql insert statement
            pstmt.executeUpdate();
        } catch (SQLException ex) {
            Logger.getLogger(CountryForm.class.getName()).log(Level.SEVERE,
"Database error", ex);
            JOptionPane.showMessageDialog(frm, "Error: Database error\n" +
ex.getMessage());
        }
    }

    private static void insert_actual(CountryForm frm){
        try{
            insert_row(frm);

            //Refreshes table and comboboxes
            refresh_controls(frm);
```

```
        }catch(SQLException ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    public static void insert_handler(CountryForm frm){
        if(frm.getJBInsert().getText().equals("INSERT")){
            frm.getJBInsert().setText("CONFIRM");

            //Disables jbEdit
            frm.getJBEdit().setEnabled(false);

            // Disables controls
            enable_controls(false, frm);
            frm.getJCBCountryID().setEnabled(false);
            frm.getJCBCountry().setEnabled(false);

            // Clears controls
            clear_controls(frm);

            // Enables
            frm.getJBInsert().setEnabled(true);
        }

        else {
            frm.getJBInsert().setText("INSERT");

            // Actual insertion
            insert_actual(frm);

            //Enables jbEdit
            frm.getJBEdit().setEnabled(true);

            //Enables controls
            enable_controls(true, frm);
            frm.getJCBCountryID().setEnabled(true);
            frm.getJCBCountry().setEnabled(true);
        }
    }

    public static void delete_handler(CountryForm frm){
        int dialogButton = JOptionPane.YES_NO_OPTION;
        int ct_id =
Integer.parseInt(String.valueOf(frm.getJCBCountryID().getSelectedItem()));

        String message = String.format("Are you sure you want to delete the r
Country ID: %d)", ct_id);
        int answer = JOptionPane.showConfirmDialog(frm, message, "DELETING RO
OF DATA", dialogButton);

        if(answer == JOptionPane.YES_OPTION){
            String query = """
```

```
                DELETE FROM country WHERE country_id = ?""";
            try(Connection conn = getConnection();
                PreparedStatement ps = conn.prepareStatement(query)){
                // Use PreparedStatement to avoid SQL injection attacks
                ps.setInt(1, ct_id);
                ps.executeUpdate();

                // Refresh table and comboboxes
                refresh_controls(frm);

            } catch (SQLException ex){
                JOptionPane.showMessageDialog(frm, ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
            }
        }
    }
}
```

This is the full version of **CountryForm.java**:

```
package sakila;
import java.awt.Toolkit;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.IOException;
import java.text.ParseException;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.JButton;
import javax.swing.JComboBox;
import javax.swing.JLabel;
import javax.swing.JTable;
import javax.swing.JTextField;

public class CountryForm extends javax.swing.JFrame {
    public CountryForm() {
        initComponents();
        Utility.setLookAndFeel(this);
        Country_Utils.refresh_controls(this);

 this.setIconImage(Toolkit.getDefaultToolkit().getImage(getClass().getResour
;
        this.setDefaultCloseOperation(this.HIDE_ON_CLOSE);
    }

    //Getter method for jtfCountryID
    public JTextField getJTFCountryID(){
        return this.jtfCountryID;
    }

    //Getter method for jtfLastUpdate
```

```
    public JTextField getJTFLastUpdate(){
        return this.jtfLastUpdate;
    }

    //Getter method for jtfCountry
    public JTextField getJTFCountry(){
        return this.jtfCountry;
    }

    //Getter method for jtCountry
    public JTable getJTCountry(){
        return this.jtCountry;
    }

    //Getter method for jcbCountryID
    public JComboBox getJCBCountryID(){
        return this.jcbCountryID;
    }

    //Getter method for jcbCountry
    public JComboBox getJCBCountry(){
        return this.jcbCountry;
    }

    //Getter method for jbEdit
    public JButton getJBEdit(){
        return this.jbEdit;
    }

    //Getter method for jbInsert
    public JButton getJBInsert(){
        return this.jbInsert;
    }

    //Getter method for jbDelete
    public JButton getJBDelete(){
        return this.jbDelete;
    }

    //Getter method for jbFirst
    public JButton getJBFirst(){
        return this.jbFirst;
    }

    //Getter method for jbPrev
    public JButton getJBPrev(){
        return this.jbPrev;
    }

    //Getter method for jbNext
    public JButton getJBNext(){
        return this.jbNext;
    }
```

```
    //Getter method for jbLast
    public JButton getJBLast(){
        return this.jbLast;
    }

    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">
    private void initComponents() {
        //...
        pack();
    }// </editor-fold>

    private void jcbCountryIDActionPerformed(java.awt.event.ActionEvent evt)
        Country_Utils.jcbCountry_handler(this, this.jcbCountryID);
    }

    private void jcbCountryActionPerformed(java.awt.event.ActionEvent evt) {
        Country_Utils.jcbCountry_handler(this, this.jcbCountry);
    }

    private void jbEditActionPerformed(java.awt.event.ActionEvent evt) {
        Country_Utils.edit_handler(this);
    }

    private void jbInsertActionPerformed(java.awt.event.ActionEvent evt) {
        Country_Utils.insert_handler(this);
    }

    private void jbDeleteActionPerformed(java.awt.event.ActionEvent evt) {
        Country_Utils.delete_handler(this);
    }

    private void jbLastActionPerformed(java.awt.event.ActionEvent evt) {
        Country_Utils.show_last_row(this);
    }

    private void jbNextActionPerformed(java.awt.event.ActionEvent evt) {
        Country_Utils.show_next_row(this);
    }

    private void jbPrevActionPerformed(java.awt.event.ActionEvent evt) {
        Country_Utils.show_prev_row(this);
    }

    private void jbFirstActionPerformed(java.awt.event.ActionEvent evt) {
        Country_Utils.show_first_row(this);
    }

    private void jtCountryMousePressed(java.awt.event.MouseEvent evt) {
        Country_Utils.mouse_pressed_handler(this);
    }
```

```
    private void jtCountryMouseClicked(java.awt.event.MouseEvent evt) {
        // instantiate CountryTableModelListener and add it as a listener to
        CountryTableModelListener tableModelListener = new
CountryTableModelListener(this.getJTCountry(), this);
        this.getJTCountry().getModel().addTableModelListener(tableModelListe
    }

    /**
     * @param args the command line arguments
     */
    public static void main(String args[]) {
        try {
            for (javax.swing.UIManager.LookAndFeelInfo info :
javax.swing.UIManager.getInstalledLookAndFeels()) {
                if ("Nimbus".equals(info.getName())) {
                    javax.swing.UIManager.setLookAndFeel(info.getClassName()
                    break;
                }
            }
        } catch (ClassNotFoundException ex) {

java.util.logging.Logger.getLogger(CountryForm.class.getName()).log(java.util
null, ex);
        } catch (InstantiationException ex) {

java.util.logging.Logger.getLogger(CountryForm.class.getName()).log(java.util
null, ex);
        } catch (IllegalAccessException ex) {

java.util.logging.Logger.getLogger(CountryForm.class.getName()).log(java.util
null, ex);
        } catch (javax.swing.UnsupportedLookAndFeelException ex) {

java.util.logging.Logger.getLogger(CountryForm.class.getName()).log(java.util
null, ex);
        }
        /* Create and display the form */
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new CountryForm().setVisible(true);
            }
        });
    }

    // Variables declaration - do not modify
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JLabel jLabel4;
    private javax.swing.JScrollPane jScrollPane1;
    private javax.swing.JButton jbDelete;
    private javax.swing.JButton jbEdit;
    private javax.swing.JButton jbFirst;
```

```
    private javax.swing.JButton jbInsert;
    private javax.swing.JButton jbLast;
    private javax.swing.JButton jbNext;
    private javax.swing.JButton jbPrev;
    private javax.swing.JComboBox<String> jcbCountry;
    private javax.swing.JComboBox<String> jcbCountryID;
    private javax.swing.JTable jtCountry;
    private javax.swing.JTextField jtfCountry;
    private javax.swing.JTextField jtfCountryID;
    private javax.swing.JTextField jtfLastUpdate;
    // End of variables declaration
}
```

CITY FORM

CITY FORM

CREATING AND POPULATING CITY TABLE

CREATING AND POPULATING CITY TABLE

Step 1	Create a new class named **Query_City**. It contains several constant SQL queries for interacting with a database that has a table named city. Here is an explanation of each query: • **sql_min**: a SQL query that selects the minimum **city_id** value from the **city** table. • **sql_max**: a SQL query that selects the maximum **city_id** value from the **city** table. • **sql_id**: a SQL query that selects all columns from the **city** table where the **city_id** matches a parameterized value (indicated by the ?). • **sql_name**: a SQL query that selects all columns from the **city** table where the **city** matches a parameterized value. • **sql_city_country_dist**: a SQL query that joins the **city** and **country** tables to determine the number of cities in each country, sorted in descending

order of city count and then ascending order of country name, and limited to the top 10 results.

- **sql_city_joint**: a SQL query that joins the **city** and **country** tables and selects several columns from both tables.
- **sql_city**: a SQL query that creates a **city** table with several columns, including an auto-incrementing **city_id** primary key and a foreign key constraint referencing the **country** table.

The class also contains getter methods for each of the constant SQL queries, which can be used to retrieve the queries from other parts of the Java program.

```
1  package sakila;
2
3  public class Query_City {
4      private static final String
5  sql_min = "SELECT MIN(city_id) FROM
6  city";
7      private static final String
8  sql_max = "SELECT MAX(city_id) FROM
9  city";
10     private static final String
11 sql_id = "SELECT * FROM city WHERE
12 city_id  = ?";
13     private static final String
14 sql_name = "SELECT * FROM city
15 WHERE city = ?";
16
17     private static final String
18 sql_city_country_dist = """
19         SELECT co.country, COUNT(*)
20 AS Number
21         FROM city ci
22         JOIN country co ON
23 co.country_id = ci.country_id
24         GROUP BY co.country
25         ORDER BY Count(*) DESC,
26 co.country ASC
27         LIMIT 10""";
28
29     private static final String
30 sql_city_joint = """
31         SELECT ci.city_id, ci.city,
32 ci.country_id,
```

```
            ci.last_update,
co.country
        FROM city ci
        JOIN country co ON
co.country_id = ci.country_id""";

    private static final String
sql_city = """
        CREATE TABLE city (
          city_id SMALLINT UNSIGNED
NOT NULL AUTO_INCREMENT,
          city VARCHAR(50) NOT
NULL,
          country_id SMALLINT
UNSIGNED NOT NULL,
          last_update TIMESTAMP NOT
NULL DEFAULT CURRENT_TIMESTAMP ON
UPDATE CURRENT_TIMESTAMP,
          PRIMARY KEY  (city_id),
          KEY idx_fk_country_id
(country_id),
          CONSTRAINT
`fk_city_country` FOREIGN KEY
(country_id) REFERENCES country
(country_id) ON DELETE RESTRICT ON
UPDATE CASCADE
        ) ENGINE=InnoDB DEFAULT
CHARSET=utf8mb4;""";

    //Getter methods
    public static String
get_sql_min() {
        return sql_min;
    }

    public static String
get_sql_max() {
        return sql_max;
    }

    public static String
get_sql_id() {
        return sql_id;
    }

    public static String
get_sql_name() {
        return sql_name;
    }

    public static String
get_sql_city() {
        return sql_city;
```

```
    }

    public static String
get_sql_city_joint() {
        return sql_city_joint;
    }

    public static String
get_sql_city_country_dist() {
        return
sql_city_country_dist;
    }
}
```

Step 2

Then, create a public class named **City**. It represents a city in the Sakila database. It has five instance variables: **city_id**, **city**, **country_id**, **country**, and **last_update**.

The class has five constructors:

1. A default constructor that initializes the instance variables to default values.
2. A three-parameter constructor that takes a city name, country ID, and last update timestamp as arguments.
3. A four-parameter constructor that takes a city ID, city name, country ID, and last update timestamp as arguments.
4. A five-parameter constructor that takes a city ID, city name, country ID, country name, and last update timestamp as arguments.

Getter and setter methods for each instance variable. The class overrides two methods: **hashCode()** and **equals()**. The **hashCode()** method returns a hash code for the object based on the instance variables. The **equals()** method compares two objects based on the values of the instance variables.

The class also has a **toString()** method that returns a string representation of the object.

The string includes the values of all the instance variables.

```
1  package sakila;
2  import java.util.Objects;
3  import java.sql.Timestamp;
4
5  public class City {
6      private int city_id;
7      private String city;
8      private int country_id;
9      private String country;
10     private Timestamp last_update;
11
12     //Default constructor
13     City(){
14         this(1, "Balige", 2, new
15 Timestamp(System.currentTimeMillis()));
16     }
17
18     //Three-params constructor
19     City(String city, int country_id,
20 Timestamp lu){
21         setCity(city);
22         setCountryID(country_id);
23         setLastUpdate(lu);
24     }
25
26
27     //Four-params constructor
28     City(int city_id, String city, int
29 country_id, Timestamp lu){
30         this(city, country_id, lu);
31         setCityID(city_id);
32     }
33
34     //Five-params constructor
35     City(int city_id, String city, int
36 country_id, String country, Timestamp
37 lu){
38         this(city_id, city, country_id,
39 lu);
40         this.country = country;
41     }
42
43
44
45
46
47
48
49
50
```

	51
	52
	53
	54
	55
	56
	57
	58
	59
	60
	61
	62
	63
	64
	65
	66
	67
	68
	69
	70
	71
	72
	73
	74
	75
	76
	77
	78
	79
	80
	81
	82
	83
	84
	85
	86
	87
	88
	89
	90
	91
	92
	93
	94
	95
	96
	97
	98
	99
	100
	101
	102
	103
	104


105

```
    // Getter methods
    public int getCityID() {return
city_id;}
    public String getCity() {return
city;}
    public int getCountryID() {return
country_id;}
    public String getCountry() {return
country;}
    public Timestamp getLastUpdate()
{return last_update;}

    //Setter methods
    public void setCityID(int id) {
        if (id <= 0) {
            throw new
IllegalArgumentException("City ID must
be greater than zero.");
        }
        this.city_id = id;
    }

    public void setCity(String name) {
        if (name == null ||
name.trim().isEmpty()) {
            throw new
IllegalArgumentException("City cannot
be null or empty");
        }
        if (name.length() > 50) {
            throw new
IllegalArgumentException("City cannot
be longer than 50 characters");
        }
        this.city = name;
    }

    public void setCountryID(int id) {
        if (id <= 0) {
            throw new
IllegalArgumentException("Country ID
must be greater than zero.");
        }
        this.country_id = id;
    }

    public void setLastUpdate(Timestamp
date){
        if (date == null) {
        throw new
IllegalArgumentException("Date cannot
be null");
        }
```

```
        this.last_update = date;
    }

    // Override the hashCode() method
    @Override
    public int hashCode() {
        return Objects.hash(city_id,
city, country_id, country,
last_update);
    }

    // Override the equals() method
    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() !=
o.getClass()) return false;
        City ct = (City) o;
        return city_id == ct.city_id &&
               country_id ==
ct.country_id &&
               Objects.equals(city,
ct.city) &&
               Objects.equals(country,
ct.country) &&

Objects.equals(last_update,
ct.last_update);
    }

    @Override
    public String toString(){
        return "\nCity ID       : " +
getCityID() +
               "\nCity          : " +
getCityID() +
               "\nCountry ID    : " +
getCountryID() +
               "\nCountry       : " +
getCountry() +
               "\nLast Update   : " +
getLastUpdate();
    }
}
```

Step 3	Create a new public class named (the **Utility** class. It extends the **Ut** several static methods that perform on the **city** table in the Sakila sampl

The class has the following methods

1. **create_city_table()**: This m
   table in the database usi
   obtained from the **Query_C**
   success message if the tal
   error message if there is a S
2. **populate_city_table()**: This
   rows of data into the **city** t
   created using a **City** obj
   constructor, and the second
   City object with a four-para
   catches any SQL exceptions
   message.
3. **read_city_table()**: This
   contents of the **city** and **c**
   together, and creates a **City**
   data. It prints out the details
   the console. Any SQL excep
   an error message is displaye

The class also defines some stati
constants:

- **FIRST_INDEX**: This is a co
  0, used to define the first ind
- **INVALID_INDEX**: This is a
  of -1, used to indicate an inv
  or list.
- **currentIndex**: This is a st
  keep track of the current ind
- **SQL_ID**: This is a constant t
  query on the **city** and **count**
  clause is then appended to t
  the results based on th
  parameter.

The class imports several Java
"java.sql" for working with databa
displaying dialog boxes, "java.util"
and **HashMap** classes, and "java.ut
messages.

```
1  package sakila;
2  import java.util.logging.Level;
3  import java.util.logging.Logger
4  import java.sql.*;
5  import java.util.ArrayList;
6  import java.util.Arrays;
7  import java.util.HashMap;
8  import java.util.Objects;
```

```
import javax.swing.JComboBox;
import javax.swing.JOptionPane;
import javax.swing.event.TableM
import javax.swing.table.Defaul
import javax.swing.table.TableM

public class City_Utils extends
    public static final int FIR
    public static final int INV

    private static int currentI
    private static final String
Query_City.get_sql_city_joint()
?";

    //Creates city table
    public static void create_c
        try (Connection conn =
            Statement stmt =
conn.createStatement();

stmt.addBatch(Query_City.get_sc
            stmt.executeBatch()

            String message =
String.format("Successfully cre
            JOptionPane.showMes
message,

 "INFORMATION",JOptionPane.INFO

        } catch (SQLException e
            JOptionPane.showMes
ex.getMessage(),
                "ERROR",JOptior
        }
    }

    //Populates city table with
    public static void populate
        try(Connection conn = g
            String sql = """
                INSERT INTO cit
country_id, last_update)
                VALUES(?, ?, ?,

            //Creates a new Cit
constructor
            PreparedStatement p
conn.prepareStatement(sql);
            City obj1 = new Cit
            ps1.setInt(1,obj1.g
            ps1.setString(2,obj
```

```
63              ps1.setInt(3,obj1.g
64
65  ps1.setTimestamp(4,obj1.getLast
66
67              // Creates a new Ci
68  params constructor
69              PreparedStatement p
70  conn.prepareStatement(sql);
71              City obj2 = new Cit
72  new Timestamp(System.currentTim
73              ps2.setInt(1,obj2.g
74              ps2.setString(2,obj
75              ps2.setInt(3,obj2.g
76
77  ps2.setTimestamp(4,obj2.getLast
78
79              ps1.executeUpdate()
80              ps2.executeUpdate()
81
82          }catch(SQLException ex)
83              JOptionPane.showMes
84  ex.getMessage(),
85                      "ERROR",JOptior
86          }
87      }
88
89      //Reads the content of join
90  tables
91      public static void read_cit
92          try(Connection conn = g
93              Statement stmt =
94  conn.createStatement();
95              ResultSet rs =
96  stmt.executeQuery(Query_City.ge
97
                while(rs.next()){
                    int cty_id = rs
                    String city = r
                    int ct_id =
  rs.getInt("country_id");
                    String country
  rs.getString("country");
                    Timestamp lu =
  rs.getTimestamp("last_update");

                    //Creates a Cit
  params constructor
                    City obj = new
  ct_id, country, lu);
                    System.out.prin
                }
                rs.close();
                stmt.close();
```

```
        }catch(SQLException ex)
            JOptionPane.showMes
ex.getMessage(),
                    "ERROR",JOptior
        }
    }
}
```

Step 4

In the driver class, **S**
create_city_table(), **populate_**
read_city_table() as shown in line 4

```
1  package sakila;
2
3  public class Sakila {
4      public static void main(Str
5  //         Utility.testConnectic
6  //         Actor_Utils.create_ac
7  //         Actor_Utils.populate_
8  //         Actor_Utils.read_acto
9  //         ActorForm frm = new A
10 //         frm.setVisible(true);
11
12 //         Language_Utils.create
13 //
14 Language_Utils.populate_languag
15 //         Language_Utils.read_l
16 //         LanguageForm frm = ne
17 //         frm.setVisible(true);
18
19 //         Category_Utils.create
20 //
21 Category_Utils.populate_categor
22 //         Category_Utils.read_c
23 //         CategoryForm frm = ne
24 //         frm.setVisible(true);
25
26 //         Film_Utils.create_fil
27 //         Film_Utils.populate_f
28 //         Film_Utils.read_film_
29 //         FilmForm frm = new Fi
30 //         frm.setVisible(true);
31
32
33 //
34 FilmActor_Utils.create_film_act
35 //
36 FilmActor_Utils.populate_film_a
37 //         FilmActor_Utils.read_
38 //         FilmActorForm frm = n
```

```
39 //        frm.setVisible(true);
40
41 //
42 FilmCategory_Utils.create_film_
43 //
44 FilmCategory_Utils.populate_fil
45 //
46 FilmCategory_Utils.read_film_ca
47 //        FilmCategoryForm frm
48 FilmCategoryForm();
49 //        frm.setVisible(true);
50
51 //        Country_Utils.create_
52 //        Country_Utils.populat
   //        Country_Utils.read_co
   //        CountryForm frm = new
   //        frm.setVisible(true);

           City_Utils.create_city_
           City_Utils.populate_cit
           City_Utils.read_city_ta
       }
   }
```

Run project to see the result in cons

```
City ID       : 1
City          : 1
Country ID    : 2
Country       : Toba State
Last Update   : 2023-04-27 07:48:58

City ID       : 2
City          : 2
Country ID    : 2
Country       : Toba State
Last Update   : 2023-04-27 07:48:58
```

DESIGNING GUI
DESIGNING GUI

Step 1	In the project, create a new **JFrame Form** and name it as **CityForm.java**. In the Design tab, add five **JLabels** to the form and set their corresponding **text** properties as CITY ID, CITY, COUNTRY ID, COUNTRY, and LAST UPDATE.
Step	Then, add three **JTextField** to the form and set their

2	corresponding **Variable Name** as **jtfCityID**, **jtfCity**, and **jtfLastUpdate**.
Step 3	Then, add eight **JButton** to the form and set their corresponding **Variable Name** as **jbFirst**, **jbPrev**, **jbNext**, **jbLast**, **jbEdit**, **jbInsert**, **jbDelete**, and **jbChart**. Set their corresponding **text** properties as FIRST, PREV, NEXT, LAST, EDIT, INSERT, DELETE, and CHART.
Step 4	Then, add four **JComboBoxes** to the form and set their corresponding **Variable Name** as **jbCityID**, **jbCity**, **jcbCountryID** and **jcbCountry**.
Step 5	Lastly, add a new **JTable** to the form set set its **Variable Name** as **jtCity**. Then, right-click on it, then choose **Table Contents...** and set the number of columns to 5 and the number of rows to 25.
Step 6	In the driver class, **Sakila.java**, create a new object of **CityForm** class using its default constructor as shown in 51 – 52:

```
1  package sakila;
2
3  public class Sakila {
4      public static void main(String[] args) {
5  //         Utility.testConnection();
6  //         Actor_Utils.create_actor_table();
7  //         Actor_Utils.populate_actor_table();
8  //         Actor_Utils.read_actor_table();
9  //         ActorForm frm = new ActorForm();
10 //         frm.setVisible(true);
11
12 //         Language_Utils.create_language_table();
13 //
14 Language_Utils.populate_language_table();
15 //         Language_Utils.read_language_table();
16 //         LanguageForm frm = new LanguageForm();
17 //         frm.setVisible(true);
18
19 //         Category_Utils.create_category_table();
20 //
21 Category_Utils.populate_category_table();
22 //         Category_Utils.read_category_table();
23 //         CategoryForm frm = new CategoryForm();
24 //         frm.setVisible(true);
25
```

```
26
27  //        Film_Utils.create_film_table();
28  //        Film_Utils.populate_film_table();
29  //        Film_Utils.read_film_table();
30  //        FilmForm frm = new FilmForm();
31  //        frm.setVisible(true);
32
33  //
34  FilmActor_Utils.create_film_actor_table();
35  //
36  FilmActor_Utils.populate_film_actor_table();
37  //        FilmActor_Utils.read_film_actor_table();
38  //        FilmActorForm frm = new FilmActorForm();
39  //        frm.setVisible(true);
40
41  //
42  FilmCategory_Utils.create_film_category_table();
43  //
44  FilmCategory_Utils.populate_film_category_table();
45  //
46  FilmCategory_Utils.read_film_category_table();
47  //        FilmCategoryForm frm = new
48  FilmCategoryForm();
49  //        frm.setVisible(true);
50
51  //        Country_Utils.create_country_table();
52  //        Country_Utils.populate_country_table();
53  //        Country_Utils.read_country_table();
54  //        CountryForm frm = new CountryForm();
    //        frm.setVisible(true);

    //        City_Utils.create_city_table();
    //        City_Utils.populate_city_table();
    //        City_Utils.read_city_table();
            CityForm frm = new CityForm();
            frm.setVisible(true);
        }
    }
```

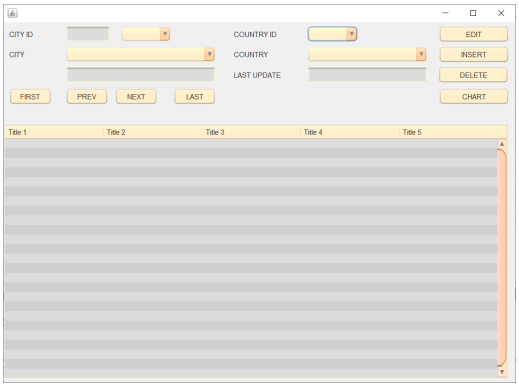


Figure 9.1 The layout of city form

Step 8	In **CityForm**'s constructor, invoke **setLookAndFeel()** to set the look and feel of the form as shown in line 17.
	
	1 package sakila;
	2
	3 import java.awt.Toolkit;
	4 import java.awt.event.ActionEvent;
	5 import
	6 java.awt.event.ActionListener;
	7 import javax.swing.JButton;
	8 import javax.swing.JComboBox;
	9 import javax.swing.JMenuItem;
	10 import javax.swing.JOptionPane;
	11 import javax.swing.JPopupMenu;
	12 import javax.swing.JTable;
	13 import javax.swing.JTextField;
	14
	15 public class CityForm extends
	16 javax.swing.JFrame {
	17 public CityForm() {
	18 initComponents();
	19
	20 Utility.setLookAndFeel(this);
	} //...
	}
	
	Run the project to see the city form as shown in Figure 9.1.

Step 9	In **CityForm.java**, define **getter()** method for every object in the form and place them outside and beneath its default constructor:
	
	1 //Getter method for jtfCityID
	2 public JTextField
	3 getJTFCityID(){
	4 return this.jtfCityID;
	5 }
	6
	7 //Getter method for
	8 jtfLastUpdate
	9 public JTextField
	10 getJTFLastUpdate(){
	11 return this.jtfLastUpdate;
	12 }
	13
	14 //Getter method for jtfCity

```
    public JTextField getJTFCity(){
        return this.jtfCity;
    }

    //Getter method for jtCity
    public JTable getJTCity(){
        return this.jtCity;
    }

    //Getter method for jcbCityID
    public JComboBox getJCBCityID()
{
        return this.jcbCityID;
    }

    //Getter method for jcbCity
    public JComboBox getJCBCity(){
        return this.jcbCity;
    }

    //Getter method for
jcbCountryID
    public JComboBox
getJCBCountryID(){
        return this.jcbCountryID;
    }

    //Getter method for jcbCountry
    public JComboBox
getJCBCountry(){
        return this.jcbCountry;
    }

    //Getter method for jbEdit
    public JButton getJBEdit(){
        return this.jbEdit;
    }

    //Getter method for jbInsert
    public JButton getJBInsert(){
        return this.jbInsert;
    }

    //Getter method for jbDelete
    public JButton getJBDelete(){
        return this.jbDelete;
    }

    //Getter method for jbFirst
    public JButton getJBFirst(){
        return this.jbFirst;
    }

```

```
69      //Getter method for jbPrev
70      public JButton getJBPrev(){
71          return this.jbPrev;
72      }
73
74      //Getter method for jbNext
75      public JButton getJBNext(){
76          return this.jbNext;
77      }
78
79      //Getter method for jbLast
        public JButton getJBLast(){
            return this.jbLast;
        }

        //Getter method for jbChart
        public JButton getJBChart(){
            return this.jbChart;
        }
```

POPULATING TABLE AND COMBOBOXES
POPULATING TABLE AND COMBOBOXES

Step 1

In **City_Utils.java**, add two new methods: **get_city_list()** and **show_table** takes three parameters: a **CityForm** object, a SQL query string, and a **Str** returns an **ArrayList** of **City** objects.

Inside the method, it creates a new **ArrayList** object and sets a **Connec** establish a database connection using the **getConnection()** method and p query string provided. If the item string is not equal to "none," the staten item string. The method then executes the SQL query and retrieves the res

The method then loops through the **ResultSet** and creates new City ob parameter constructor. The constructor takes an integer **city_id**, a String integer **country_id**. It also takes two more parameters, a String repr **Timestamp** object representing the last update time.

The **show_table_city()** method takes two parameters: a **CityForm** objec The method creates a new **DefaultTableModel** object with zero rows an headers using the **set_column_header()** method and sets the model **CityForm** object. The method then creates an object array with five **ArrayList**, adding a new row to the model for each **City** object. The row d attributes: **city_id**, **city**, **country_id**, **country**, and **last_update**.

```
    private static ArrayList<City> get_city_list(CityForm
frm, String sql, String item){
        ArrayList<City> list = new ArrayList<>();
        Connection conv = null;

        try(Connection conn = getConnection();
            PreparedStatement ps =
conn.prepareStatement(sql)){
            if (item.equalsIgnoreCase("none")==false) {
                 ps.setString(1,item);
            }
            ResultSet rs = ps.executeQuery();

            City obj;
            while(rs.next()){
                //Using three-params constructor
                obj = new City(rs.getInt("city_id"),
                    rs.getString("city"),
                    rs.getInt("country_id"),
                    rs.getString("country"),
                    rs.getTimestamp("last_update"));

                list.add(obj);
            }
        }catch (SQLException ex){
            JOptionPane.showMessageDialog(frm,
ex.getMessage(),
                "ERROR",JOptionPane.ERROR_MESSAGE);
        }
        return list;
    }

    private static void show_table_city(CityForm frm,
ArrayList<City> list) throws SQLException{
        DefaultTableModel model = new DefaultTableModel(0,0);

        String header[] = {"City ID", "City", "Country ID",
"Country", "Last Update"};


model.setColumnIdentifiers(set_column_header(frm.getJTCity(),
header));
        frm.getJTCity().setModel(model);

        Object[] row = new Object[5];

        for(int i=0; i<list.size(); i++){
            row[0] = list.get(i).getCityID();
            row[1] = list.get(i).getCity();
            row[2] = list.get(i).getCountryID();
            row[3] = list.get(i).getCountry();
            row[4] = list.get(i).getLastUpdate();


            model.addRow(row);
```

```
        }
    }
```

Step 2

In **City_Utils.java**, define **refresh_controls()** method. It takes a **CityF** purpose of this method is to refresh the contents of several GUI controls o

Inside the method, it sets the location of the form to the center of the s method then proceeds to refresh the content of several controls by exec and updating the corresponding controls on the form.

The method first calls the **table_renderer()** method to apply an alt component of the **CityForm** object.

Next, the method calls the **get_city_list()** method to retrieve a list of **Ci JTable** component. It calls populate_combobox method for four differ **jcbCityID**, **jcbCity**, **jcbCountryID**, and **jcbCountry** combo boxes, respec

Each **populate_combobox()** method call takes a SQL query string, a **CityForm** object as parameters. The method executes the query, retriev **JComboBox** with the values returned by the query.

Finally, if an error occurs while executing the SQL queries, the meth showing the error message.

```
1   public static void
2  refresh_controls(CityForm frm){
3
4  frm.setLocationRelativeTo(null);
5          frm.setTitle("CITY FORM");
6
7          //Shows the content of
8  country table and populates
9  combobox
10         try{
11             //Makes alternating
12 color for table rows
13
14 table_renderer(frm.getJTCity());
15
16             //Populates table
17             ArrayList<City> list =
18 get_city_list(frm,
19 Query_City.get_sql_city_joint() + "
20 ORDER BY city_id", "none");
21             show_table_city(frm,
22 list);
23
24             //Populates jcbCityID
```

```
            String sql_city_id =
 "SELECT city_id FROM city ORDER BY
city_id";

populate_combobox(sql_city_id,
frm.getJCBCityID(), frm);

            //Populates jcbCity
            String sql_city =
 "SELECT DISTINCT city FROM city
ORDER BY city";

populate_combobox(sql_city,
frm.getJCBCity(), frm);

            //Populates
jcbCountryID
            String sql_country_id =
"SELECT country_id FROM country
ORDER BY country_id";

populate_combobox(sql_country_id,
frm.getJCBCountryID(), frm);

            //Populates
getJCBCountry()
            String sql_ct = "SELECT
DISTINCT country FROM country ORDER
BY country";

populate_combobox(sql_ct,
frm.getJCBCountry(), frm);

        }catch (SQLException ex){

JOptionPane.showMessageDialog(frm,
ex.getMessage(),

 "ERROR",JOptionPane.ERROR_MESSAGE)
;
        }
    }
```

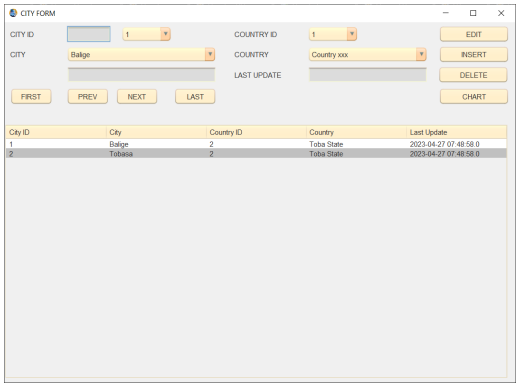


Figure 9.2 The content of joined **country** and **city** tables

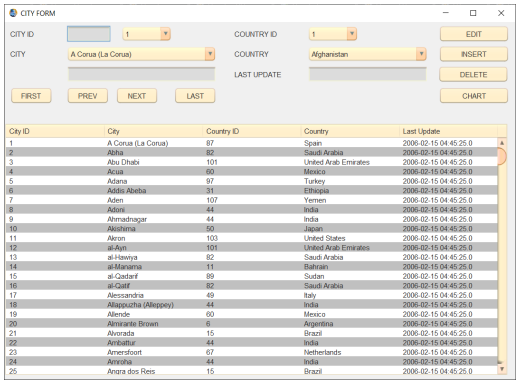


Figure 9.3 The the content of joined **country** and **city** tables in original internet displayed in **jtCity**

Step 3

In **CityForm**'s default constructor, the **City_Utils.refresh_controls(this** the controls in the **CityForm** with data from a database using the **ref City_Utils** class.

The **this.setIconImage()** method sets the icon of the **C** **this.setDefaultCloseOperation(this.HIDE_ON_CLOSE)** method sets th to hide the form instead of exiting the application when the close button is

```
1  package sakila;
2  import java.awt.Toolkit;
3  import java.awt.event.ActionEvent;
4  import java.awt.event.ActionListener;
5  import java.io.IOException;
6  import java.text.ParseException;
7  import java.util.logging.Level;
8  import java.util.logging.Logger;
9  import javax.swing.JButton;
10 import javax.swing.JComboBox;
11 import javax.swing.JLabel;
12 import javax.swing.JTable;
```

```
13 import javax.swing.JTextField;
14
15 public class CityForm extends javax.swing.JFrame {
16     public CityForm() {
17         initComponents();
18         Utility.setLookAndFeel(this);
19         City_Utils.refresh_controls(this);
20
21  this.setIconImage(Toolkit.getDefaultToolkit().getImage(getClass().
22 ;
23         this.setDefaultCloseOperation(this.HIDE_ON_CLOSE);
24     }
25     //...
26 }
```

Step 4

Run the project to see the content of joined **country** and **city** tables disp 9.2.

If you use the data from **Sakila** MySQL database available in the intern joined **country** and **city** tables displayed in **jtCity** as shown in Figure 9.3.

DISPLAYING AND NAVIGATING DATA ROW BY ROW

DISPLAYING AND NAVIGATING DATA ROW BY ROW

Step 1

In **City_Utils**, define a new method named **display_country_data()**. Thi static method in the **CityForm** class that displays country data based on t and the selected item. It takes three arguments:

- **frm**: A **CityForm** object that represents the user interface form.
- **sql**: A string that contains the SQL query to be executed.
- **item**: A generic type T that represents the selected item.

The method first opens a connection to the database using the **getC** method. It then creates a prepared statement object and sets the ite parameter using the **setObject()** method. The SQL query is execut **executeQuery()** method, and the resulting **ResultSet** object is iterated do-while loop.

Inside the loop, the **find_combo_value_selected()** method is called twice the selected items in the **jcbCountryID** and **jcbCountry** combo boxes **country_id** and **country** values in the **ResultSet**.

If there are no rows in the **ResultSet**, the method clears the selected combo boxes. If an **SQLException** occurs, an error message is displa **JOptionPane.showMessageDialog()** method.

Note that the method does not return anything, but instead updates the cc
the frm object based on the SQL query results.

```
   //Displays country data result row by row
    private static <T> void
display_country_data(CityForm frm, String sql, T
item){
        try(Connection conn = getConnection()){
            PreparedStatement ps =
conn.prepareStatement(sql);
            ps.setObject(1,item);
            ResultSet rs = ps.executeQuery();

            if (!rs.next()) {
                // no row found, clear the form
fields

frm.getJCBCountryID().setSelectedIndex(-1);

frm.getJCBCountry().setSelectedIndex(-1);
                return;
            }

            do{
                // Determines item selected from
jcbCountryID

find_combo_value_selected(frm.getJCBCountryID(),
rs.getInt("country_id"));

                // Determines item selected from
jcbCountry

find_combo_value_selected(frm.getJCBCountry(),
rs.getString("country"));
            }while(rs.next());

            rs.close();
            ps.close();
        }catch(SQLException ex){
            JOptionPane.showMessageDialog(frm,
ex.getMessage(),

 "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }
```

Step 2	Still in the same class, define another method named **jcbCountry_ha** method handles the event when the user selects an item from either the

or "Country" **JComboBox**. It first gets the selected item from the **JCor** then determines whether the **JComboBox** is the "Country ID" **JComboBox**. It then calls the **display_country_data()** method, pa **CityForm**, the SQL statement to execute, and the selected **display_country_data()** method displays the country data result row by r

```
1     public static void
2  jcbCountry_handler(CityForm frm,
3  JComboBox<String> jcb) {
4          Object item =
5  jcb.getSelectedItem();
6          String sql = "";
7          if
8  (jcb.equals(frm.getJCBCountryID()))
9  {
10             sql =
11 Query_Country.get_sql_id();
           } else if
   (jcb.equals(frm.getJCBCountry())) {
           sql =
   Query_Country.get_sql_name();
           }

           display_country_data(frm,
   sql, item);
       }
```

Step 3

In **CityForm**, double click on **jcbCountryID** and **jcbCountry** combobox their corresponding event handler as follows:

```
1     private void
2  jcbCountryActionPerformed(java.awt.event.ActionEvent
3  evt) {
4          City_Utils.jcbCountry_handler(this,
5  this.jcbCountry);
6      }
7

      private void
   jcbCountryIDActionPerformed(java.awt.event.ActionEvent
   evt) {
          City_Utils.jcbCountry_handler(this,
   this.jcbCountryID);
      }
```

These two methods are event handlers for when the user selects an "Country" and "Country ID" **JComboBoxes**, respectively. The **jcbCountry_handler()** method in the **City_Utils** class and pass in the cur of the **CityForm** and the **JComboBox** that triggered the event. This

determines the appropriate SQL query based on which **JComboBox** was displays the data for the selected item in the appropriate fields in the form

Step 4

Run the project. Choose one of items in **jcbCountryID** and/or **jcbCountry** to see row by row the content of **country** table as shown in Figure 9.4.

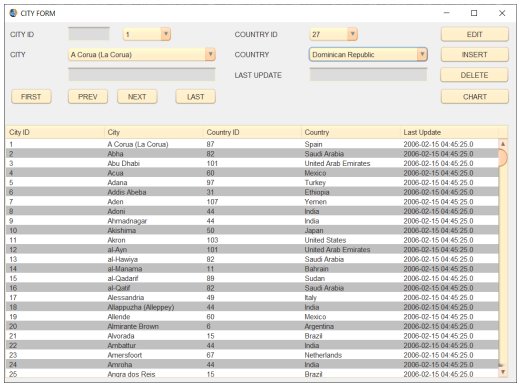


Figure 9.4 Displaying row by row the content of **country** tabl

Step 5

In **City_Utils** class, define two new methods named **clear_cor display_city_data()**. The **display_city_data()** method takes three ar instance of **CityForm** class named frm, a String variable named **sql**, a variable named **item** of type T.

Inside the method, it first creates a **PreparedStatement** object u argument, sets the value of the first parameter as item, and executes the row is found in the result set, it clears the form fields by calling the **clea** method and returns. Otherwise, it loops through each row of the result se corresponding values to the relevant form fields. It also **find_combo_value_selected()** method to determine the selected it **jcbCityID**, **jcbCity**, **jcbCountryID**, and **jcbCountry** combo boxes. Finally result set and the prepared statement. If an exception occurs, it displ message dialog box using the **JOptionPane.showMessageDialog** method

```
1     private static void clear_controls(CityForm frm){
2         frm.getJTFCityID().setText("");
3         frm.getJTFCity().setText("");
4         frm.getJTFLastUpdate().setText("");
5         frm.getJCBCountryID().setSelectedIndex(-1);
6         frm.getJCBCountry().setSelectedIndex(-1);
7     }
8
9    //Displays city data result row by row
10    private static <T> void display_city_data(CityForm frm, String
11 {
12        try(Connection conn = getConnection()){
13            PreparedStatement ps = conn.prepareStatement(sql);
14            ps.setObject(1,item);
```

```
            ResultSet rs = ps.executeQuery();

            if (!rs.next()) {
                // no row found, clear the form fields
                clear_controls(frm);
                return;
            }

            do{

frm.getJTFCityID().setText(String.valueOf(rs.getInt("city_id")));
                frm.getJTFCity().setText(rs.getString("city"));

frm.getJTFLastUpdate().setText(String.valueOf(rs.getTimestamp("last

                // Determines item selected from jcbCityID
                find_combo_value_selected(frm.getJCBCityID(),
rs.getInt("city_id"));

                // Determines item selected from jcbCity
                find_combo_value_selected(frm.getJCBCity(),
rs.getString("city"));

                // Determines item selected from jcbCountryID
                find_combo_value_selected(frm.getJCBCountryID(),
rs.getInt("country_id"));

                // Determines item selected from jcbCountry
                find_combo_value_selected(frm.getJCBCountry(),
rs.getString("country"));
            }while(rs.next());

            rs.close();
            ps.close();
        }catch(SQLException ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }
```

Step 6

In the same class, define another method named **jcbCity_handler()** **CityForm** object and a **JComboBox** object as parameters. It first gets the from the **JComboBox** and then sets the SQL query string based on **JComboBox** is **frm.getJCBCityID()** or **frm.getJCBCity()**. The SQL qu constructed using the **Query_City.get_sql_city_joint()** method, which ret query string to join the city and country tables. The WHERE clause is then query string to search for the selected city either by **city_id** or by **city** **display_city_data()** method is called with the **CityForm** object, the con query string, and the selected item as arguments to display the city data o

```
    public static void
jcbCity_handler(CityForm frm,
JComboBox<String> jcb) {
        Object item =
jcb.getSelectedItem();
        String sql = "";
        if
(jcb.equals(frm.getJCBCityID())) {
            sql =
Query_City.get_sql_city_joint() + "
WHERE city_id = ?";
        } else if
(jcb.equals(frm.getJCBCity())) {
            sql =
Query_City.get_sql_city_joint() + "
WHERE city = ?";
        }

        display_city_data(frm, sql,
item);
    }
```

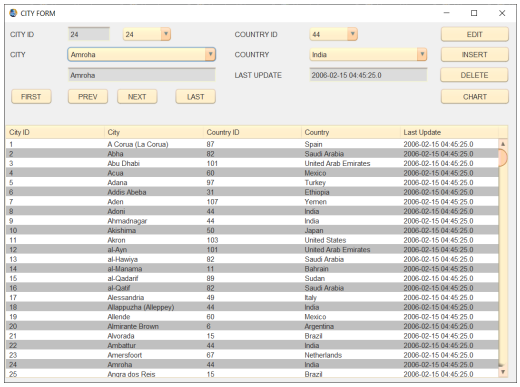


Figure 9.5 Displaying row by row the content of joined **city** and **coun**

Step 7

In **CityForm**, double click on **jcbCityID** and **jcbCity** comboboxes and corresponding event handler as follows:

```
    private void
jcbCityIDActionPerformed(java.awt.event.ActionEvent
evt) {
        City_Utils.jcbCity_handler(this,
this.jcbCityID);
    }

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

```
        City_Utils.jcbCity_handler(this,
this.jcbCity);
    }
```

These are event handling methods for the two city **JComboBoxes** (**jc** **jcbCity**). They call the **jcbCity_handler()** method in **City_Utils** class pas current **CityForm** instance) and the respective **JComboBox** as para method retrieves the selected item from the **JComboBox**, determines th SQL query based on the **JComboBox**, and calls the **display_city_da** passing the **CityForm** instance, the SQL query, and the selected item as p

Step 8

Run the project. Choose one of items in **jcbCityID** and/or **jcbCity** combo row by row the content of joined **city** and **country** tables as shown in Figu

Step 9

Define four navigating methods in **City_Utils** class. These method implement functionality for navigating through the rows of data in the city

1. **show_first_row()** method sets the item of the **jcbCityID** combo b index and displays the corresponding city data using the **displa** method.
2. **show_last_row()** method sets the item of the **jcbCityID** combo b index and displays the corresponding city data using the **displa** method.
3. **show_prev_row()** method decrements the current index and if i the first index, it sets the current index to the first index. It then at the current index of the **jcbCityID** combo box and corresponding city data using the **display_city_data()** method.
4. **show_next_row()** method increments the current index and if it is the last index, it sets the current index to the last index. It then ge the current index of the **jcbCityID** combo box and displays the c city data using the **display_city_data()** method.

```
1     public static void show_first_row(CityForm frm){
2         String item =
3 String.valueOf(frm.getJCBCityID().getItemAt(FIRST_INDEX));
4         display_city_data(frm, SQL_ID, item);
5         currentIndex = FIRST_INDEX;
6     }
7
8     public static void show_last_row(CityForm frm){
9         int endIndex = frm.getJCBCityID().getItemCount() -
10 1;
11        String item =
12 String.valueOf(frm.getJCBCityID().getItemAt(endIndex));
13        display_city_data(frm, SQL_ID, item);
14        currentIndex = endIndex;
15    }
16
```

```
    public static void show_prev_row(CityForm frm){
        currentIndex--;
        if(currentIndex < FIRST_INDEX){
            currentIndex = FIRST_INDEX;
            return;
        }
        String item =
String.valueOf(frm.getJCBCityID().getItemAt(currentIndex));
        display_city_data(frm, SQL_ID, item);
    }

    public static void show_next_row(CityForm frm){
        int endIndex = frm.getJCBCityID().getItemCount() -
1;
        currentIndex++;
        if(currentIndex > endIndex){
            currentIndex = endIndex;
            return;
        }
        String item =
String.valueOf(frm.getJCBCityID().getItemAt(currentIndex));
        display_city_data(frm, SQL_ID, item);
    }
```

Step 10

Then in **CityForm**, double click on each navigation buttons to corresponding event handler:

```
    private void
jbLastActionPerformed(java.awt.event.ActionEvent
evt) {
        City_Utils.show_last_row(this);
    }

    private void
jbNextActionPerformed(java.awt.event.ActionEvent
evt) {
        City_Utils.show_next_row(this);
    }

    private void
jbPrevActionPerformed(java.awt.event.ActionEvent
evt) {
        City_Utils.show_prev_row(this);
    }

    private void
jbFirstActionPerformed(java.awt.event.ActionEvent
evt) {
        City_Utils.show_first_row(this);
    }
```

These methods are associated with the "First", "Previous", "Next", and " respectively. When clicked, each button invokes its corresponding meth the first, previous, next, or last row of data in the **CityForm**.

Step 11

Run the project. Click on one or more navigation buttons to see the resul Figure 9.6.

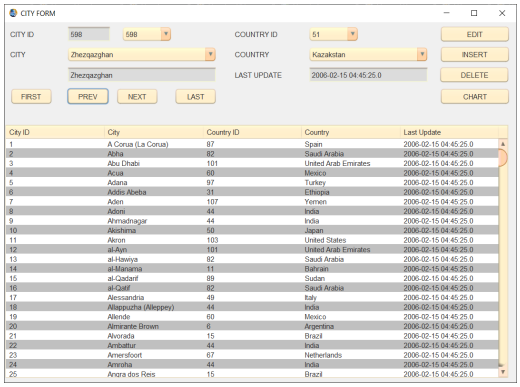


Figure 9.6 User clicks on one or more navigation buttons on city

Step 12

Define **mouse_pressed_handler()** method in **City_Utils** class. This me the mouse click event on the **JTable** component of the **CityForm**. It gets row index from the **JTable**, and if there is no row selected, it displays a message to prompt the user to select a row. If a row is selected, it gets th the first column of the selected row, and then calls the **display_city_data** display the data of the selected city on the **CityForm**. The city ID is argument to the SQL query defined by the **SQL_ID** constant. If ther exception, it catches it and displays an error message with the exception d

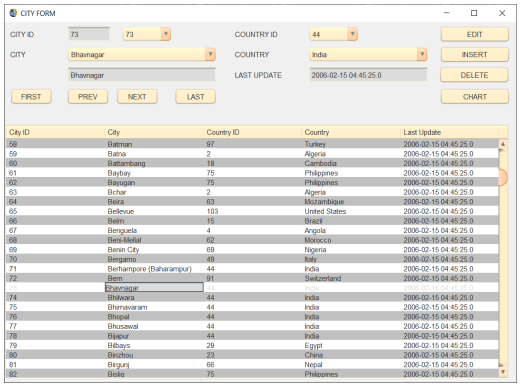


Figure 9.7 User double-clicks on any row in **jtCity**

```
1
2
3
```

```
    public static void mouse_pressed_handler(CityForm frm) {
        Objects.requireNonNull(frm, "frm must not be null");
```

```
        int selectedIndex = frm.getJTCity().getSelectedRow();
        if (selectedIndex == -1) {
            JOptionPane.showMessageDialog(frm, "Please select a row to view its data.",
                "No row selected",
JOptionPane.INFORMATION_MESSAGE);
            return;
        }

        try (Connection conn = getConnection()) {
            String id =
String.valueOf(frm.getJTCity().getModel().getValueAt(selectedIndex,
0));

            // Displays city data
            display_city_data(frm, SQL_ID, id);

        } catch (SQLException ex) {

Logger.getLogger(CityForm.class.getName()).log(Level.SEVERE, "Error displaying city data", ex);
            String message = "Error displaying city data: " +
ex.getMessage();
            String stackTrace =
Arrays.toString(ex.getStackTrace());
            JOptionPane.showMessageDialog(frm, message + "\n\n" +
stackTrace, "ERROR", JOptionPane.ERROR_MESSAGE);
        }
    }
```

Step 13

Right click on **jtCity**. Then, choose **Events > Mouse > mousePresse** event handler:

```
    private void
jtCityMousePressed(java.awt.event.MouseEvent
evt) {

 City_Utils.mouse_pressed_handler(this);
    }
```

Step 14

Run the project. Double click on any row in **jtCity** table. You will see the c row of joined **country** and **city** tables displayed in textfields and combobo in Figure 9.7.

UPDATING RECORD
UPDATING RECORD

Step 1

In **City_Utils** class, define a new method named **update_row_by_city_id()**. This method updates a row in the **city** table of the database based on the given **city_id**. It first retrieves the row using the **city_id** parameter to check if it exists. If it does, it creates a **City** object with the updated data, and uses a prepared statement to update the row in the city table.

If the **city_id** is not found, an error message is displayed using a **JOptionPane** dialog.

If there is a SQL or a **NumberFormatException** exception during the update process, the method logs the exception and displays an error message using a **JOptionPane** dialog.

The method takes the following parameters:

1. **frm**: the **CityForm** object that holds the GUI elements
2. **city_id**: the ID of the city to update
3. **city**: the updated name of the city
4. **country_id**: the updated ID of the country that the city belongs to

The method throws a SQLException if there is a database error during the update process, and a **NumberFormatException** if there is an error parsing the **city_id** parameter to an integer.

```
    //Updates row of data in city tabel by city_id
    public static void update_row_by_city_id(CityForm frm,
int city_id, String city, int country_id) throws
SQLException{
        Connection conn = getConnection();
        ResultSet rs = null;
        String query_id = "SELECT city_id FROM city WHERE
city_id = ?";
        String update_query = """
            UPDATE city SET city = ?, country_id = ? WHERE
city_id = ?""";
        try(PreparedStatement idPs =
conn.prepareStatement(query_id,

ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_READ_ONLY);
        PreparedStatement updatePS =
conn.prepareStatement(update_query,
                ResultSet.TYPE_SCROLL_SENSITIVE,
ResultSet.CONCUR_UPDATABLE))
        {
```

```
21              idPs.setInt(1,city_id);
22              if(!idPs.execute()){
23                  String message = "Can't find city_id " +
24  city_id;
25
26                  JOptionPane.showMessageDialog(frm, message,
27                      "ERROR",JOptionPane.ERROR_MESSAGE);
28              } else{
29                  rs = idPs.getResultSet();
30                  rs.next();
31
32                  //Creates a City object using four-params
33  constructor
34                  City obj = new City(city_id, city,
35  country_id, new Timestamp(System.currentTimeMillis()));
36                  updatePS.setString(1, obj.getCity());
37                  updatePS.setInt(2, obj.getCountryID());
38                  updatePS.setInt(3, obj.getCityID());
39
40                  updatePS.executeUpdate();
41                  rs.close();
42                  updatePS.close();
43                  idPs.close();
44                  conn.close();
45              }
46          }catch(SQLException ex){
47
48  Logger.getLogger(CityForm.class.getName()).log(Level.SEVERE,
49  "Error updating city data", ex);
50              String message = "Error updating city data: " +
55  ex.getMessage();
56              String stackTrace =
    Arrays.toString(ex.getStackTrace());
                JOptionPane.showMessageDialog(null, message +
    "\n\n" + stackTrace, "ERROR", JOptionPane.ERROR_MESSAGE);
            }catch(java.lang.NumberFormatException ex){

    Logger.getLogger(CityForm.class.getName()).log(Level.SEVERE,
    "Invalid Input", ex);
                String message = "Invalid Input: " +
    ex.getMessage();
                String stackTrace =
    Arrays.toString(ex.getStackTrace());
                JOptionPane.showMessageDialog(null, message +
    "\n\n" + stackTrace, "ERROR", JOptionPane.ERROR_MESSAGE);
            }
        }
```

Step 2	Then in the same class, define a new method **read_inputs()**. This method reads the user inputs from the **CityForm** and validates

them. It returns a **HashMap** containing the validated input data.

The method first initializes an empty **HashMap** to hold the input data. It then retrieves the city ID, country ID, and city name from the relevant components on the **CityForm**.

Next, the method validates the country ID and city ID by parsing them as integers and checking that they are positive values. If either value is invalid, an exception is thrown with an appropriate error message, and the exception is caught and re-thrown to the calling method.

Finally, the method checks that the city name is not empty. If it is empty, an exception is thrown with an error message, and the exception is caught and re-thrown to the calling method.

If all input values are valid, they are added to the **HashMap** and returned to the calling method.

```
1   private static HashMap<String, String>
2   read_inputs(CityForm frm) {
3           HashMap<String, String> input_data = new
4   HashMap<>();
5           String city_id =
6   String.valueOf(frm.getJCBCityID().getSelectedItem());
7           String country_id =
8   String.valueOf(frm.getJCBCountryID().getSelectedItem());
9           String city = frm.getJTFCity().getText();
10
11          // Validate user input
12          int country_id_int = 0;
13          try {
14              country_id_int =
15  Integer.parseInt(country_id);
16              if (country_id_int <= 0) {
17                  throw new
18  IllegalArgumentException("Country ID cannot be negative
19  or zero");
20              }
21          } catch (NumberFormatException ex) {
22              JOptionPane.showMessageDialog(frm, "Invalid
23  Country ID: " + country_id,
24              "Error", JOptionPane.ERROR_MESSAGE);
25              throw ex;
26          } catch (IllegalArgumentException ex) {
27              JOptionPane.showMessageDialog(frm,
28  ex.getMessage(),
29              "Error", JOptionPane.ERROR_MESSAGE);
30              throw ex;
31          }
32
```

```
        int city_id_int = 0;
        try {
            city_id_int = Integer.parseInt(city_id);
            if (city_id_int <= 0) {
                throw new IllegalArgumentException("City ID cannot
be negative or zero");
            }
        } catch (NumberFormatException ex) {
            JOptionPane.showMessageDialog(frm, "Invalid
City ID: " + city_id,
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        } catch (IllegalArgumentException ex) {
            JOptionPane.showMessageDialog(frm,
ex.getMessage(),
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        }

        if (city == null || city.isEmpty()) {
            JOptionPane.showMessageDialog(frm, "City
cannot be empty",
            "Error", JOptionPane.ERROR_MESSAGE);
            throw new IllegalArgumentException("City
cannot be empty");
        }

        input_data.put("city_id", city_id);
        input_data.put("country_id", country_id);
        input_data.put("city", city);

        return input_data;
    }
```

Step 3

Still in the same class, define another method named **edit_actual()**. It is responsible for updating the city data in the database based on the user input. It first reads the input data using the read_inputs method and then calls the **update_row_by_city_id()** method to update the data in the database. If an **SQLException** is caught, it displays an error message to the user. Finally, it refreshes all the controls on the form using the **refresh_controls()** method.

```
    private static void edit_actual(CityForm
frm){
        try{
            HashMap<String, String> input_data
= read_inputs(frm);
            int city_id =
Integer.parseInt(input_data.get("city_id"));
```

```
            int country_id =
Integer.parseInt(input_data.get("country_id"));
            String city =
input_data.get("city");

            update_row_by_city_id(frm, city_id,
city, country_id);

            //Refreshes all objects on form
            refresh_controls(frm);

        }catch(SQLException ex){
            JOptionPane.showMessageDialog(frm,
ex.getMessage(),

 "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }
```

Step 4

Lastly, define two new methods named **enable_controls()** and **edit_handler()**. The **edit_handler()** method is responsible for handling the edit button's action on the form. When the user clicks the edit button, it checks whether the text on the button is "EDIT" or "CONFIRM." If the text is "EDIT," it changes the text to "CONFIRM" and disables all the form controls, allowing the user to edit the city information. If the text is "CONFIRM," it changes the text back to "EDIT," calls the **edit_actual()** method to actually update the city information in the database, enables all the form controls again, and refreshes the form controls.

```
    private static void
enable_controls(boolean state, CityForm frm)
{
        frm.getJBFirst().setEnabled(state);
        frm.getJBPrev().setEnabled(state);
        frm.getJBNext().setEnabled(state);
        frm.getJBLast().setEnabled(state);
        frm.getJBInsert().setEnabled(state);
        frm.getJBDelete().setEnabled(state);

frm.getJTFCityID().setEnabled(state);
    }

    public static void edit_handler(CityForm
frm){

 if(frm.getJBEdit().getText().equals("EDIT")
){

frm.getJBEdit().setText("CONFIRM");
```

```
21
22              // Disables controls
23              enable_controls(false, frm);
24          }
25
26          else {
27              frm.getJBEdit().setText("EDIT");
28
                // Actual editing
                edit_actual(frm);

                //Enables controls
                enable_controls(true, frm);
            }
        }
```

Step 5

Run the project. Choose **city_id** using **jcbCityID** or **jcbCity** combobox. Or, you can choose one of rows in **jtCity** (in this case, **city_id = 600**). Then, click on EDIT button as shown in Figure 9.8.

Edit the city name and/or choose the country. Then, click on CONFIRM button. The edited row had been saved into **city** table as shown in Figure 9.9.

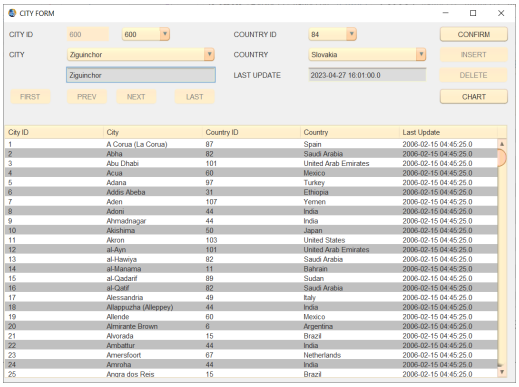


Figure 9.8 The city form is in editing state

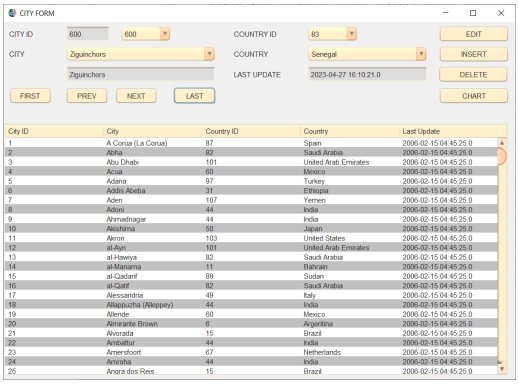


Figure 9.9 The edited row had been saved into database

UPDATING RECORD DIRECTLY ON JTABLE
UPDATING RECORD DIRECTLY ON JTABLE

Step 1

In **City_Utility** class, define a new method nam **edit_database_from_jtable()**. This method is used to handle updates ma to the **JTable** in the **CityForm**. When a cell in the **JTable** is updated **TableModelEvent** is triggered, and this method is called.
The method first checks if the event type is UPDATE, which indicates tha cell has been updated. It then gets the row, column, and value of the upda cell using the **TableModel** interface. The method then calls **update_row_by_city_id()** method to update the corresponding row in database.

If the update is successful, the method calls the **refresh_controls()** meth to update all the objects on the form with the latest data from the database

If an exception occurs during the update, the method catches it and displ an error message using the **JOptionPane.showMessageDialog()** meth The error message is specific to the type of exception caught, and exception message is included in the error message.

```
1   public static void
2  edit_database_from_jtable(TableModelEvent e, CityForm frm){
3          if (e.getType() == TableModelEvent.UPDATE) {
4              int row = e.getFirstRow();
5              TableModel model = (TableModel)e.getSource();
6              int city_id =
7  Integer.parseInt(String.valueOf(model.getValueAt(row, 0)));
8              String city = (String) model.getValueAt(row, 1);
9              int country_id =
10 Integer.parseInt(String.valueOf(model.getValueAt(row, 2)));
11
12             try{
13                 update_row_by_city_id(frm, city_id, city,
14 country_id);
15
16                 //Refreshes all objects on form
17                 refresh_controls(frm);
18
19             } catch
20 (SQLIntegrityConstraintViolationException ex) {
21
22 Logger.getLogger(CityForm.class.getName()).log(Level.SEVERE,
23 "Duplicate entry", ex);
24
```

```
                JOptionPane.showMessageDialog(frm, "Error:
Duplicate entry\n" + ex.getMessage());
            } catch (SQLSyntaxErrorException ex) {

Logger.getLogger(CityForm.class.getName()).log(Level.SEVERE,
"Invalid SQL syntax", ex);
                JOptionPane.showMessageDialog(frm, "Error:
Invalid SQL syntax\n" + ex.getMessage());
            } catch (SQLException ex) {

Logger.getLogger(CityForm.class.getName()).log(Level.SEVERE,
"Database error", ex);
                JOptionPane.showMessageDialog(frm, "Error:
Database error\n" + ex.getMessage());
            }
        }
    }
```

Step 2

Create a new public class named **CityTableModelListener**. It impleme the **TableModelListener** interface. It takes a **JTable** and a **CityForm** parameters in its constructor.

The **tableChanged()** method is called whenever the model data changes the table. Inside this method, it calls the **edit_database_from_jtabl** method in the **City_Utils** class, passing the event and the **CityForm** parameters. This method handles updating the database based on changes made in the table.

The code also checks if there is a cell editor in progress and stops it if th is. This ensures that any changes made to the cell are saved before update is processed.

```
package sakila;
import
javax.swing.event.TableModelEvent;
import
javax.swing.event.TableModelListener;
import javax.swing.JTable;

public class CityTableModelListener
implements TableModelListener {
    private final JTable jt;
    private final CityForm frm;

    public
CityTableModelListener(JTable jt,
CityForm frm) {
        this.jt = jt;
        this.frm = frm;
```

```
    }

    @Override
    public void
tableChanged(TableModelEvent e) {

City_Utils.edit_database_from_jtable(e,
frm);

        if (jt.getCellEditor() != null)
{

jt.getCellEditor().stopCellEditing();
        }
    }
}
```

Step 3

Right click on **jtCity**. Then, choose **Events > Mouse > mouseClick** Define its event handler:

```
    private void jtCityMouseClicked(java.awt.event.MouseEvent evt) {
        // instantiate CityTableModelListener and add it as a listen
to the table model
        CityTableModelListener tableModelListener = new
CityTableModelListener(this.getJTCity(), this);

 this.getJTCity().getModel().addTableModelListener(tableModelListene
;
    }
```

It is an event handler for the mouse click event on the **JTable** named **jtCi** When the table is clicked, it creates a new instance **CityTableModelListener** and passes the **JTable** and the **CityForm** parameters. It then adds the new **CityTableModelListener** instance a listener to the table model of **jtCity**. This is done to detect any changes ma to the data in the **JTable**, so that the changes can be reflected in underlying database.

After the **edit_database_from_jtable()** method is called, **stopCellEditing()** method is called on any active cell editor for the tab This ensures that any changes made by the user are committed before listener updates the database.

Step 4

Run the project. Click on any cell in second column in any row in **jtCity** t you want to edit. Then, change it. Then, click any where outside corresponding cell. The edited data had been saved into database.

INSERTING NEW RECORD
INSERTING NEW RECORD

Step 1

In **City_Utils** class, define a method named **insert_row()**. It inserts a new row into the **city** table. It takes a **CityForm** object as an argument, which is used to read input data from the user interface.

The method first reads the city name and country ID from the input data. It then constructs an SQL insert statement with placeholders for these values.

Next, it creates a new **City** object with the input data and a timestamp representing the current time. It sets the values of the first two placeholders in the SQL statement to the city name and country ID, respectively.

Finally, it executes the SQL insert statement using a JDBC **PreparedStatement** object, and catches any **SQLException** that may occur. If an exception is caught, it logs the error and displays an error message to the user.

```
1   //Inserts new row into city table
2   private static void insert_row(CityForm frm) throws
3   SQLException{
4           HashMap<String, String> input_data =
5   read_inputs(frm);
6           String city = input_data.get("city");
7           int country_id =
8   Integer.parseInt(input_data.get("country_id"));
9
10          // SQL insert statement
11          String sql = """
12                  INSERT INTO city(city, country_id) VALUES(?,
13  ?)""";
14
15          try(Connection conn = getConnection();
16           PreparedStatement pstmt =
17  conn.prepareStatement(sql)){
18
19              //Creates a City object three-params constructor
20              City obj = new City(city, country_id, new
21  Timestamp(System.currentTimeMillis()));
22              pstmt.setString(1,obj.getCity());
23              pstmt.setInt(2,obj.getCountryID());
24
25              //Executes the sql insert statement
```

```
            pstmt.executeUpdate();
        } catch (SQLException ex) {

Logger.getLogger(CityForm.class.getName()).log(Level.SEVERE,
"Database error", ex);
            JOptionPane.showMessageDialog(frm, "Error:
Database error\n" + ex.getMessage());
        }
    }
```

Step 2

Still in **City_Utils.java**, define **insert_actual()** and **insert_handler()** methods. The **insert_handler()** method is called when the user clicks a button to insert a new row into the database. If the button text is "INSERT", it changes the button text to "CONFIRM" and disables the "jbEdit" button and some controls on the form. It also clears the values of those controls. If the button text is "CONFIRM", it calls the **insert_actual()** method to actually insert the new row into the database, and then re-enables the "jbEdit" button and form controls.

The **insert_actual()** method calls the **insert_row()** method to insert the new row into the database, and then refreshes the table and comboboxes using the **refresh_controls()** method. If an **SQLException** is caught, it displays an error message to the user.

```
1
2
3
```

```
4       private static void insert_actual(CityForm
5   frm){
6           try{
7               insert_row(frm);
8
9               //Refreshes table and comboboxes
10              refresh_controls(frm);
11
12          }catch(SQLException ex){
13              JOptionPane.showMessageDialog(frm,
14  ex.getMessage(),
15
16   "ERROR",JOptionPane.ERROR_MESSAGE);
17          }
18      }
19
20      public static void insert_handler(CityForm
21  frm){
22
23   if(frm.getJBInsert().getText().equals("INSERT")
24  ){
25
26  frm.getJBInsert().setText("CONFIRM");
27
28              //Disables jbEdit
29              frm.getJBEdit().setEnabled(false);
30
31              // Disables controls
32              enable_controls(false, frm);
33
34  frm.getJCBCityID().setEnabled(false);
35              frm.getJCBCity().setEnabled(false);
36
37              // Clears controls
38              clear_controls(frm);
39
40              // Enables
41              frm.getJBInsert().setEnabled(true);
42          }
43
44          else {
45              frm.getJBInsert().setText("INSERT");
46
47              // Actual insertion
                insert_actual(frm);

                //Enables jbEdit
                frm.getJBEdit().setEnabled(true);

                //Enables controls
                enable_controls(true, frm);
                frm.getJCBCityID().setEnabled(true);
                frm.getJCBCity().setEnabled(true);
```

```
        }
    }
```

Step 3

In **CityForm.java**, dou
create its event listener

```
    private void
jbInsertActionPerfo
evt) {
        City_Utils.
    }
```

Step 4

Run the project. Click
the state of city form
shown in Figure 9.10.

Then, type a city nam
click CONFIRM button
table as shown in Figur

Figure 9.10 When us
city form will

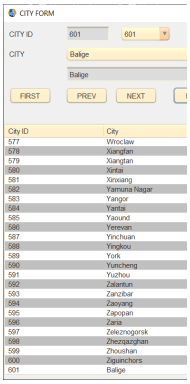


Figure 9.11 The new

DELETING RECORD
DELETING RECORD

Step 1

Then in **City_Utils** class, define **delete_handler()** method. It is used to ha deleting a row from the **city** table. It takes a **CityForm** object as an argu which is used to access the user interface.

The method first retrieves the selected city ID from the **jcbCityID** comb on the form. It then displays a confirmation dialog box to the user, aski they are sure they want to delete the row with the selected city ID.

If the user clicks the "YES" button in the dialog box, the method construc SQL DELETE statement with a placeholder for the city ID. It then uses a **PreparedStatement** object to execute the SQL statement with the sel city ID, and catches any **SQLException** that may occur.

If the delete operation is successful, the method calls the **refresh_contr** method to refresh the table and comboboxes on the form. If an **SQLExcep** is caught, it displays an error message to the user.

```
1     public static void delete_handler(CityForm frm){
2         int dialogButton = JOptionPane.YES_NO_OPTION;
3         int city_id =
4 Integer.parseInt(String.valueOf(frm.getJCBCityID().getSelectedItem(
5 
6         String message = String.format("Are you sure you want to de
7 the row City ID: %d)", city_id);
8         int answer = JOptionPane.showConfirmDialog(frm, message,
9 "DELETING ROW OF DATA", dialogButton);
10 
11        if(answer == JOptionPane.YES_OPTION){
12            String query = """
13                DELETE FROM city WHERE city_id = ?""";
14            try(Connection conn = getConnection();
15                PreparedStatement ps = conn.prepareStatement(query)
16                // Use PreparedStatement to avoid SQL injection att
17                ps.setInt(1, city_id);
18                ps.executeUpdate();
19 
20                // Refresh table and comboboxes
21                refresh_controls(frm);
22 
23            } catch (SQLException ex){
24                JOptionPane.showMessageDialog(frm, ex.getMessage(),
```

```
                    "ERROR",JOptionPane.ERROR_MESSAGE);
                }
            }
        }
```

Step 2

In **CityForm.java**, double click on DELETE button to generate its listener:

```
    private void jbDeleteActionPerformed(java.awt.event.ActionEvent evt) {
        City_Utils.delete_handler(this);
    }
```

Step 3

Run the project. Choose **city_id** using **jcbCityID** or **jcbCity** combobox. Click on DELETE button. The corresponding row of data had been deleted database.

PLOTTING CHART
PLOTTING CHART

Step 1

In **CityForm**, add two **JPanels** and set their corresponding **Variable Name** as **jPanel1** and **jPanel2**. Then, add getter method for each object as follows:

```
    //Getter method for jPanel1
    public JPanel getJPanel1(){
        return this.jPanel1;
    }

    //Getter method for jPanel2
    public JPanel getJPanel2(){
        return this.jPanel2;
    }
```

Step 2

In **City_Utils** class, define two new methods. These are two Java methods that draw charts based on data retrieved from the database. Both methods take two arguments: a **CityForm** object

and a **JPanel** object. The **JPanel** object is where the chart will be drawn.

The first method, **draw_pie_chart_city_by_country()**, draws a pie chart that shows the top 10 city distributions by country. It first sets the preferred size of the **JPanel** to the current size of the panel. It then calls the **create_pie_dataset()** method, which takes an SQL query string and column names as arguments and returns a **DefaultPieDataset** object. This method is not shown in the code snippet, but it is likely to contain the SQL query string that retrieves the data from the database.

Once the dataset is created, the method calls the **draw_piechart_with_dataset()** method, which takes the **CityForm** object, the **JPanel** object, the dataset, and a chart title as arguments.

The second method, **draw_bar_chart_city_by_country()**, draws a bar chart that shows the top 10 city distributions by country. It first sets the preferred size of the **JPanel** to the current size of the panel. It then calls the **create_bar_dataset()** method, which takes an SQL query string and column names as arguments and returns a **DefaultCategoryDataset** object.

Once the dataset is created, the method calls the **draw_barchart_with_dataset()** method, which takes the **CityForm** object, the **JPanel** object, the dataset, a chart title, an x-axis label, and a y-axis label as arguments.

```
1   private static void
2  draw_pie_chart_city_by_country(CityForm frm, JPanel jp){
3          jp.setPreferredSize(new Dimension(jp.getWidth(),
4  jp.getHeight()));
5          DefaultPieDataset dataset =
6  create_pie_dataset(Query_City.get_sql_city_country_dist(),
7  "Number", "co.country");
8
9          //Draws piechart city distribution by country
10         draw_piechart_with_dataset(frm, jp, dataset, "TOP
11 10 CITY DISTRIBUTION BY COUNTRY");
12     }
13
14     private static void
15 draw_bar_chart_city_by_country(CityForm frm, JPanel jp){
16         jp.setPreferredSize(new Dimension(jp.getWidth(),
17 jp.getHeight()));
18
19         DefaultCategoryDataset dataset =
20 create_bar_dataset(Query_City.get_sql_city_country_dist(),
```

```
"Number", "co.country");

        //Draws barchart city distribution by country
        draw_barchart_with_dataset(frm, jp, dataset, "THE
10 CITY DISTRIBUTION BY COUNTRY", "COUNTRY", "NUMBER");
    }
```

Step 3

In **City_Utils** class, define a new method named **jbchart_handler()**.

```
    public static void
jbchart_handler(CityForm frm){
        //Draws piechart city
distribution by country

draw_pie_chart_city_by_country(frm,
frm.getJPanel1());

        //Draws barchart city
distribution by country

draw_bar_chart_city_by_country(frm,
frm.getJPanel2());
    }
```

Step 4

In **CityForm**, double click on **jbChart** button to define its event listener:

```
    private void
jbChartActionPerformed(java.awt.event.ActionEvent
evt) {
        City_Utils.jbchart_handler(this);
    }
```

Step 5

Run the project. Click on CHART button on the form. You will see the two charts displayed on the panels as shown in Figure 9.12.

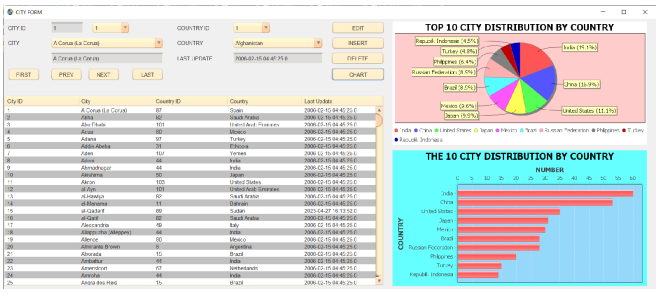


Figure 9.12 The top 10 city distribution by by country

This is the full version of **City_Utils.java**:

```
package sakila;
import java.awt.Dimension;
import java.util.logging.Level;
import java.util.logging.Logger;
import java.sql.*;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashMap;
import java.util.Objects;
import javax.swing.JComboBox;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.event.TableModelEvent;
import javax.swing.table.DefaultTableModel;
import javax.swing.table.TableModel;
import org.jfree.data.category.DefaultCategoryDataset;
import org.jfree.data.general.DefaultPieDataset;
import static sakila.Utility.create_bar_dataset;
import static sakila.Utility.draw_barchart_with_dataset;

public class City_Utils extends Utility{
    public static final int FIRST_INDEX = 0;
    public static final int INVALID_INDEX = -1;

    private static int currentIndex = FIRST_INDEX;
    private static final String SQL_ID = Query_City.get_sql_city_joint() + "
WHERE city_id = ?";

    //Creates city table
    public static void create_city_table() {
        try (Connection conn = getConnection()) {
            Statement stmt = conn.createStatement();
            stmt.addBatch(Query_City.get_sql_city());
            stmt.executeBatch();

            String message = String.format("Successfully creates city table")
            JOptionPane.showMessageDialog(null, message,
            "INFORMATION",JOptionPane.INFORMATION_MESSAGE);

        } catch (SQLException ex) {
            JOptionPane.showMessageDialog(null, ex.getMessage(),
                "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    //Populates city table with some rows of data
```

```
    public static void populate_city_table(){
        try(Connection conn = getConnection()){
            String sql = """
                INSERT INTO city(city_id, city, country_id, last_update)
                VALUES(?, ?, ?, ?)""";

            //Creates a new City class with default constructor
            PreparedStatement ps1 = conn.prepareStatement(sql);
            City obj1 = new City();
            ps1.setInt(1,obj1.getCityID());
            ps1.setString(2,obj1.getCity());
            ps1.setInt(3,obj1.getCountryID());
            ps1.setTimestamp(4,obj1.getLastUpdate());

            // Creates a new City class with four-params constructor
            PreparedStatement ps2 = conn.prepareStatement(sql);
            City obj2 = new City(2, "Tobasa", 2, new
Timestamp(System.currentTimeMillis()));
            ps2.setInt(1,obj2.getCityID());
            ps2.setString(2,obj2.getCity());
            ps2.setInt(3,obj2.getCountryID());
            ps2.setTimestamp(4,obj2.getLastUpdate());

            ps1.executeUpdate();
            ps2.executeUpdate();

        }catch(SQLException ex){
            JOptionPane.showMessageDialog(null, ex.getMessage(),
                "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    //Reads the content of joined city and country tables
    public static void read_city_table(){
        try(Connection conn = getConnection()){
            Statement stmt = conn.createStatement();
            ResultSet rs = stmt.executeQuery(Query_City.get_sql_city_joint())

            while(rs.next()){
                int cty_id = rs.getInt("city_id");
                String city = rs.getString("city");
                int ct_id = rs.getInt("country_id");
                String country = rs.getString("country");
                Timestamp lu = rs.getTimestamp("last_update");

                //Creates a City object using five-params constructor
                City obj = new City(cty_id, city, ct_id, country, lu);
                System.out.println(obj);
            }
            rs.close();
            stmt.close();

        }catch(SQLException ex){
```

```
            JOptionPane.showMessageDialog(null, ex.getMessage(),
                "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    private static ArrayList<City> get_city_list(CityForm frm, String sql,
String item){
        ArrayList<City> list = new ArrayList<>();
        Connection conv = null;

        try(Connection conn = getConnection();
            PreparedStatement ps = conn.prepareStatement(sql)){
            if (item.equalsIgnoreCase("none")==false) {
                 ps.setString(1,item);
            }
            ResultSet rs = ps.executeQuery();

            City obj;
            while(rs.next()){
                //Using three-params constructor
                obj = new City(rs.getInt("city_id"),
                    rs.getString("city"),
                    rs.getInt("country_id"),
                    rs.getString("country"),
                    rs.getTimestamp("last_update"));

                list.add(obj);
            }
        }catch (SQLException ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
                "ERROR",JOptionPane.ERROR_MESSAGE);
        }
        return list;
    }

    private static void show_table_city(CityForm frm, ArrayList<City> list)
throws SQLException{
        DefaultTableModel model = new DefaultTableModel(0,0);

        String header[] = {"City ID", "City", "Country ID", "Country", "Last
Update"};

        model.setColumnIdentifiers(set_column_header(frm.getJTCity(), header)
        frm.getJTCity().setModel(model);

        Object[] row = new Object[5];

        for(int i=0; i<list.size(); i++){
            row[0] = list.get(i).getCityID();
            row[1] = list.get(i).getCity();
            row[2] = list.get(i).getCountryID();
            row[3] = list.get(i).getCountry();
            row[4] = list.get(i).getLastUpdate();
```

```
            model.addRow(row);
        }
    }

    public static void refresh_controls(CityForm frm){
        frm.setLocationRelativeTo(null);
        frm.setTitle("CITY FORM");

        //Shows the content of country table and populates combobox
        try{
            //Makes alternating color for table rows
            table_renderer(frm.getJTCity());

            //Populates table
            ArrayList<City> list = get_city_list(frm,
Query_City.get_sql_city_joint() + " ORDER BY city_id", "none");
            show_table_city(frm, list);

            //Populates jcbCityID
            String sql_city_id = "SELECT city_id FROM city ORDER BY city_id";
            populate_combobox(sql_city_id, frm.getJCBCityID(), frm);

            //Populates jcbCity
            String sql_city = "SELECT DISTINCT city FROM city ORDER BY city";
            populate_combobox(sql_city, frm.getJCBCity(), frm);

            //Populates jcbCountryID
            String sql_country_id = "SELECT country_id FROM country ORDER BY
country_id";
            populate_combobox(sql_country_id, frm.getJCBCountryID(), frm);

            //Populates getJCBCountry()
            String sql_ct = "SELECT DISTINCT country FROM country ORDER BY
country";
            populate_combobox(sql_ct, frm.getJCBCountry(), frm);

        }catch (SQLException ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
            "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

   //Displays country data result row by row
    private static <T> void display_country_data(CityForm frm, String sql, T
item){
        try(Connection conn = getConnection()){
            PreparedStatement ps = conn.prepareStatement(sql);
            ps.setObject(1,item);
            ResultSet rs = ps.executeQuery();

            if (!rs.next()) {
                // no row found, clear the form fields
```

```
                frm.getJCBCountryID().setSelectedIndex(-1);
                frm.getJCBCountry().setSelectedIndex(-1);
                return;
            }

            do{
                // Determines item selected from jcbCountryID
                find_combo_value_selected(frm.getJCBCountryID(),
rs.getInt("country_id"));

                // Determines item selected from jcbCountry
                find_combo_value_selected(frm.getJCBCountry(),
rs.getString("country"));
            }while(rs.next());

            rs.close();
            ps.close();
        }catch(SQLException ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    public static void jcbCountry_handler(CityForm frm, JComboBox<String> jcb
{
        Object item = jcb.getSelectedItem();
        String sql = "";
        if (jcb.equals(frm.getJCBCountryID())) {
            sql = Query_Country.get_sql_id();
        } else if (jcb.equals(frm.getJCBCountry())) {
            sql = Query_Country.get_sql_name();
        }

        display_country_data(frm, sql, item);
    }

    private static void clear_controls(CityForm frm){
        frm.getJTFCityID().setText("");
        frm.getJTFCity().setText("");
        frm.getJTFLastUpdate().setText("");
        frm.getJCBCountryID().setSelectedIndex(-1);
        frm.getJCBCountry().setSelectedIndex(-1);
    }

   //Displays city data result row by row
    private static <T> void display_city_data(CityForm frm, String sql, T ite
{
        try(Connection conn = getConnection()){
            PreparedStatement ps = conn.prepareStatement(sql);
            ps.setObject(1,item);
            ResultSet rs = ps.executeQuery();

            if (!rs.next()) {
```

```
                // no row found, clear the form fields
                clear_controls(frm);
                return;
            }

            do{

frm.getJTFCityID().setText(String.valueOf(rs.getInt("city_id")));
                frm.getJTFCity().setText(rs.getString("city"));

frm.getJTFLastUpdate().setText(String.valueOf(rs.getTimestamp("last_update"));

                // Determines item selected from jcbCityID
                find_combo_value_selected(frm.getJCBCityID(),
rs.getInt("city_id"));

                // Determines item selected from jcbCity
                find_combo_value_selected(frm.getJCBCity(),
rs.getString("city"));

                // Determines item selected from jcbCountryID
                find_combo_value_selected(frm.getJCBCountryID(),
rs.getInt("country_id"));

                // Determines item selected from jcbCountry
                find_combo_value_selected(frm.getJCBCountry(),
rs.getString("country"));
            }while(rs.next());

            rs.close();
            ps.close();
        }catch(SQLException ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    public static void jcbCity_handler(CityForm frm, JComboBox<String> jcb) {
        Object item = jcb.getSelectedItem();
        String sql = "";
        if (jcb.equals(frm.getJCBCityID())) {
            sql = Query_City.get_sql_city_joint() + " WHERE city_id = ?";
        } else if (jcb.equals(frm.getJCBCity())) {
            sql = Query_City.get_sql_city_joint() + " WHERE city = ?";
        }

        display_city_data(frm, sql, item);
    }

    public static void show_first_row(CityForm frm){
        String item =
String.valueOf(frm.getJCBCityID().getItemAt(FIRST_INDEX));
        display_city_data(frm, SQL_ID, item);
```

```
        currentIndex = FIRST_INDEX;
    }

    public static void show_last_row(CityForm frm){
        int endIndex = frm.getJCBCityID().getItemCount() - 1;
        String item = String.valueOf(frm.getJCBCityID().getItemAt(endIndex));
        display_city_data(frm, SQL_ID, item);
        currentIndex = endIndex;
    }

    public static void show_prev_row(CityForm frm){
        currentIndex--;
        if(currentIndex < FIRST_INDEX){
            currentIndex = FIRST_INDEX;
            return;
        }
        String item =
String.valueOf(frm.getJCBCityID().getItemAt(currentIndex));
        display_city_data(frm, SQL_ID, item);
    }

    public static void show_next_row(CityForm frm){
        int endIndex = frm.getJCBCityID().getItemCount() - 1;
        currentIndex++;
        if(currentIndex > endIndex){
            currentIndex = endIndex;
            return;
        }
        String item =
String.valueOf(frm.getJCBCityID().getItemAt(currentIndex));
        display_city_data(frm, SQL_ID, item);
    }

    public static void mouse_pressed_handler(CityForm frm) {
        Objects.requireNonNull(frm, "frm must not be null");

        int selectedIndex = frm.getJTCity().getSelectedRow();
        if (selectedIndex == -1) {
            JOptionPane.showMessageDialog(frm, "Please select a row to view 
data.",
                "No row selected", JOptionPane.INFORMATION_MESSAGE);
            return;
        }

        try (Connection conn = getConnection()) {
            String id =
String.valueOf(frm.getJTCity().getModel().getValueAt(selectedIndex, 0));

            // Displays city data
            display_city_data(frm, SQL_ID, id);

        } catch (SQLException ex) {
```

```
            Logger.getLogger(CityForm.class.getName()).log(Level.SEVERE, "Err
displaying city data", ex);
            String message = "Error displaying city data: " + ex.getMessage()
            String stackTrace = Arrays.toString(ex.getStackTrace());
            JOptionPane.showMessageDialog(frm, message + "\n\n" + stackTrace,
"ERROR", JOptionPane.ERROR_MESSAGE);
        }
    }

    //Updates row of data in city tabel by city_id
    public static void update_row_by_city_id(CityForm frm, int city_id, Strin
city, int country_id) throws SQLException{
        Connection conn = getConnection();
        ResultSet rs = null;
        String query_id = "SELECT city_id FROM city WHERE city_id = ?";
        String update_query = """
            UPDATE city SET city = ?, country_id = ? WHERE city_id = ?""";
        try(PreparedStatement idPs = conn.prepareStatement(query_id,

ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_READ_ONLY);
        PreparedStatement updatePS = conn.prepareStatement(update_query,
                ResultSet.TYPE_SCROLL_SENSITIVE, ResultSet.CONCUR_UPDATABLE))
        {
            idPs.setInt(1,city_id);
            if(!idPs.execute()){
                String message = "Can't find city_id " + city_id;

                JOptionPane.showMessageDialog(frm, message,
                    "ERROR",JOptionPane.ERROR_MESSAGE);
            } else{
                rs = idPs.getResultSet();
                rs.next();

                //Creates a City object using four-params constructor
                City obj = new City(city_id, city, country_id, new
Timestamp(System.currentTimeMillis()));
                updatePS.setString(1, obj.getCity());
                updatePS.setInt(2, obj.getCountryID());
                updatePS.setInt(3, obj.getCityID());

                updatePS.executeUpdate();
                rs.close();
                updatePS.close();
                idPs.close();
                conn.close();
            }
        }catch(SQLException ex){
            Logger.getLogger(CityForm.class.getName()).log(Level.SEVERE, "Err
updating city data", ex);
            String message = "Error updating city data: " + ex.getMessage();
            String stackTrace = Arrays.toString(ex.getStackTrace());
            JOptionPane.showMessageDialog(null, message + "\n\n" + stackTrace
"ERROR", JOptionPane.ERROR_MESSAGE);
```

```
        }catch(java.lang.NumberFormatException ex){
            Logger.getLogger(CityForm.class.getName()).log(Level.SEVERE,
"Invalid Input", ex);
            String message = "Invalid Input: " + ex.getMessage();
            String stackTrace = Arrays.toString(ex.getStackTrace());
            JOptionPane.showMessageDialog(null, message + "\n\n" + stackTrace
"ERROR", JOptionPane.ERROR_MESSAGE);
        }
    }

    private static HashMap<String, String> read_inputs(CityForm frm) {
        HashMap<String, String> input_data = new HashMap<>();
        String city_id = String.valueOf(frm.getJCBCityID().getSelectedItem())
        String country_id =
String.valueOf(frm.getJCBCountryID().getSelectedItem());
        String city = frm.getJTFCity().getText();

        // Validate user input
        int country_id_int = 0;
        try {
            country_id_int = Integer.parseInt(country_id);
            if (country_id_int <= 0) {
                throw new IllegalArgumentException("Country ID cannot be
negative or zero");
            }
        } catch (NumberFormatException ex) {
            JOptionPane.showMessageDialog(frm, "Invalid Country ID: " +
country_id,
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        } catch (IllegalArgumentException ex) {
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        }

        int city_id_int = 0;
        try {
            city_id_int = Integer.parseInt(city_id);
            if (city_id_int <= 0) {
                throw new IllegalArgumentException("City ID cannot be negativ
or zero");
            }
        } catch (NumberFormatException ex) {
            JOptionPane.showMessageDialog(frm, "Invalid City ID: " + city_id,
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        } catch (IllegalArgumentException ex) {
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        }
```

```
        if (city == null || city.isEmpty()) {
            JOptionPane.showMessageDialog(frm, "City cannot be empty",
            "Error", JOptionPane.ERROR_MESSAGE);
            throw new IllegalArgumentException("City cannot be empty");
        }

        input_data.put("city_id", city_id);
        input_data.put("country_id", country_id);
        input_data.put("city", city);

        return input_data;
    }

    private static void edit_actual(CityForm frm){
        try{
            HashMap<String, String> input_data = read_inputs(frm);
            int city_id = Integer.parseInt(input_data.get("city_id"));
            int country_id = Integer.parseInt(input_data.get("country_id"));
            String city = input_data.get("city");

            update_row_by_city_id(frm, city_id, city, country_id);

            //Refreshes all objects on form
            refresh_controls(frm);

        }catch(SQLException ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }
    private static void enable_controls(boolean state, CityForm frm){
        frm.getJBFirst().setEnabled(state);
        frm.getJBPrev().setEnabled(state);
        frm.getJBNext().setEnabled(state);
        frm.getJBLast().setEnabled(state);
        frm.getJBInsert().setEnabled(state);
        frm.getJBDelete().setEnabled(state);
        frm.getJTFCityID().setEnabled(state);
    }
    public static void edit_handler(CityForm frm){
        if(frm.getJBEdit().getText().equals("EDIT")){
            frm.getJBEdit().setText("CONFIRM");

            // Disables controls
            enable_controls(false, frm);
        }

        else {
            frm.getJBEdit().setText("EDIT");

            // Actual editing
            edit_actual(frm);
```

```
            //Enables controls
            enable_controls(true, frm);
        }
    }

    public static void edit_database_from_jtable(TableModelEvent e, CityForm
frm){
        if (e.getType() == TableModelEvent.UPDATE) {
            int row = e.getFirstRow();
            TableModel model = (TableModel)e.getSource();
            int city_id = Integer.parseInt(String.valueOf(model.getValueAt(r
0)));
            String city = (String) model.getValueAt(row, 1);
            int country_id =
Integer.parseInt(String.valueOf(model.getValueAt(row, 2)));

            try{
                update_row_by_city_id(frm, city_id, city, country_id);

                //Refreshes all objects on form
                refresh_controls(frm);

            } catch (SQLIntegrityConstraintViolationException ex) {
                Logger.getLogger(CityForm.class.getName()).log(Level.SEVERE,
"Duplicate entry", ex);
                JOptionPane.showMessageDialog(frm, "Error: Duplicate entry\n'
ex.getMessage());
            } catch (SQLSyntaxErrorException ex) {
                Logger.getLogger(CityForm.class.getName()).log(Level.SEVERE,
"Invalid SQL syntax", ex);
                JOptionPane.showMessageDialog(frm, "Error: Invalid SQL
syntax\n" + ex.getMessage());
            } catch (SQLException ex) {
                Logger.getLogger(CityForm.class.getName()).log(Level.SEVERE,
"Database error", ex);
                JOptionPane.showMessageDialog(frm, "Error: Database error\n"
ex.getMessage());
            }
        }
    }

    //Inserts new row into city table
    private static void insert_row(CityForm frm) throws SQLException{
        HashMap<String, String> input_data = read_inputs(frm);
        String city = input_data.get("city");
        int country_id = Integer.parseInt(input_data.get("country_id"));

        // SQL insert statement
        String sql = """
            INSERT INTO city(city, country_id) VALUES(?, ?)""";

        try(Connection conn = getConnection();
         PreparedStatement pstmt = conn.prepareStatement(sql)){
```

```
            //Creates a City object three-params constructor
            City obj = new City(city, country_id, new
Timestamp(System.currentTimeMillis()));
            pstmt.setString(1,obj.getCity());
            pstmt.setInt(2,obj.getCountryID());

            //Executes the sql insert statement
            pstmt.executeUpdate();
        } catch (SQLException ex) {
            Logger.getLogger(CityForm.class.getName()).log(Level.SEVERE,
"Database error", ex);
            JOptionPane.showMessageDialog(frm, "Error: Database error\n" +
ex.getMessage());
        }
    }

    private static void insert_actual(CityForm frm){
        try{
            insert_row(frm);

            //Refreshes table and comboboxes
            refresh_controls(frm);

        }catch(SQLException ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }
    public static void insert_handler(CityForm frm){
        if(frm.getJBInsert().getText().equals("INSERT")){
            frm.getJBInsert().setText("CONFIRM");

            //Disables jbEdit
            frm.getJBEdit().setEnabled(false);

            // Disables controls
            enable_controls(false, frm);
            frm.getJCBCityID().setEnabled(false);
            frm.getJCBCity().setEnabled(false);

            // Clears controls
            clear_controls(frm);

            // Enables
            frm.getJBInsert().setEnabled(true);
        }

        else {
            frm.getJBInsert().setText("INSERT");

            // Actual insertion
            insert_actual(frm);
```

```
            //Enables jbEdit
            frm.getJBEdit().setEnabled(true);

            //Enables controls
            enable_controls(true, frm);
            frm.getJCBCityID().setEnabled(true);
            frm.getJCBCity().setEnabled(true);
        }
    }

    public static void delete_handler(CityForm frm){
        int dialogButton = JOptionPane.YES_NO_OPTION;
        int city_id =
Integer.parseInt(String.valueOf(frm.getJCBCityID().getSelectedItem()));

        String message = String.format("Are you sure you want to delete the r
City ID: %d)", city_id);
        int answer = JOptionPane.showConfirmDialog(frm, message, "DELETING RO
OF DATA", dialogButton);

        if(answer == JOptionPane.YES_OPTION){
            String query = """
                DELETE FROM city WHERE city_id = ?""";
            try(Connection conn = getConnection();
                PreparedStatement ps = conn.prepareStatement(query)){
                // Use PreparedStatement to avoid SQL injection attacks
                ps.setInt(1, city_id);
                ps.executeUpdate();

                // Refresh table and comboboxes
                refresh_controls(frm);

            } catch (SQLException ex){
                JOptionPane.showMessageDialog(frm, ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
            }
        }
    }

    private static void draw_pie_chart_city_by_country(CityForm frm, JPanel j
{
        jp.setPreferredSize(new Dimension(jp.getWidth(), jp.getHeight()));
        DefaultPieDataset dataset =
create_pie_dataset(Query_City.get_sql_city_country_dist(), "Number",
"co.country");

        //Draws piechart city distribution by country
        draw_piechart_with_dataset(frm, jp, dataset, "TOP 10 CITY DISTRIBUTIO
BY COUNTRY");
    }
```

```
    private static void draw_bar_chart_city_by_country(CityForm frm, JPanel j
{
        jp.setPreferredSize(new Dimension(jp.getWidth(), jp.getHeight()));

        DefaultCategoryDataset dataset =
create_bar_dataset(Query_City.get_sql_city_country_dist(), "Number",
"co.country");

        //Draws barchart city distribution by country
        draw_barchart_with_dataset(frm, jp, dataset, "THE 10 CITY DISTRIBUTI(
BY COUNTRY", "COUNTRY", "NUMBER");
    }

    public static void jbchart_handler(CityForm frm){
        //Draws piechart city distribution by country
        draw_pie_chart_city_by_country(frm, frm.getJPanel1());

        //Draws barchart city distribution by country
        draw_bar_chart_city_by_country(frm, frm.getJPanel2());
    }
}
```

This is the full version of **CityForm.java**:

```
package sakila;
import java.awt.Toolkit;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.IOException;
import java.text.ParseException;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.JButton;
import javax.swing.JComboBox;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JTable;
import javax.swing.JTextField;

public class CityForm extends javax.swing.JFrame {
    public CityForm() {
        initComponents();
        Utility.setLookAndFeel(this);
        City_Utils.refresh_controls(this);

 this.setIconImage(Toolkit.getDefaultToolkit().getImage(getClass().getResourc
;
//        this.setDefaultCloseOperation(this.HIDE_ON_CLOSE);
    }

    //Getter method for jtfCityID
    public JTextField getJTFCityID(){
```

```
        return this.jtfCityID;
    }

    //Getter method for jtfLastUpdate
    public JTextField getJTFLastUpdate(){
        return this.jtfLastUpdate;
    }

    //Getter method for jtfCity
    public JTextField getJTFCity(){
        return this.jtfCity;
    }

    //Getter method for jtCity
    public JTable getJTCity(){
        return this.jtCity;
    }

    //Getter method for jcbCityID
    public JComboBox getJCBCityID(){
        return this.jcbCityID;
    }

    //Getter method for jcbCity
    public JComboBox getJCBCity(){
        return this.jcbCity;
    }

    //Getter method for jcbCountryID
    public JComboBox getJCBCountryID(){
        return this.jcbCountryID;
    }

    //Getter method for jcbCountry
    public JComboBox getJCBCountry(){
        return this.jcbCountry;
    }

    //Getter method for jbEdit
    public JButton getJBEdit(){
        return this.jbEdit;
    }
```

```
    //Getter method for jbInsert
    public JButton getJBInsert(){
        return this.jbInsert;
    }

    //Getter method for jbDelete
    public JButton getJBDelete(){
        return this.jbDelete;
    }

    //Getter method for jbFirst
    public JButton getJBFirst(){
        return this.jbFirst;
    }

    //Getter method for jbPrev
    public JButton getJBPrev(){
        return this.jbPrev;
    }

    //Getter method for jbNext
    public JButton getJBNext(){
        return this.jbNext;
    }

    //Getter method for jbLast
    public JButton getJBLast(){
        return this.jbLast;
    }

    //Getter method for jbChart
    public JButton getJBChart(){
        return this.jbChart;
    }

    //Getter method for jPanel1
    public JPanel getJPanel1(){
        return this.jPanel1;
    }

    //Getter method for jPanel2
    public JPanel getJPanel2(){
        return this.jPanel2;
    }

    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">
    private void initComponents() {
        //...
        pack();
    }// </editor-fold>
```

```
    private void jcbCityIDActionPerformed(java.awt.event.ActionEvent evt) {
        City_Utils.jcbCity_handler(this, this.jcbCityID);
    }

    private void jcbCityActionPerformed(java.awt.event.ActionEvent evt) {
        City_Utils.jcbCity_handler(this, this.jcbCity);
    }

    private void jbEditActionPerformed(java.awt.event.ActionEvent evt) {
        City_Utils.edit_handler(this);
    }

    private void jbInsertActionPerformed(java.awt.event.ActionEvent evt) {
        City_Utils.insert_handler(this);
    }

    private void jbDeleteActionPerformed(java.awt.event.ActionEvent evt) {
        City_Utils.delete_handler(this);
    }

    private void jbLastActionPerformed(java.awt.event.ActionEvent evt) {
        City_Utils.show_last_row(this);
    }

    private void jbNextActionPerformed(java.awt.event.ActionEvent evt) {
        City_Utils.show_next_row(this);
    }

    private void jbPrevActionPerformed(java.awt.event.ActionEvent evt) {
        City_Utils.show_prev_row(this);
    }

    private void jbFirstActionPerformed(java.awt.event.ActionEvent evt) {
        City_Utils.show_first_row(this);
    }

    private void jtCityMousePressed(java.awt.event.MouseEvent evt) {
        City_Utils.mouse_pressed_handler(this);
    }

    private void jtCityMouseClicked(java.awt.event.MouseEvent evt) {
        // instantiate CityTableModelListener and add it as a listener to the
        CityTableModelListener tableModelListener = new CityTableModelListene
this);
        this.getJTCity().getModel().addTableModelListener(tableModelListener)
    }

    private void jcbCountryActionPerformed(java.awt.event.ActionEvent evt) {
        City_Utils.jcbCountry_handler(this, this.jcbCountry);
    }

    private void jcbCountryIDActionPerformed(java.awt.event.ActionEvent evt)
```

```
        City_Utils.jcbCountry_handler(this, this.jcbCountryID);
    }

    private void jbChartActionPerformed(java.awt.event.ActionEvent evt) {
        City_Utils.jbchart_handler(this);
    }

    /**
     * @param args the command line arguments
     */
    public static void main(String args[]) {
        /* Set the Nimbus look and feel */
        //<editor-fold defaultstate="collapsed" desc=" Look and feel setting
        /* If Nimbus (introduced in Java SE 6) is not available, stay with th
feel.
         * For details see
http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html
         */
        try {
            for (javax.swing.UIManager.LookAndFeelInfo info :
javax.swing.UIManager.getInstalledLookAndFeels()) {
                if ("Nimbus".equals(info.getName())) {
                    javax.swing.UIManager.setLookAndFeel(info.getClassName())
                    break;
                }
            }
        } catch (ClassNotFoundException ex) {

java.util.logging.Logger.getLogger(CityForm.class.getName()).log(java.util.lo
null, ex);
        } catch (InstantiationException ex) {

java.util.logging.Logger.getLogger(CityForm.class.getName()).log(java.util.lo
null, ex);
        } catch (IllegalAccessException ex) {

java.util.logging.Logger.getLogger(CityForm.class.getName()).log(java.util.lo
null, ex);
        } catch (javax.swing.UnsupportedLookAndFeelException ex) {

java.util.logging.Logger.getLogger(CityForm.class.getName()).log(java.util.lo
null, ex);
        }

        /* Create and display the form */
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new CityForm().setVisible(true);
            }
        });
    }

    // Variables declaration - do not modify
```

```
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JLabel jLabel3;
    private javax.swing.JLabel jLabel4;
    private javax.swing.JLabel jLabel5;
    private javax.swing.JPanel jPanel1;
    private javax.swing.JPanel jPanel2;
    private javax.swing.JScrollPane jScrollPane1;
    private javax.swing.JButton jbChart;
    private javax.swing.JButton jbDelete;
    private javax.swing.JButton jbEdit;
    private javax.swing.JButton jbFirst;
    private javax.swing.JButton jbInsert;
    private javax.swing.JButton jbLast;
    private javax.swing.JButton jbNext;
    private javax.swing.JButton jbPrev;
    private javax.swing.JComboBox<String> jcbCity;
    private javax.swing.JComboBox<String> jcbCityID;
    private javax.swing.JComboBox<String> jcbCountry;
    private javax.swing.JComboBox<String> jcbCountryID;
    private javax.swing.JTable jtCity;
    private javax.swing.JTextField jtfCity;
    private javax.swing.JTextField jtfCityID;
    private javax.swing.JTextField jtfLastUpdate;
    // End of variables declaration
}
```

ADDRESS FORM

ADDRESS FORM

CREATING AND POPULATING ADDRESS TABLE

CREATING AND POPULATING ADDRESS TABLE

Step 1	Create a new class named **Query_Address**. It contains several static String variables that store SQL queries. Here is an explanation of each query: 1. **sql_min**: This query retrieves the minimum value of the **address_id** column from the **address** table. 2. **sql_max**: This query retrieves the maximum value of the **address_id** column from the **address** table. 3. **sql_id**: This query retrieves all columns from the **address** table along with the corresponding **city** and **country** for a specific **address_id** passed as a parameter. 4. **sql_address_district_dist**: This query retrieves the **district** column and the number of occurrences of each district from the address table, grouped by district, ordered by the number of occurrences in descending order, and limited to the top 10 districts.

5. **sql_address_country_dist**: This query retrieves the **country** column and the number of occurrences of each country from the **address** table joined with the **city** and **country** tables, grouped by country, ordered by the number of occurrences in descending order, and limited to the top 10 countries.
6. **sql_address_city_dist**: This query retrieves the **city** column and the number of occurrences of each city from the **address** table joined with the **city** table, grouped by city, ordered by the number of occurrences in descending order, and limited to the top 10 cities.
7. **sql_address_joint**: This query retrieves all columns from the address table along with the corresponding city and country, by joining the **address**, **city**, and **country** tables.
8. **sql_address**: This query creates a new table called address with several columns, including **address_id**, **address**, **address2**, **district**, **city_id**, **postal_code**, **phone**, and **last_update**. It also sets the **address_id** column as the primary key and creates a foreign key constraint referencing the **city_id** column in the **city** table.

Each query is accompanied by a corresponding getter method that returns the respective SQL query string.

```
1  package sakila;
2
3  public class Query_Address {
4      private static final String
5  sql_min = "SELECT MIN(address_id)
6  FROM address";
7      private static final String
8  sql_max = "SELECT MAX(address_id)
9  FROM address";
10     private static final String
11 sql_id = get_sql_address_joint() +
12 " WHERE address_id  = ?";
13
14     private static final String
15 sql_address_district_dist = """
16             SELECT ad.district,
17 COUNT(*) AS Number
18             FROM address ad
19             GROUP BY ad.district
20             ORDER BY Count(*) DESC
21             LIMIT 10""";
22
23     private static final String
24 sql_address_country_dist = """
25
```

```
        SELECT co.country, COUNT(*)
AS Number
        FROM address ad
        JOIN city ci ON ci.city_id
= ad.city_id
        JOIN country co ON
co.country_id = ci.country_id
        GROUP BY co.country
        ORDER BY Count(*) DESC
        LIMIT 10""";

    private static final String
sql_address_city_dist = """
        SELECT ci.city, COUNT(*) AS
Number
        FROM address ad
        JOIN city ci ON ci.city_id
= ad.city_id
        GROUP BY ci.city
        ORDER BY Count(*) DESC
        LIMIT 10""";

    private static final String
sql_address_joint = """
    SELECT ad.address_id,
ad.address, ad.address2,
        ad.district, ad.city_id,
ad.postal_code,
        ad.phone, ad.last_update,
ci.city,
        co.country
    FROM address ad
    JOIN city ci ON ci.city_id =
ad.city_id
    JOIN country co ON
co.country_id = ci.country_id""";

    private static final String
sql_address = """
        CREATE TABLE address (
          address_id SMALLINT
UNSIGNED NOT NULL AUTO_INCREMENT,
          address VARCHAR(50) NOT
NULL,
          address2 VARCHAR(50)
DEFAULT NULL,
          district VARCHAR(20) NOT
NULL,
          city_id SMALLINT UNSIGNED
NOT NULL,
          postal_code VARCHAR(10)
DEFAULT NULL,


```

```
          phone VARCHAR(20) NOT
NULL,
          last_update TIMESTAMP NOT
NULL DEFAULT CURRENT_TIMESTAMP ON
UPDATE CURRENT_TIMESTAMP,
          PRIMARY KEY
(address_id),
          KEY idx_fk_city_id
(city_id),
          CONSTRAINT
`fk_address_city` FOREIGN KEY
(city_id) REFERENCES city (city_id)
ON DELETE RESTRICT ON UPDATE
CASCADE
        ) ENGINE=InnoDB DEFAULT
CHARSET=utf8mb4;""";


    //Getter methods
    public static String
get_sql_min() {
        return sql_min;
    }

    public static String
get_sql_max() {
        return sql_max;
    }

    public static String
get_sql_id() {
        return sql_id;
    }

    public static String
get_sql_address() {
        return sql_address;
    }

    public static String
get_sql_address_joint() {
        return sql_address_joint;
    }

    public static String
get_sql_address_city_dist() {
        return
sql_address_city_dist;
    }

    public static String
get_sql_address_country_dist() {
```

```
        return
sql_address_country_dist;
    }

    public static String
get_sql_address_district_dist() {
        return
sql_address_district_dist;
    }
}
```

Step 2

Then, create a public class named **Address**. It represents an address in the Sakila DVD rental database. It has 10 instance variables, including the address ID, the street address, a second address line, the district, the city ID, postal code, phone number, the last update timestamp, the city, and the country.

The class has several constructors with different numbers of parameters, allowing for different ways to create an **Address** object. It also has getter and setter methods for each instance variable, with appropriate input validation in the setters.

The class also overrides the **hashCode()**, **equals()**, and **toString()** methods for proper object comparison and printing.

```
1       package sakila;
2       import java.util.Objects;
3       import java.sql.Timestamp;
4
5       public class Address {
6           //10 instance variables
7           private int address_id;
8           private String address;
9           private String address2;
10          private String district;
11          private int city_id;
12          private String postal_code;
13          private String phone;
14          private Timestamp last_update;
15          private String city;
16          private String country;
17
18          //default-constructor
19          Address(){
20              this(1, "Address xxx", null,
21      "District xxx", 1, "Post xxx", "Phone
22      xxx", new
23      Timestamp(System.currentTimeMillis()));
24          }
```

```
25
26      //Seven-params constructor
27      Address(String add, String add2,
28  String dist, int city_id, String post,
29  String ph, Timestamp lu){
30          setAddress(add);
31          setAddress2(add2);
32          setDistrict(dist);
33          setCityID(city_id);
34          setPostalCode(post);
35          setPhone(ph);
36          setLastUpdate(lu);
37      }
38
39      //Eight-params constructor
40      Address(int add_id, String add,
41  String add2, String dist, int city_id,
42  String post, String ph, Timestamp lu){
43          this(add, add2, dist, city_id,
44  post, ph, lu);
45          setAddressID(add_id);
46      }
47
48      //Ten-params constructor
49      Address(int add_id, String add,
50  String add2, String dist, int city_id,
51  String post, String ph, String city,
52  String country, Timestamp lu){
53          this(add_id, add, add2, dist,
54  city_id, post, ph, lu);
55          this.city = city;
56          this.country = country;
57      }
58
59      //Getter methods
60      public int getAddressID() {return
61  address_id;}
62      public String getAddress() {return
63  address;}
64      public String getAddress2() {return
65  address2;}
66      public String getDistrict() {return
67  district;}
68      public int getCityID() {return
69  city_id;}
70      public String getPostalCode()
71  {return postal_code;}
72      public String getPhone() {return
73  phone;}
74      public Timestamp getLastUpdate()
75  {return last_update;}
76      public String getCity() {return
77  city;}
78
```

```
    public String getCountry() {return country;}

    //Setter methods
    public void setAddressID(int id) {
        if (id <= 0) {
            throw new IllegalArgumentException("Address ID must be greater than zero.");
        }
        this.address_id = id;
    }

    public void setAddress(String add) {
        if (add == null || add.trim().isEmpty()) {
            throw new IllegalArgumentException("Address cannot be null or empty");
        }
        if (add.length() > 50) {
            throw new IllegalArgumentException("Address cannot be longer than 50 characters");
        }
        this.address = add;
    }

    public void setAddress2(String add) {
        if (add != null && add.length() > 50) {
            throw new IllegalArgumentException("Address2 cannot be longer than 50 characters");
        }
        this.address2 = (add != null) ? add : ""; // set default value to empty string if add is null
    }

    public void setDistrict(String dist) {
        if (dist == null || dist.trim().isEmpty()) {
            throw new IllegalArgumentException("District cannot be null or empty");
        }
        if (dist.length() > 20) {
```

```
            throw new
IllegalArgumentException("District
cannot be longer than 20 characters");
        }
        this.district = dist;
    }

    public void setCityID(int id) {
        if (id <= 0) {
            throw new
IllegalArgumentException("City ID must be greater
than zero.");
        }
        this.city_id = id;
    }

    public void setPostalCode(String
post) {
        if (post != null &&
post.length() > 10) {
            throw new
IllegalArgumentException("Postal code
cannot be longer than 10 characters");
        }
        this.postal_code = (post !=
null) ? post : ""; // set default value
to empty string if add is null
    }

    public void setPhone(String phone)
{
        if (phone == null ||
phone.trim().isEmpty()) {
            throw new
IllegalArgumentException("Phone cannot
be null or empty");
        }
        if (phone.length() > 20) {
            throw new
IllegalArgumentException("Phone cannot
be longer than 20 characters");
        }
        this.phone = phone;
    }

    public void setLastUpdate(Timestamp
date){
        if (date == null) {
        throw new
IllegalArgumentException("Date cannot
be null");
        }
        this.last_update = date;
```

```
    }

    // Override the hashCode() method
    @Override
    public int hashCode() {
        return Objects.hash(address_id,
address, address2, district,
      city_id, postal_code, phone,
last_update, city, country);
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() !=
o.getClass()) return false;
        Address add = (Address) o;
        return city_id == add.city_id
&&
                address_id ==
add.address_id &&
                Objects.equals(address,
add.address) &&
                Objects.equals(address2,
add.address2) &&
                Objects.equals(district,
add.district) &&

Objects.equals(postal_code,
add.postal_code) &&
                Objects.equals(phone,
add.phone) &&
                Objects.equals(city,
add.city) &&
                Objects.equals(country,
add.country) &&

Objects.equals(last_update,
add.last_update);
    }

    @Override
    public String toString(){
        return "\nAddress ID    : " +
getAddressID() +
                "\nAddress       : " +
getAddress() +
                "\nAddress2      : " +
getAddress2() +
                "\nDistrict      : " +
getDistrict() +
                "\nCity ID       : " +
getCityID() +
```

```
                "\nPostal Code   : " +
getPostalCode() +
                "\nPhone         : " +
getPhone() +
                "\nCity          : " +
getCity() +
                "\nCountry       : " +
getCountry() +
                "\nLast Update   : " +
getLastUpdate();
    }
}
```

Step 3

Create a new public class named **Address_Utils**. It extends the **Utility** class.

```
1  package sakila;
2  import java.awt.Dimension;
3  import java.util.logging.Level;
4  import java.util.logging.Logger;
5  import java.sql.*;
6  import java.util.ArrayList;
7  import java.util.Arrays;
8  import java.util.HashMap;
9  import java.util.Objects;
10 import javax.swing.JComboBox;
11 import javax.swing.JOptionPane;
12 import javax.swing.JPanel;
13 import javax.swing.event.TableModelEvent;
14 import javax.swing.table.DefaultTableModel;
15 import javax.swing.table.TableModel;
16 import org.jfree.data.category.DefaultCategoryDataset;
17 import org.jfree.data.general.DefaultPieDataset;
18 import static sakila.Utility.create_bar_dataset;
19 import static sakila.Utility.draw_barchart_with_dataset;
20
21 public class Address_Utils extends Utility{
22     public static final int FIRST_INDEX = 0;
23     public static final int INVALID_INDEX = -1;
24
25     private static int currentIndex = FIRST_INDEX;
26     private static final String SQL_ID =
27 Query_Address.get_sql_id();
28
29     //Creates address table
30     public static void create_address_table() {
31         try (Connection conn = getConnection()) {
32             Statement stmt = conn.createStatement();
33
34 stmt.addBatch(Query_Address.get_sql_address());
35
```

```
            stmt.executeBatch();

            String message = String.format("Successfully
creates address table");
            JOptionPane.showMessageDialog(null, message,

 "INFORMATION",JOptionPane.INFORMATION_MESSAGE);

        } catch (SQLException ex) {
            JOptionPane.showMessageDialog(null,
ex.getMessage(),
                "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    //Populates address table with some rows of data
    public static void populate_address_table(){
        try(Connection conn = getConnection()){
            String sql = """
                INSERT INTO address(address_id, address,
address2, district,
                    city_id, postal_code, phone,
last_update)
                VALUES(?, ?, ?, ?, ?, ?, ?, ?)""";

            //Creates a new Address class with default
constructor
            PreparedStatement ps1 =
conn.prepareStatement(sql);
            Address obj1 = new Address();
            ps1.setInt(1,obj1.getAddressID());
            ps1.setString(2,obj1.getAddress());
            ps1.setString(3,obj1.getAddress2());
            ps1.setString(4,obj1.getDistrict());
            ps1.setInt(5,obj1.getCityID());
            ps1.setString(6,obj1.getPostalCode());
            ps1.setString(7,obj1.getPhone());
            ps1.setTimestamp(8,obj1.getLastUpdate());

            // Creates a new Address class with eight-
params constructor
            PreparedStatement ps2 =
conn.prepareStatement(sql);
            Address obj2 = new Address(2, "Balige No. 1",
"Balige No. 2", "BLG", 2, "89222", "0892682322", new
Timestamp(System.currentTimeMillis()));
            ps2.setInt(1,obj2.getAddressID());
            ps2.setString(2,obj2.getAddress());
            ps2.setString(3,obj2.getAddress2());
            ps2.setString(4,obj2.getDistrict());
            ps2.setInt(5,obj2.getCityID());
            ps2.setString(6,obj2.getPostalCode());
            ps2.setString(7,obj2.getPhone());

```

```
            ps2.setTimestamp(8,obj2.getLastUpdate());

            ps1.executeUpdate();
            ps2.executeUpdate();

        }catch(SQLException ex){
            JOptionPane.showMessageDialog(null,
ex.getMessage(),
                "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    //Reads the content of joined address, city and
country tables
    public static void read_address_table(){
        try(Connection conn = getConnection()){
            Statement stmt = conn.createStatement();
            ResultSet rs =
stmt.executeQuery(Query_Address.get_sql_address_joint());

            while(rs.next()){
                int add_id = rs.getInt("address_id");
                String add = rs.getString("address");
                String add2 = rs.getString("address2");
                String dist = rs.getString("district");
                int city_id = rs.getInt("city_id");
                String post =
rs.getString("postal_code");
                String ph = rs.getString("phone");
                String city = rs.getString("city");
                String country = rs.getString("country");
                Timestamp lu =
rs.getTimestamp("last_update");

                //Creates a Address object using ten-
params constructor
                Address obj = new Address(add_id, add,
add2, dist, city_id, post, ph, city, country, lu);
                System.out.println(obj);
            }
            rs.close();
            stmt.close();

        }catch(SQLException ex){
            JOptionPane.showMessageDialog(null,
ex.getMessage(),
                "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }
}
```

Step 4	In the driver class, **Sakila.java**, invoke **create_address_table()**, **populate_address_table()**, and **read_address_table()** as shown in line 54 – 56: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36

```
37 package sakila;
38
39 public class Sakila {
40     public static void main(String[] args) {
41 //         Utility.testConnection();
42 //         Actor_Utils.create_actor_table();
43 //         Actor_Utils.populate_actor_table();
44 //         Actor_Utils.read_actor_table();
45 //         ActorForm frm = new ActorForm();
46 //         frm.setVisible(true);
47
48 //         Language_Utils.create_language_table();
49 //
50 Language_Utils.populate_language_table();
51 //         Language_Utils.read_language_table();
52 //         LanguageForm frm = new LanguageForm();
53 //         frm.setVisible(true);
54
55 //         Category_Utils.create_category_table();
56 //
57 Category_Utils.populate_category_table();
58 //         Category_Utils.read_category_table();
   //         CategoryForm frm = new CategoryForm();
   //         frm.setVisible(true);

   //         Film_Utils.create_film_table();
   //         Film_Utils.populate_film_table();
   //         Film_Utils.read_film_table();
   //         FilmForm frm = new FilmForm();
   //         frm.setVisible(true);

   //
   FilmActor_Utils.create_film_actor_table();
   //
   FilmActor_Utils.populate_film_actor_table();
   //         FilmActor_Utils.read_film_actor_table();
   //         FilmActorForm frm = new FilmActorForm();
   //         frm.setVisible(true);

   //
   FilmCategory_Utils.create_film_category_table();
   //
   FilmCategory_Utils.populate_film_category_table();
   //
   FilmCategory_Utils.read_film_category_table();
   //         FilmCategoryForm frm = new
   FilmCategoryForm();
   //         frm.setVisible(true);

   //         Country_Utils.create_country_table();
   //         Country_Utils.populate_country_table();
   //         Country_Utils.read_country_table();
```

```
//        CountryForm frm = new CountryForm();
//        frm.setVisible(true);

//        City_Utils.create_city_table();
//        City_Utils.populate_city_table();
//        City_Utils.read_city_table();
//        CityForm frm = new CityForm();
//        frm.setVisible(true);

        Address_Utils.create_address_table();
        Address_Utils.populate_address_table();
        Address_Utils.read_address_table();
    }
}
```

Run project to see the result in console:

```
Address ID    : 1
Address       : Address xxx
Address2      :
District      : District xxx
City ID       : 1
Postal Code   : Post xxx
Phone         : Phone xxx
City          : Balige
Country       : Toba State
Last Update   : 2023-04-28 11:12:40.0

Address ID    : 2
Address       : Balige No. 1
Address2      : Balige No. 2
District      : BLG
City ID       : 2
Postal Code   : 89222
Phone         : 0892682322
City          : Tobasa
Country       : Toba State
Last Update   : 2023-04-28 11:12:40.0
```

DESIGNING GUI
DESIGNING GUI

Step 1	In the project, create a new **JFrame Form** and name it as **AddressForm.java**. In the Design tab, add nine **JLabels** to the form and set their corresponding **text** properties as ADDRESS ID, ADDRESS, ADDRESS 2,

	DISTRICT, POSTAL CODE, CITY, COUNTRY, PHONE, and LAST UPDATE.
Step 2	Then, add seven **JTextField** to the form and set their corresponding **Variable Name** as **jtfAddressID**, **jtfAddress**, **jtfAddress2**, **jtfDistrict**, **jtfPostalCode**, **jtfPhone**, and **jtfLastUpdate**.
Step 3	Then, add ten **JButton** to the form and set their corresponding **Variable Name** as **jbFirst**, **jbPrev**, **jbNext**, **jbLast**, **jbEdit**, **jbInsert**, **jbDelete**, **jbCountryForm**, **jbCityForm**, and **jbChart**. Set their corresponding **text** properties as FIRST, PREV, NEXT, LAST, EDIT, INSERT, DELETE, COUNTRY FORM, CITY FORM, and CHART.
Step 4	Then, add five **JComboBoxes** to the form and set their corresponding **Variable Name** as **jcbAddressID**, **jcbDistrict**, **jcbCityID**, **jcbCity**, and **jcbCountry**.
Step 5	Lastly, add a new **JTable** to the form set set its **Variable Name** as **jtAddress**. Then, right-click on it, then choose **Table Contents...** and set the number of columns to 10 and the number of rows to 50.
Step 6	In the driver class, **Sakila.java**, create a new object of **AddressForm** class using its default constructor as shown in 57 – 58:

```
1  package sakila;
2
3  public class Sakila {
4      public static void main(String[] args) {
5  //         Utility.testConnection();
6  //         Actor_Utils.create_actor_table();
7  //         Actor_Utils.populate_actor_table();
8  //         Actor_Utils.read_actor_table();
9  //         ActorForm frm = new ActorForm();
10 //         frm.setVisible(true);
11
12 //         Language_Utils.create_language_table();
13 //
14 Language_Utils.populate_language_table();
15 //         Language_Utils.read_language_table();
16 //         LanguageForm frm = new LanguageForm();
17 //         frm.setVisible(true);
18
19
```

```
//        Category_Utils.create_category_table();
//
Category_Utils.populate_category_table();
//        Category_Utils.read_category_table();
//        CategoryForm frm = new CategoryForm();
//        frm.setVisible(true);

//        Film_Utils.create_film_table();
//        Film_Utils.populate_film_table();
//        Film_Utils.read_film_table();
//        FilmForm frm = new FilmForm();
//        frm.setVisible(true);


//
FilmActor_Utils.create_film_actor_table();
//
FilmActor_Utils.populate_film_actor_table();
//        FilmActor_Utils.read_film_actor_table();
//        FilmActorForm frm = new FilmActorForm();
//        frm.setVisible(true);

//
FilmCategory_Utils.create_film_category_table();
//
FilmCategory_Utils.populate_film_category_table();
//
FilmCategory_Utils.read_film_category_table();
//        FilmCategoryForm frm = new
FilmCategoryForm();
//        frm.setVisible(true);

//        Country_Utils.create_country_table();
//        Country_Utils.populate_country_table();
//        Country_Utils.read_country_table();
//        CountryForm frm = new CountryForm();
//        frm.setVisible(true);

//        City_Utils.create_city_table();
//        City_Utils.populate_city_table();
//        City_Utils.read_city_table();
//        CityForm frm = new CityForm();
//        frm.setVisible(true);

//        Address_Utils.create_address_table();
//        Address_Utils.populate_address_table();
//        Address_Utils.read_address_table();
        AddressForm frm = new AddressForm();
        frm.setVisible(true);
    }
}
```

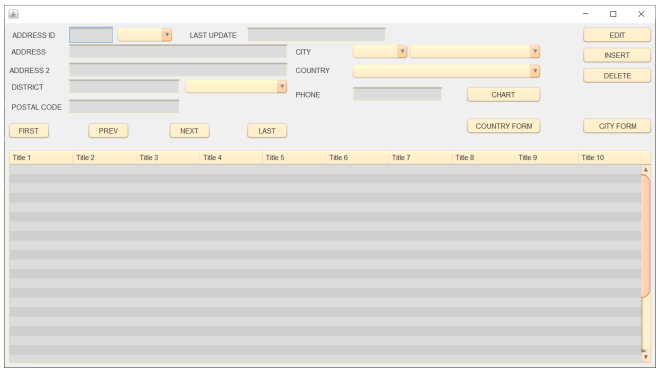


Figure 10.1 The layout of address form

Step 8

In **AddressForm**'s constructor, invoke **setLookAndFeel()** to set the look and feel of the form as shown in line 17.

```
1  package sakila;
2
3  import java.awt.Toolkit;
4  import java.awt.event.ActionEvent;
5  import
6  java.awt.event.ActionListener;
7  import javax.swing.JButton;
8  import javax.swing.JComboBox;
9  import javax.swing.JMenuItem;
10 import javax.swing.JOptionPane;
11 import javax.swing.JPopupMenu;
12 import javax.swing.JTable;
13 import javax.swing.JTextField;
14
15 public class AddressForm extends
16 javax.swing.JFrame {
17     public AddressForm() {
18         initComponents();
19
20 Utility.setLookAndFeel(this);
      }
      //...
  }
```

Run the project to see the address form as shown in Figure 10.1.

Step 9

In **AddressForm.java**, define **getter()** method for every object in the form and place them outside and beneath its default constructor:

```
    //Getter method for
jtfAddressID
    public JTextField
getJTFAddressID(){
        return this.jtfAddressID;
    }

    //Getter method for
jcbAddressID
    public JComboBox
getJCBAddressID(){
        return this.jcbAddressID;
    }

    //Getter method for jtfAddress
    public JTextField
getJTFAddress(){
        return this.jtfAddress;
    }

    //Getter method for jtfAddress2
    public JTextField
getJTFAddress2(){
        return this.jtfAddress2;
    }

    //Getter method for jtfDistrict
    public JTextField
getJTFDistrict(){
        return this.jtfDistrict;
    }

    //Getter method for jcbDistrict
    public JComboBox
getJCBDistrict(){
        return this.jcbDistrict;
    }

    //Getter method for
jtfPostalCode
    public JTextField
getJTFPostalCode(){
        return this.jtfPostalCode;
    }

    //Getter method for
jtfLastUpdate
    public JTextField
getJTFLastUpdate(){
        return this.jtfLastUpdate;
    }

    //Getter method for jcbCityID

```

```
55      public JComboBox getJCBCityID()
56  {
57          return this.jcbCityID;
58      }
59
60      //Getter method for jcbCity
61      public JComboBox getJCBCity(){
62          return this.jcbCity;
63      }
64
65      //Getter method for jcbCountry
66      public JComboBox
67  getJCBCountry(){
68          return this.jcbCountry;
69      }
70
71      //Getter method for jtfPhone
72      public JTextField getJTFPhone()
73  {
74          return this.jtfPhone;
75      }
76
77      //Getter method for jbEdit
78      public JButton getJBEdit(){
79          return this.jbEdit;
80      }
81
82      //Getter method for jbInsert
83      public JButton getJBInsert(){
84          return this.jbInsert;
85      }
86
87      //Getter method for jbDelete
88      public JButton getJBDelete(){
89          return this.jbDelete;
90      }
91
92      //Getter method for pbChart
93      public JButton getJBChart(){
94          return this.pbChart;
95      }
96
97      //Getter method for jbFirst
98      public JButton getJBFirst(){
99          return this.jbFirst;
100     }
101
102     //Getter method for jbPrev
103     public JButton getJBPrev(){
104         return this.jbPrev;
105     }
106
107     //Getter method for jbNext
108
```

```
109     public JButton getJBNext(){
110         return this.jbNext;
111     }
112 
113     //Getter method for jbLast
114     public JButton getJBLast(){
        return this.jbLast;
    }

    //Getter method for jtAddress
    public JTable getJTAddress(){
        return this.jtAddress;
    }

    //Getter method for
jbCountryForm
    public JButton
getJBCountryForm(){
        return this.jbCountryForm;
    }

    //Getter method for jbCityForm
    public JButton getJBCityForm(){
        return this.jbCityForm;
    }
```

POPULATING TABLE AND COMBOBOXES
POPULATING TABLE AND COMBOBOXES

Step 1

In **Address_Utils.java**, add two new methods: **get_address_list()** and **sh**

The first method, **get_address_list()**, is a private static method that take object, a SQL query string, and an item string. It returns an **ArrayList** of *A*

Here is what the method does:

1. It creates a new **ArrayList<Address>** object called **list**.
2. It tries to connect to a database using the **getConnection()** metho
3. It creates a new **PreparedStatement** object called ps using t parameter.
4. If the item string is not equal to **"none"**, it sets the first parameter
5. It executes the SQL query using the **executeQuery()** method and called **rs**.
6. It creates a new **Address** object for each row in the result se "address_id", "address", "address2", "district", "city_id", "postal_

"last_update".
7. It adds each **Address** object to the list.
8. If an **SQLException** occurs, it displays an error message dialog bo
9. It returns the list.

The second method, **show_table_address()**, is a private static met **AddressForm** object and an **ArrayList** of **Address** objects. It does not ret

Here is what the method does:

1. It creates a new **DefaultTableModel** object called model with zer
2. It creates an array of strings called header containing the column
3. It calls a **set_column_header()** method with the **jtAddress** obje the header array as parameters, and sets the column identifiers of
4. It sets the table model of the **jtAddress** object in the **AddressFor**
5. It creates a new array of objects called row with length 10.
6. It loops through each **Address** object in the list.
7. It sets each element of the row array to the corresponding value fr
8. It adds the row to the model.
9. It returns nothing.

```
1     private static ArrayList<Address>
2 get_address_list(AddressForm frm, String sql, String item){
3         ArrayList<Address> list = new ArrayList<>();
4 
5         try(Connection conn = getConnection();
6             PreparedStatement ps = conn.prepareStatement(sql)){
7             if (item.equalsIgnoreCase("none")==false) {
8                  ps.setString(1,item);
9             }
10            ResultSet rs = ps.executeQuery();
11 
12            Address obj;
13            while(rs.next()){
14                //Using ten-params constructor
15                obj = new Address(rs.getInt("address_id"),
16                        rs.getString("address"),
17                        rs.getString("address2"),
18                        rs.getString("district"),
19                        rs.getInt("city_id"),
20                        rs.getString("postal_code"),
21                        rs.getString("phone"),
22                        rs.getString("city"),
23                        rs.getString("country"),
24                        rs.getTimestamp("last_update"));
25 
26                list.add(obj);
27            }
28        }catch (SQLException ex){
29            JOptionPane.showMessageDialog(frm, ex.getMessage(),
30                "ERROR",JOptionPane.ERROR_MESSAGE);
```

```
        }
        return list;
    }

    private static void show_table_address(AddressForm frm,
ArrayList<Address> list) throws SQLException{
        DefaultTableModel model = new DefaultTableModel(0,0);

        String header[] = {"Address ID", "Address", "Address
2", "District",
                "City ID", "Postal Code", "Phone", "City",
"Country", "Last Update"};


model.setColumnIdentifiers(set_column_header(frm.getJTAddress(),
header));
        frm.getJTAddress().setModel(model);

        Object[] row = new Object[10];

        for(int i=0; i<list.size(); i++){
            row[0] = list.get(i).getAddressID();
            row[1] = list.get(i).getAddress();
            row[2] = list.get(i).getAddress2();
            row[3] = list.get(i).getDistrict();
            row[4] = list.get(i).getCityID();
            row[5] = list.get(i).getPostalCode();
            row[6] = list.get(i).getPhone();
            row[7] = list.get(i).getCity();
            row[8] = list.get(i).getCountry();
            row[9] = list.get(i).getLastUpdate();

            model.addRow(row);
        }
    }
```

Step 2

In **Address_Utils.java**, define **refresh_controls()** method. It is a **AddressForm** object as an argument. It is used to update the UI control data from the database.

Here are the steps involved in the refresh_controls method:

1. Set the location and title of the **AddressForm**.
2. Call the **table_renderer()** method, which sets the alternating row
3. Call the **get_address_list()** method with the **AddressForm**, a S "none" as arguments to retrieve a list of Address objects from the method is then called with the **AddressForm** and the list of **Addr** the table in the UI.

4. Call the **populate_combobox()** method with a SQL query st
   AddressForm as arguments to populate the **JComboBox** controls f
   and country.
5. If there is a **SQLException**, display an error message dialog box w

The **refresh_controls()** method essentially updates the UI controls on t
from the database, allowing the user to interact with the latest data.

```
    public static void
refresh_controls(AddressForm frm){

frm.setLocationRelativeTo(null);
        frm.setTitle("ADDRESS FORM");

        //Shows the content of
address table and populates combobox
        try{
            //Makes alternating color
for table rows

table_renderer(frm.getJTAddress());

            //Populates table
            ArrayList<Address> list =
get_address_list(frm,
Query_Address.get_sql_address_joint()
+ " ORDER BY address_id", "none");
            show_table_address(frm,
list);

            //Populates jcbAddressID
            String sql_add_id =
 "SELECT address_id FROM address
ORDER BY address_id";

populate_combobox(sql_add_id,
frm.getJCBAddressID(), frm);

            //Populates jcbDistrictID
            String sql_dist = "SELECT
DISTINCT district FROM address ORDER
BY district";

populate_combobox(sql_dist,
frm.getJCBDistrict(), frm);

            //Populates jcbCityID
            String sql_city_id =
 "SELECT city_id FROM city ORDER BY
city_id";

populate_combobox(sql_city_id,
```

```
frm.getJCBCityID(), frm);

            //Populates jcbCity
            String sql_city = "SELECT
DISTINCT city FROM city ORDER BY
city";

populate_combobox(sql_city,
frm.getJCBCity(), frm);

            //Populates
getJCBCountry()
            String sql_ct = "SELECT
DISTINCT country FROM country ORDER
BY country";
            populate_combobox(sql_ct,
frm.getJCBCountry(), frm);

        }catch (SQLException ex){

JOptionPane.showMessageDialog(frm,
ex.getMessage(),

 "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }
```

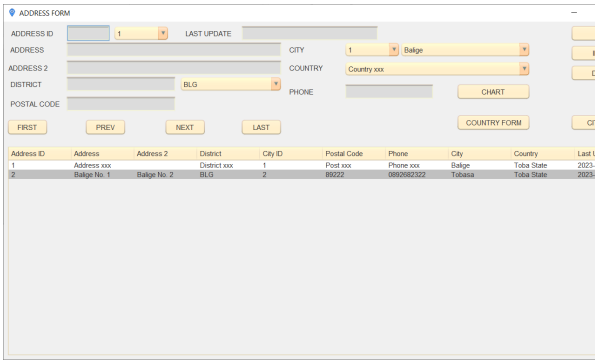


Figure 10.2 The content of joined **address**, **country** and **city** ta

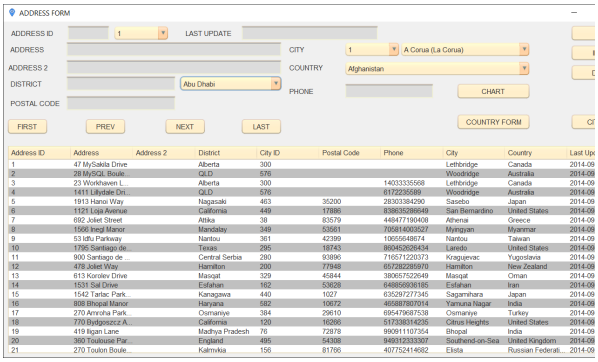


Figure 10.3 The the content of joined **address**, **country** and **city** tables i
the internet displayed in **jtAddress**

Step 3

In **AddressForm**'s default constructor, the **Address_Utils.refresh_c** populates the controls in the **AddressForm** with data from a database from the **Address_Utils** class.

The **this.setIconImage()** method sets the icon of the **A** **this.setDefaultCloseOperation(this.HIDE_ON_CLOSE)** method sets th to hide the form instead of exiting the application when the close button is

```
1  package sakila;
2  import java.awt.Toolkit;
3  import java.awt.event.ActionEvent;
4  import java.awt.event.ActionListener;
5  import java.io.IOException;
6  import java.text.ParseException;
7  import java.util.logging.Level;
8  import java.util.logging.Logger;
9  import javax.swing.JButton;
10 import javax.swing.JComboBox;
11 import javax.swing.JLabel;
12 import javax.swing.JTable;
13 import javax.swing.JTextField;
14
15 public class AddressForm extends javax.swing.JFrame {
16     public AddressForm() {
17         initComponents();
18         Utility.setLookAndFeel(this);
19         Address_Utils.refresh_controls(this);
20
21  this.setIconImage(Toolkit.getDefaultToolkit().getImage(getClass().
22 ;
23         this.setDefaultCloseOperation(this.HIDE_ON_CLOSE);
24     }
25     //...
26 }
```

Step 4

Run the project to see the content of joined **address**, **country** and **city** ta in Figure 10.2.

If you use the data from **Sakila** MySQL database available in the internet **address**, **country** and **city** tables displayed in **jtAddress** as shown in Figu

DISPLAYING AND NAVIGATING DATA ROW BY ROW
DISPLAYING AND NAVIGATING DATA ROW BY ROW

Step 1

In **Address_Utils**, define a new method named **display_country_city_da** an **AddressForm** object, an SQL query, and an item as input parameters the item parameter to set a parameter in the SQL query and executes the a database connection.

If the query returns no results, the method clears the selection of three **J** (**jcbCityID**, **jcbCity**, and **jcbCountry**) in the **AddressForm** object Otherwise, the method loops through the result set and sets the selecte **JComboBoxes** based on the values in the result **find_combo_value_selected()** method is called to set the selected it **JComboBox**.

```
1   //Displays joined country and city data
2  result row by row
3      private static <T> void
4  display_country_city_data(AddressForm frm,
5  String sql, T item){
6          try(Connection conn = getConnection())
7  {
8              PreparedStatement ps =
9  conn.prepareStatement(sql);
10             ps.setObject(1,item);
11             ResultSet rs = ps.executeQuery();
12
13             if (!rs.next()) {
14                 // no row found, clear the
15 form fields
16
17 frm.getJCBCityID().setSelectedIndex(-1);
18
19 frm.getJCBCity().setSelectedIndex(-1);
20
21 frm.getJCBCountry().setSelectedIndex(-1);
22                 return;
23             }
24
25             do{
26                 // Determines item selected
27 from jcbCityID
28
29 find_combo_value_selected(frm.getJCBCityID(),
30 rs.getInt("city_id"));
31
32                 // Determines item selected
33 from jcbCity
34
35 find_combo_value_selected(frm.getJCBCity(),
36 rs.getString("city"));
```

```
                // Determines item selected
from jcbCountry

find_combo_value_selected(frm.getJCBCountry(),
rs.getString("country"));
            }while(rs.next());

            rs.close();
            ps.close();
        }catch(SQLException ex){
            JOptionPane.showMessageDialog(frm,
ex.getMessage(),

 "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }
```

Step 2

Still in the same class, define another method named **jcbCity_handler()**. I event when a **JComboBox** (**jcb**) selection changes. It takes an **Addres** and a **JComboBox** as input parameters. It retrieves the currently select the **JComboBox** and determines whether it is **jcbCityID** or **jcbCity** by with the corresponding **JComboBox** in the **AddressForm** object.

Depending on the selected **JComboBox**, the method creates an SQL qu **get_sql_city_joint()** method from the **Query_City** class and adds a WHE filter the query based on the selected item. The SQL query is passed al selected item and **AddressForm** object to the **display_country_city_da** which retrieves the country and city data from the database and displays using the **find_combo_value_selected()** method.

```
1    public static void
2  jcbCity_handler(AddressForm frm,
3  JComboBox<String> jcb) {
4          Object item =
5  jcb.getSelectedItem();
6          String sql = "";
7          if
8  (jcb.equals(frm.getJCBCityID())) {
9              sql =
10 Query_City.get_sql_city_joint() + "
11 WHERE city_id = ?";
           } else if
   (jcb.equals(frm.getJCBCity())) {
               sql =
   Query_City.get_sql_city_joint() + "
   WHERE city = ?";
           }
```

```
display_country_city_data(frm, sql,
item);
    }
```

Step 3

In **AddressForm**, double click on **jcbCityID** and **jcbCity** comboboxes an corresponding event handler as follows:

```
    private void
jcbCityActionPerformed(java.awt.event.ActionEvent
evt) {
        Address_Utils.jcbCity_handler(this,
this.jcbCity);
    }

    private void
jcbCityIDActionPerformed(java.awt.event.ActionEvent
evt) {
        Address_Utils.jcbCity_handler(this,
this.jcbCityID);
    }
```

These are Java event handling methods that are triggered when the select **jcbCity** or **jcbCityID** JComboBox is changed. Both methods call the **jcbCi** method from the **Address_Utils** class, passing in the current **AddressFor** the corresponding **JComboBox** as input parameters.

The **jcbCity_handler()** method retrieves the currently selected ite **JComboBox** and creates an SQL query based on the selected **JComboBox** the **display_country_city_data()** method to retrieve and display the c country and city data in the GUI based on the SQL query and selected item

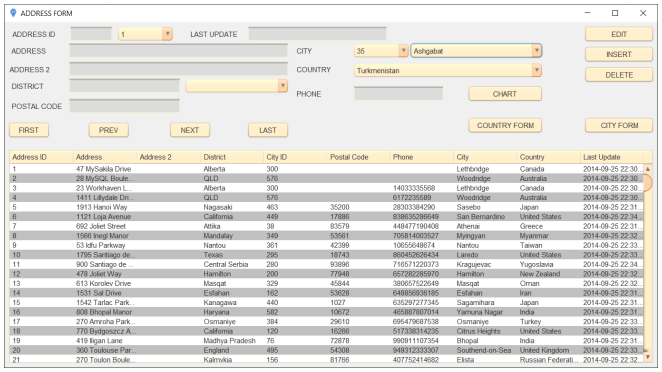


Figure 10.4 Displaying row by row the content of joined **city** and **cour**

Step 4

Run the project. Choose one of items in **jcbCityID** and/or **jcbCity** comb row by row the content of joined **city** and **country** tables as shown in Figu

Step 5	In **Address_Utils** class, define two new methods named **clear_co** **display_address_data()**. These are two methods from the **Address_Ut** are used to display data in the **AddressForm** GUI. The **clear_controls()** method simply clears all the text fields and combo **AddressForm** GUI. The **display_address_data()** method retrieves data from the database SQL query and a selected item, and then displays the data in the corre fields and combo boxes in the **AddressForm**. This method first con database and executes the SQL query with the given item parameter. returns no rows, the method calls **clear_controls()** to clear the form field returns one or more rows, the method loops through each row and retriev for each field in the **AddressForm** GUI. Finally, the method sets the sele each combo box based on the retrieved data.

```
1     private static void clear_controls(AddressForm frm){
2         frm.getJTFAddressID().setText("");
3         frm.getJTFAddress().setText("");
4         frm.getJTFLastUpdate().setText("");
5         frm.getJTFAddress2().setText("");
6         frm.getJTFDistrict().setText("");
7         frm.getJTFPhone().setText("");
8         frm.getJTFPostalCode().setText("");
9         frm.getJTFAddress2().setText("");
10
11        frm.getJCBCountry().setSelectedIndex(-1);
12        frm.getJCBCityID().setSelectedIndex(-1);
13        frm.getJCBCity().setSelectedIndex(-1);
14        frm.getJCBCountry().setSelectedIndex(-1);
15    }
16
17   //Displays address data result row by row
18    private static <T> void display_address_data(AddressForm frm, S
19 item){
20        try(Connection conn = getConnection()){
21            PreparedStatement ps = conn.prepareStatement(sql);
22            ps.setObject(1,item);
23            ResultSet rs = ps.executeQuery();
24
25            if (!rs.next()) {
26                // no row found, clear the form fields
27                clear_controls(frm);
28                return;
29            }
30
31            do{
32
33 frm.getJTFAddressID().setText(String.valueOf(rs.getInt("address_id"
```

```
34                  frm.getJTFAddress().setText(rs.getString("address")
35                  frm.getJTFAddress2().setText(rs.getString("address2
36                  frm.getJTFDistrict().setText(rs.getString("district
37                  frm.getJTFPhone().setText(rs.getString("phone"));
38                  frm.getJTFPostalCode().setText(rs.getString("postal
39
40  frm.getJTFLastUpdate().setText(String.valueOf(rs.getTimestamp("last
41
42                  // Determines item selected from jcbAddressID
43                  find_combo_value_selected(frm.getJCBAddressID(),
44  rs.getInt("address_id"));
45
46                  // Determines item selected from jcbDistrict
47                  find_combo_value_selected(frm.getJCBDistrict(),
48  rs.getString("district"));
49
50                  // Determines item selected from jcbCityID
51                  find_combo_value_selected(frm.getJCBCityID(),
52  rs.getInt("city_id"));
53
54                  // Determines item selected from jcbCity
55                  find_combo_value_selected(frm.getJCBCity(),
56  rs.getString("city"));
57
58              }while(rs.next());
59
60              rs.close();
62              ps.close();
62          }catch(SQLException ex){
63              JOptionPane.showMessageDialog(frm, ex.getMessage(),
64                      "ERROR",JOptionPane.ERROR_MESSAGE);
65          }
      }
```

Step 6

In the same class, define another method named **jcbAddress_handler()** handle events that occur when the user selects an item from the **JCo** address ID.

The method first gets the selected item from the **JComboBox** and then parameter to the **display_address_data()** method along with the SQL The **display_address_data()** method uses the selected item to corresponding address data from the database and display it on the *A* object.

```
1       public static void
2   jcbAddress_handler(AddressForm frm) {
3           Object item =
4   frm.getJCBAddressID().getSelectedItem();
            display_address_data(frm,
    SQL_ID, item);
```

```
    }
```

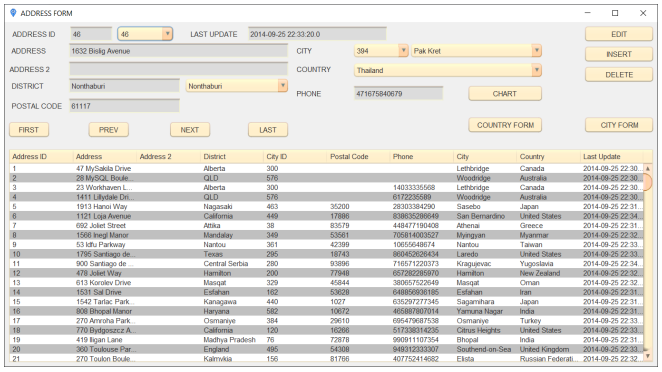


Figure 10.5 Displaying row by row the content of joined **address**, **city** a tables

Step 7

In **AddressForm**, double click on **jcbAddressID** combobox and corresponding event handler as follows:

```
    private void jcbAddressIDActionPerformed(java.awt.event.ActionEvent evt) {
        Address_Utils.jcbAddress_handler(this);
    }
```

Step 8

Run the project. Choose one of items in **jcbAddressID** combobox to see rc content of joined **address**, **city** and **country** tables as shown in Figure 10.

Step 9

Define four navigating methods in **Address_Utils** class. These methods implement a basic navigation functionality for moving through the rows c data displayed in the **AddressForm**.

show_first_row() sets the item selected in the **jcbAddressID** combo bc item, calls **display_address_data()** to display the corresponding row of c **currentIndex** to the index of the first item.

show_last_row() sets the item selected in the **jcbAddressID** combo bc item, calls **display_address_data** to display the corresponding row of d **currentIndex** to the index of the last item.

show_prev_row() decrements currentIndex by 1, checks if it is less thar the first item, and if so, sets **currentIndex** back to the index of the first sets the item selected in the **jcbAddressID** combo box to the item at the c and calls **display_address_data** to display the corresponding row of data.

show_next_row() increments currentIndex by 1, checks if it is greater tl
of the last item, and if so, sets **currentIndex** back to the index of the last
sets the item selected in the **jcbAddressID** combo box to the item at the c
and calls **display_address_data** to display the corresponding row of data.

```
1     public static void show_first_row(AddressForm frm){
2         String item =
3 String.valueOf(frm.getJCBAddressID().getItemAt(FIRST_INDEX));
4         display_address_data(frm, SQL_ID, item);
5         currentIndex = FIRST_INDEX;
6     }
7
8     public static void show_last_row(AddressForm frm){
9         int endIndex = frm.getJCBAddressID().getItemCount() -
10 1;
11        String item =
12 String.valueOf(frm.getJCBAddressID().getItemAt(endIndex));
13        display_address_data(frm, SQL_ID, item);
14        currentIndex = endIndex;
15    }
16
17    public static void show_prev_row(AddressForm frm){
18        currentIndex--;
19        if(currentIndex < FIRST_INDEX){
20            currentIndex = FIRST_INDEX;
21            return;
22        }
23        String item =
24 String.valueOf(frm.getJCBAddressID().getItemAt(currentIndex));
25        display_address_data(frm, SQL_ID, item);
26    }
27
28    public static void show_next_row(AddressForm frm){
29        int endIndex = frm.getJCBAddressID().getItemCount() -
30 1;
31        currentIndex++;
32        if(currentIndex > endIndex){
33            currentIndex = endIndex;
34            return;
35        }
36        String item =
  String.valueOf(frm.getJCBAddressID().getItemAt(currentIndex));
  display_address_data(frm, SQL_ID, item);
  }
```

Step 10

Then in **AddressForm**, double click on each navigation buttons to corresponding event handler:

```
    private void
jbLastActionPerformed(java.awt.event.ActionEvent
evt) {
        Address_Utils.show_last_row(this);
    }

    private void
jbNextActionPerformed(java.awt.event.ActionEvent
evt) {
        Address_Utils.show_next_row(this);
    }

    private void
jbPrevActionPerformed(java.awt.event.ActionEvent
evt) {
        Address_Utils.show_prev_row(this);
    }

    private void
jbFirstActionPerformed(java.awt.event.ActionEvent
evt) {
        Address_Utils.show_first_row(this);
    }
```

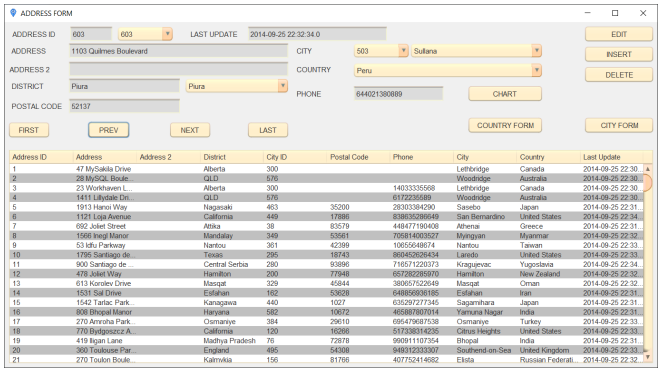


Figure 10.6 User clicks on one or more navigation buttons on addre

These methods are associated with the "First", "Previous", "Next", and " respectively. When clicked, each button invokes its corresponding meth the first, previous, next, or last row of data in the **AddressForm**.

Step 11	Run the project. Click on one or more navigation buttons to see the resul Figure 10.6.
Step 12	Define **mouse_pressed_handler()** method in **Address_Utils** class. handles the event when the user clicks on a row in the table (**frm.getJTAddress()**). It first checks if a row is selected, and if not, message informing the user to select a row. If a row is selected, it gets th

the address from the first column of the selected row, and the **display_address_data()** method to display the data for that address in the

The **display_address_data()** method is called with the **SQL_ID** cons selected ID as arguments, which retrieves the data for that address from and populates the form fields with the data. If an **SQLException** occur process, an error message is displayed with the details of the error.

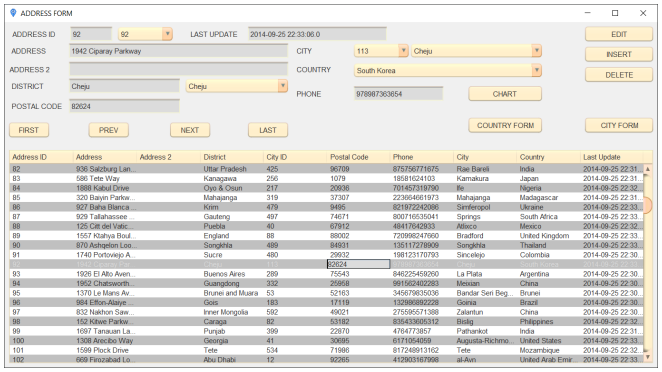


Figure 10.7 User double-clicks on any row in **jtAddress**

```
    public static void mouse_pressed_handler(AddressForm frm) {
        Objects.requireNonNull(frm, "frm must not be null");

        int selectedIndex = frm.getJTAddress().getSelectedRow();
        if (selectedIndex == -1) {
            JOptionPane.showMessageDialog(frm, "Please select a row to view its
data.",
                "No row selected", JOptionPane.INFORMATION_MESSAGE)
            return;
        }

        try (Connection conn = getConnection()) {
            String id =
String.valueOf(frm.getJTAddress().getModel().getValueAt(selectedInd
0));

            // Displays address data
            display_address_data(frm, SQL_ID, id);

        } catch (SQLException ex) {

Logger.getLogger(AddressForm.class.getName()).log(Level.SEVERE, "Er
displaying address data", ex);
            String message = "Error displaying address data: " +
ex.getMessage();
            String stackTrace = Arrays.toString(ex.getStackTrace())
            JOptionPane.showMessageDialog(frm, message + "\n\n" +
stackTrace, "ERROR", JOptionPane.ERROR_MESSAGE);
        }
    }
```

Step 13	Right click on **jtAddress**. Then, choose **Events > Mouse > mousePress** event handler: `1 2 3` `private void jtAddressMousePressed(java.awt.event.MouseEvent evt) { Address_Utils.mouse_pressed_handler(this); }`
Step 14	Run the project. Double click on any row in **jtAddress** table. You corresponding row of joined **address**, **country** and **city** tables displayed and comboboxes as shown in Figure 10.7.

UPDATING RECORD
UPDATING RECORD

Step 1	In **Address_Utils** class, define a new method named **update_row_by_address_id()**. This method updates a row in the **address** table in the database, identified by the given **add_id**. It takes in the following parameters: • **frm**: An instance of the **AddressForm** class. • **add_id**: An integer representing the **address_id** of the row to be updated. • **add**: A string representing the updated address. • **add2**: A string representing the updated secondary address. • **dist**: A string representing the updated district. • **city_id**: An integer representing the updated city ID. • **post**: A string representing the updated postal code. • **ph**: A string representing the updated phone number. The method starts by getting a connection to the database and preparing two statements - one to check if the given **add_id** exists in the **address** table, and the other to update the row with the given data. The **PreparedStatement** objects are then used to execute the queries, and if the **add_id** is found in the table, the row is updated with the new data.

If an **SQLException** or a **NumberFormatException** occurs, an error message is logged, and a **JOptionPane** is displayed with the error message and stack trace.

```
    //Updates row of data in address tabel by address_id
    public static void update_row_by_address_id(AddressForm
frm, int add_id, String add, String add2, String dist,
            int city_id, String post, String ph) throws
SQLException{
        Connection conn = getConnection();
        ResultSet rs = null;
        String query_id = "SELECT address_id FROM address WHERE
address_id = ?";
        String update_query = """
            UPDATE address SET address = ?, address2 = ?,
district = ?,
            city_id = ?, postal_code = ?, phone = ? WHERE
address_id = ?""";
        try(PreparedStatement idPs =
conn.prepareStatement(query_id,

ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_READ_ONLY);
        PreparedStatement updatePS =
conn.prepareStatement(update_query,
                ResultSet.TYPE_SCROLL_SENSITIVE,
ResultSet.CONCUR_UPDATABLE))
        {
            idPs.setInt(1,add_id);
            if(!idPs.execute()){
                String message = "Can't find address_id " +
add_id;

                JOptionPane.showMessageDialog(frm, message,
                    "ERROR",JOptionPane.ERROR_MESSAGE);
            } else{
                rs = idPs.getResultSet();
                rs.next();

                //Creates a Address object using eight-params
constructor
                Address obj = new Address(add_id, add, add2,
dist, city_id, post, ph, new
Timestamp(System.currentTimeMillis()));
                updatePS.setString(1, obj.getAddress());
                updatePS.setString(2, obj.getAddress2());
                updatePS.setString(3, obj.getDistrict());
                updatePS.setInt(4, obj.getCityID());
                updatePS.setString(5, obj.getPostalCode());
                updatePS.setString(6, obj.getPhone());
                updatePS.setInt(7, obj.getAddressID());

                updatePS.executeUpdate();
                rs.close();
```

```
50                updatePS.close();
55                idPs.close();
56                conn.close();
57            }
58        }catch(SQLException ex){
59
60 Logger.getLogger(AddressForm.class.getName()).log(Level.SEVERE,
61 "Error updating address data", ex);
62            String message = "Error updating address data: " +
   ex.getMessage();
            String stackTrace =
   Arrays.toString(ex.getStackTrace());
            JOptionPane.showMessageDialog(null, message +
   "\n\n" + stackTrace, "ERROR", JOptionPane.ERROR_MESSAGE);
        }catch(java.lang.NumberFormatException ex){

   Logger.getLogger(AddressForm.class.getName()).log(Level.SEVERE,
   "Invalid Input", ex);
            String message = "Invalid Input: " +
   ex.getMessage();
            String stackTrace =
   Arrays.toString(ex.getStackTrace());
            JOptionPane.showMessageDialog(null, message +
   "\n\n" + stackTrace, "ERROR", JOptionPane.ERROR_MESSAGE);
        }
    }
```

Step 2

Then in the same class, define a new method **read_inputs()**. This method reads user input data from the **AddressForm** and returns it in a **HashMap**. It also performs some validation on the input data.

The input data is obtained by calling various methods on the **AddressForm** object, such as **getJCBAddressID()**, **getJTFAddress()**, etc. The input is validated by checking that certain fields are not empty and that the address ID and city ID are valid integers. If the validation fails, an exception is thrown and an error message is displayed to the user using **JOptionPane**.

The validated input is then stored in a **HashMap**, with each key-value pair representing a specific piece of input data. The **HashMap** is then returned to the calling method.

```
1     private static HashMap<String, String>
2 read_inputs(AddressForm frm) {
3         HashMap<String, String> input_data = new
4 HashMap<>();
5         String add_id =
6 String.valueOf(frm.getJCBAddressID().getSelectedItem());
```

```
        String city_id =
String.valueOf(frm.getJCBCityID().getSelectedItem());
        String add = frm.getJTFAddress().getText();
        String add2 = frm.getJTFAddress2().getText();
        String dist = frm.getJTFDistrict().getText();
        String ph = frm.getJTFPhone().getText();
        String post = frm.getJTFPostalCode().getText();

        // Validate user input
        int add_id_int = 0;
        try {
            add_id_int = Integer.parseInt(add_id);
            if (add_id_int <= 0) {
                throw new
IllegalArgumentException("Address ID cannot be negative
or zero");
            }
        } catch (NumberFormatException ex) {
            JOptionPane.showMessageDialog(frm, "Invalid
Address ID: " + add_id,
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        } catch (IllegalArgumentException ex) {
            JOptionPane.showMessageDialog(frm,
ex.getMessage(),
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        }

        int city_id_int = 0;
        try {
            city_id_int = Integer.parseInt(city_id);
            if (city_id_int <= 0) {
                throw new IllegalArgumentException("City ID cannot
be negative or zero");

            }
        } catch (NumberFormatException ex) {
            JOptionPane.showMessageDialog(frm, "Invalid
City ID: " + city_id,
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        } catch (IllegalArgumentException ex) {
            JOptionPane.showMessageDialog(frm,
ex.getMessage(),
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        }

        if (add == null || add.isEmpty()) {
            JOptionPane.showMessageDialog(frm, "Address
cannot be empty",
            "Error", JOptionPane.ERROR_MESSAGE);
            throw new IllegalArgumentException("Address
```

```
cannot be empty");
        }

        if (dist == null || dist.isEmpty()) {
            JOptionPane.showMessageDialog(frm, "District
cannot be empty",
            "Error", JOptionPane.ERROR_MESSAGE);
            throw new IllegalArgumentException("District
cannot be empty");
        }

        if (ph == null || ph.isEmpty()) {
            JOptionPane.showMessageDialog(frm, "Phone
cannot be empty",
            "Error", JOptionPane.ERROR_MESSAGE);
            throw new IllegalArgumentException("Phone
cannot be empty");

        }

        input_data.put("add_id", add_id);
        input_data.put("city_id", city_id);
        input_data.put("add", add);
        input_data.put("add2", add2);
        input_data.put("dist", dist);
        input_data.put("ph", ph);
        input_data.put("post", post);

        return input_data;
    }
```

Step 3

Still in the same class, define another method named **edit_actual()**. It responsible for editing an existing address in the database. It first reads the input data from the form using the read_inputs method, which returns a **HashMap** containing the values entered by the user. It then parses the relevant fields from the **HashMap**, and passes them to the **update_row_by_address_id()** method, which updates the corresponding row in the database with the new values.

If the update is successful, the method calls **refresh_controls()** to update the controls on the form. If an **SQLException** occurs during the update, it displays an error message to the user.

```
    private static void
edit_actual(AddressForm frm){
        try{
            HashMap<String, String>
input_data = read_inputs(frm);
            int add_id =
Integer.parseInt(input_data.get("add_id"));
```

```
            int city_id =
Integer.parseInt(input_data.get("city_id"));
            String add =
input_data.get("add");
            String add2 =
input_data.get("add2");
            String dist =
input_data.get("dist");
            String post =
input_data.get("post");
            String ph =
input_data.get("ph");

            update_row_by_address_id(frm,
add_id, add, add2, dist, city_id, post, ph);

            //Refreshes all objects on form
            refresh_controls(frm);

        }catch(SQLException ex){

JOptionPane.showMessageDialog(frm,
ex.getMessage(),

 "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }
```

Step 4

Lastly, define two new methods named **enable_controls()** and **edit_handler()**. When the button is first clicked, **edit_handler()** changes its text to "CONFIRM" and disables all other controls on the form by calling the **enable_controls()** method with a state of false. When the button is clicked again, it changes its text back to "EDIT", calls the **edit_actual()** method to update the database with the changes made in the form, and then enables the controls again by calling **enable_controls()** with a state of true.

```
    private static void
enable_controls(boolean state, AddressForm
frm){
        frm.getJBFirst().setEnabled(state);
        frm.getJBPrev().setEnabled(state);
        frm.getJBNext().setEnabled(state);
        frm.getJBLast().setEnabled(state);
        frm.getJBInsert().setEnabled(state);
        frm.getJBDelete().setEnabled(state);

frm.getJTFAddressID().setEnabled(state);

frm.getJCBCountry().setEnabled(state);
```

```
    }

    public static void
edit_handler(AddressForm frm){

 if(frm.getJBEdit().getText().equals("EDIT")
){

frm.getJBEdit().setText("CONFIRM");

            // Disables controls
            enable_controls(false, frm);
        }

        else {
            frm.getJBEdit().setText("EDIT");

            // Actual editing
            edit_actual(frm);

            //Enables controls
            enable_controls(true, frm);
        }
    }
```

Step 5

Run the project. Choose **address_id** using **jcbAddressID** combobox. Or, you can choose one of rows in **jtAddress** (in this case, **address_id = 4**). Then, click on EDIT button as shown in Figure 10.8.

Edit anything you want. Then, click on CONFIRM button. The edited row had been saved into **address** table as shown in Figure 10.9.

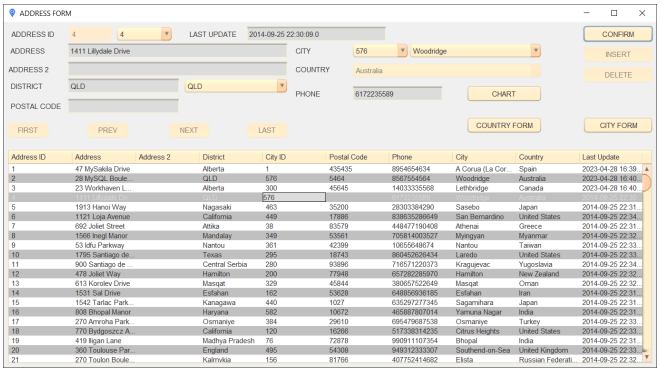


Figure 10.8 The address form is in editing state

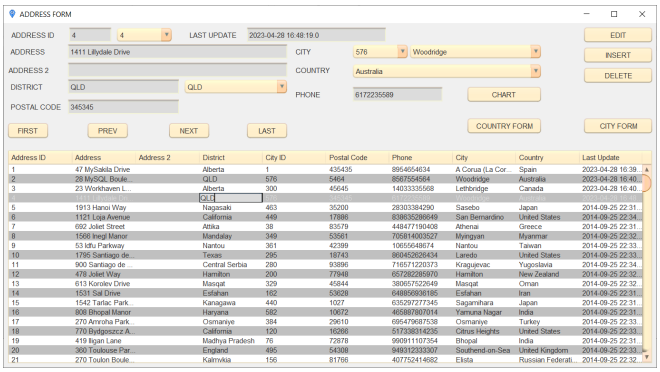


Figure 10.9 The edited row had been saved into database

UPDATING RECORD DIRECTLY ON JTABLE
UPDATING RECORD DIRECTLY ON JTABLE

Step 1

In **Address_Utility** class, define a new method r **edit_database_from_jtable()**. This method handles the updating c database based on changes made in the **JTable** displayed on the **Address** It listens for updates to the table model, and when an update occurs, it ret the updated values from the model and updates the corresponding row table.

First, it checks if the event type is a table update. If it is, it gets the ro model of the updated cell. Then, it retrieves the updated values from the and stores them in variables. Finally, it calls the **update_row_by_addres** method to update the corresponding row in the database with the new valu

If there is an exception during the update process, it logs the exceptio displays an error message to the user through a **JOptionPane**. Po exceptions include **SQLIntegrityConstraintViolationException** (if the u violates a unique constraint in the database), **SQLSyntaxErrorExcepti** the update contains invalid SQL syntax), and **SQLException** (for g database errors).

```
1     public static void
2 edit_database_from_jtable(TableModelEvent e, AddressForm frm){
3         if (e.getType() == TableModelEvent.UPDATE) {
4             int row = e.getFirstRow();
5             TableModel model = (TableModel)e.getSource();
6             int add_id =
7 Integer.parseInt(String.valueOf(model.getValueAt(row, 0)));
```

```
            String add = String.valueOf(model.getValueAt(row,
1));
            String add2 = String.valueOf(model.getValueAt(row,
2));
            String dist = String.valueOf(model.getValueAt(row,
3));
            int city_id =
Integer.parseInt(String.valueOf(model.getValueAt(row, 4)));
            String post = String.valueOf(model.getValueAt(row,
5));
            String ph = String.valueOf(model.getValueAt(row,
6));

            try{
                update_row_by_address_id(frm, add_id, add,
add2, dist, city_id, post, ph);

                //Refreshes all objects on form
                refresh_controls(frm);

            } catch (SQLIntegrityConstraintViolationException
ex) {

Logger.getLogger(AddressForm.class.getName()).log(Level.SEVERE,
"Duplicate entry", ex);
                JOptionPane.showMessageDialog(frm, "Error:
Duplicate entry\n" + ex.getMessage());
            } catch (SQLSyntaxErrorException ex) {

Logger.getLogger(AddressForm.class.getName()).log(Level.SEVERE,
"Invalid SQL syntax", ex);
                JOptionPane.showMessageDialog(frm, "Error:
Invalid SQL syntax\n" + ex.getMessage());
            } catch (SQLException ex) {

Logger.getLogger(AddressForm.class.getName()).log(Level.SEVERE,
"Database error", ex);
                JOptionPane.showMessageDialog(frm, "Error:
Database error\n" + ex.getMessage());
            }
        }
    }
```

Step 2

Create a new public class named **AddressTableModelListener**. It liste changes in the table and calls the **edit_database_from_jtable()** metho the **Address_Utils** class to update the corresponding row in the database.

The constructor takes in a **JTable** and an **AddressForm** object, which ar to pass to the **edit_database_from_jtable()** method and to stop cell editi is in progress when the table changes.

```
1  package sakila;
2  import javax.swing.event.TableModelEvent;
3  import
4  javax.swing.event.TableModelListener;
5  import javax.swing.JTable;
6
7  public class AddressTableModelListener
8  implements TableModelListener {
9      private final JTable jt;
10     private final AddressForm frm;
11
12     public
13 AddressTableModelListener(JTable jt,
14 AddressForm frm) {
15         this.jt = jt;
16         this.frm = frm;
17     }
18
19     @Override
20     public void
21 tableChanged(TableModelEvent e) {
22
23 Address_Utils.edit_database_from_jtable(e,
   frm);

         if (jt.getCellEditor() != null) {

jt.getCellEditor().stopCellEditing();
         }
     }
}
```

Step 3

Right click on **jtAddress**. Then, choose **Events > Mouse > mouseCli** Define its event handler:

```
1     private void jtAddressMouseClicked(java.awt.event.MouseEvent evt
2         // instantiate AddressTableModelListener and add it as a lis
3 to the table model
4         AddressTableModelListener tableModelListener = new
5 AddressTableModelListener(this.getJTAddress(), this);
6
7  this.getJTAddress().getModel().addTableModelListener(tableModelList
   ;
     }
```

The **jtAddressMouseClicked()** method is adding **AddressTableModelListener** to the **TableModel** of the **JTable** i

	AddressForm. The listener is created with the **JTable** and **Address** objects as arguments. The purpose of the **AddressTableModelListener** is to listen for changes data in the **JTable**, and when a change occurs, it calls **edit_database_from_jtable()** method in the **Address_Utils** class to upda corresponding record in the database. The method also checks if the cell editor is active and stops it if it is, to e that the changes made by the user are saved before the cell loses focus.
Step 4	Run the project. Click on any cell in second column to seventh column row in **jtAddress** that you want to edit. Then, change it. Then, click any outside the corresponding cell. The edited data had been saved into databa

INSERTING NEW RECORD
INSERTING NEW RECORD

Step 1	In **Address_Utils** class, define a method named **insert_row()**. This method inserts a new row into the **address** table. It first reads the input data from the form, then prepares an SQL insert statement using a prepared statement. It then creates an **Address** object with the input data and sets the prepared statement parameters with the object's properties. Finally, it executes the insert statement using the **executeUpdate()** method. If an exception occurs during the execution of the insert statement, it logs the error and shows an error message to the user using a **JOptionPane**.

```
1     //Inserts new row into address table
2     private static void insert_row(AddressForm frm) throws
3  SQLException{
4          HashMap<String, String> input_data = read_inputs(frm);
5          int city_id =
6  Integer.parseInt(input_data.get("city_id"));
7          String add = input_data.get("add");
8          String add2 = input_data.get("add2");
9          String dist = input_data.get("dist");
10         String post = input_data.get("post");
11         String ph = input_data.get("ph");
12
13         // SQL insert statement
14         String sql = """
15             INSERT INTO address(address, address2, district,
```

```
                city_id, postal_code, phone)
                VALUES(?, ?, ?, ?, ?, ?)""";

        try(Connection conn = getConnection();
         PreparedStatement pstmt = conn.prepareStatement(sql)){

            //Creates a Address object seven-params constructor
            Address obj = new Address(add, add2, dist, city_id,
post, ph, new Timestamp(System.currentTimeMillis()));
            pstmt.setString(1,obj.getAddress());
            pstmt.setString(2,obj.getAddress2());
            pstmt.setString(3,obj.getDistrict());
            pstmt.setInt(4,obj.getCityID());
            pstmt.setString(5,obj.getPostalCode());
            pstmt.setString(6,obj.getPhone());

            //Executes the sql insert statement
            pstmt.executeUpdate();
        } catch (SQLException ex) {

Logger.getLogger(AddressForm.class.getName()).log(Level.SEVERE,
"Database error", ex);
            JOptionPane.showMessageDialog(frm, "Error: Database
error\n" + ex.getMessage());
        }
    }
```

Step 2

Still in **Address_Utils.java**, define **insert_actual()** and **insert_handler()** methods. The **insert_handler()** method is responsible for handling the user's actions when inserting a new address into the database. When the user clicks the "INSERT" button, the method changes the button's text to "CONFIRM" and disables the "EDIT" button, as well as all controls on the form. It also clears the controls, except for the Address ID combo box, and enables the "CONFIRM" button.

When the user clicks the "CONFIRM" button, the method calls the **insert_actual()** method to perform the actual insertion into the database. It then restores the original state of the form, including enabling the "EDIT" button and all controls, clearing the controls, and resetting the "INSERT" button's text. If an exception is thrown during the insertion process, the method displays an error message to the user.

```
    private static void
insert_actual(AddressForm frm){
        try{
            insert_row(frm);

```

```
6               //Refreshes table and comboboxes
7               refresh_controls(frm);
8
9           }catch(SQLException ex){
10              JOptionPane.showMessageDialog(frm,
11  ex.getMessage(),
12
13   "ERROR",JOptionPane.ERROR_MESSAGE);
14          }
15      }
16
17      public static void
18  insert_handler(AddressForm frm){
19
20   if(frm.getJBInsert().getText().equals("INSERT")
21  ){
22
23  frm.getJBInsert().setText("CONFIRM");
24
25              //Disables jbEdit
26              frm.getJBEdit().setEnabled(false);
27
28              // Disables controls
29              enable_controls(false, frm);
30
31  frm.getJCBAddressID().setEnabled(false);
32
33              // Clears controls
34              clear_controls(frm);
35
36              // Enables
37
38
39
40
41
42
43
44
45
```

```
frm.getJBInsert().setEnabled(true);
        }

        else {

frm.getJBInsert().setText("INSERT");

            // Actual insertion
            insert_actual(frm);

            //Enables jbEdit

frm.getJBEdit().setEnabled(true);

            //Enables controls
            enable_controls(true, frm);

frm.getJCBAddressID().setEnabled(true);
        }
    }
```

Step 3

In **AddressForm.java**, double click to create its event listener:

```
1  private void
2  jbInsertActionPerformed(java.awt
3  evt) {
       Address_Utils.insert_har
   }
```

Step 4

Run the project. Click on INSERT t the state of address form when inse as shown in Figure 10.10.

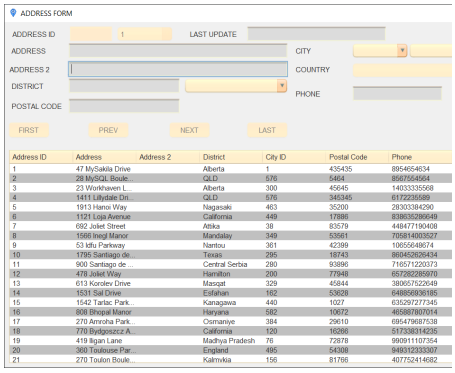


Figure 10.10 When user clicks on
address form will be in state

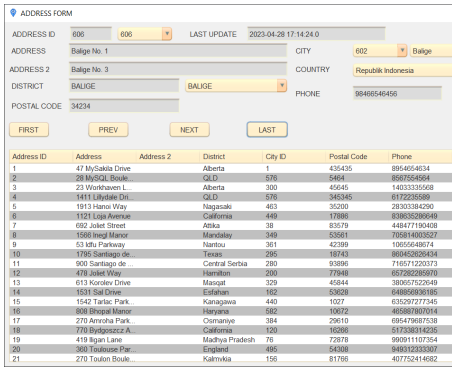


Figure 10.11 The new data had
address table

Then, fill all fields in, including c
Then, click CONFIRM button to sa
into **address** table as shown in Figu

DELETING RECORD

DELETING RECORD

Step 1

Then in **Address_Utils** class, define **delete_handler()** method. This handles the deletion of a row in the **address** table. It first gets the Addre the selected item in the **JComboBox**, and then displays a confirmation d the user. If the user selects "Yes", it executes an SQL DELETE state remove the row from the database.

The method uses a **PreparedStatement** to avoid SQL injection attac passes the Address ID as a parameter to the query. If the query is e successfully, it calls the **refresh_controls()** method to update the ta **JComboBoxes** with the latest data from the database. If there is a executing the query, it displays an error message to the user.

```
1     public static void delete_handler(AddressForm frm){
2         int dialogButton = JOptionPane.YES_NO_OPTION;
3         int add_id =
4 Integer.parseInt(String.valueOf(frm.getJCBAddressID().getSelectedIt
5
6         String message = String.format("Are you sure you want to de
7 the row Address ID: %d)", add_id);
8         int answer = JOptionPane.showConfirmDialog(frm, message, "D
9 ROW OF DATA", dialogButton);
10
11        if(answer == JOptionPane.YES_OPTION){
12            String query = """
13                DELETE FROM address WHERE address_id = ?""";
14            try(Connection conn = getConnection();
```

```
                PreparedStatement ps = conn.prepareStatement(query)
                // Use PreparedStatement to avoid SQL injection att
                ps.setInt(1, add_id);
                ps.executeUpdate();

                // Refresh table and comboboxes
                refresh_controls(frm);

            } catch (SQLException ex){
                JOptionPane.showMessageDialog(frm, ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
            }
        }
    }
```

Step 2 — In **AddressForm.java**, double click on DELETE button to generate it listener:

```
    private void
jbDeleteActionPerformed(java.awt.event.ActionEvent
evt) {
        Address_Utils.delete_handler(this);
    }
```

Step 3 — Run the project. Choose **address_id** using **jcbAddressID** combobox. The on DELETE button. The corresponding row of data had been delet database.

PLOTTING CHART
PLOTTING CHART

Step 1 — Create a new **JFrame** and save it as **Charts_Address.java**.

Step 2 — In **Charts_Address.java**, add three **JPanels** and set their corresponding **Variable Name** as **jPanel1**, **jPanel2**, and **jPanel3**. Then, add getter method for each object as follows:

```
    //Getter method for jPanel1
    public JPanel getJPanel1(){
```

```
        return this.jPanel1;
    }

    //Getter method for jPanel2
    public JPanel getJPanel2(){
        return this.jPanel2;
    }

    //Getter method for jPanel3
    public JPanel getJPanel3(){
        return this.jPanel3;
    }
```

Step 3

In **Address_Utils** class, define three new methods. These are methods that draw different types of charts for the address data.

The **draw_pie_chart_address_by_district()** method creates a **DefaultPieDataset** using **create_pie_dataset()** method, which executes a SQL query to get the number of addresses per district and creates a dataset with that data. Then, it calls **draw_piechart_with_dataset()** to draw a pie chart with that dataset.

The **draw_pie_chart_address_by_country()** method is similar to the previous method, but it uses a different SQL query to get the number of addresses per country and create a dataset with that data. Then, it calls **draw_piechart_with_dataset()** to draw a pie chart with that dataset.

The **draw_bar_chart_address_by_city()** method creates a **DefaultCategoryDataset** using **create_bar_dataset()** method, which executes a SQL query to get the number of addresses per city and creates a dataset with that data. Then, it calls **draw_barchart_with_dataset()** to draw a bar chart with that dataset.

```
    private static void
draw_pie_chart_address_by_district(Charts_Address frm, JPanel jp)
{
        jp.setPreferredSize(new Dimension(jp.getWidth(),
jp.getHeight()));
        DefaultPieDataset dataset =
create_pie_dataset(Query_Address.get_sql_address_district_dist(),
"Number", "district");

        //Draws piechart address distribution by district
        draw_piechart_with_dataset(frm, jp, dataset, "TOP 10
ADDRESS DISTRIBUTION BY DISTRICT");
    }
```

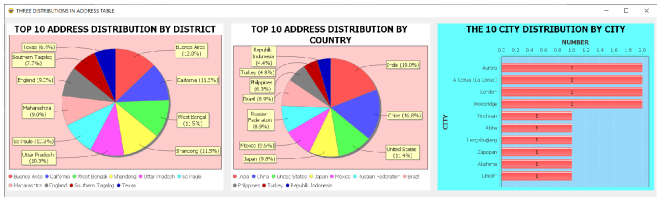
```
    private static void
draw_pie_chart_address_by_country(Charts_Address frm, JPanel jp){
        jp.setPreferredSize(new Dimension(jp.getWidth(),
jp.getHeight()));
        DefaultPieDataset dataset =
create_pie_dataset(Query_Address.get_sql_address_country_dist(),
"Number", "country");

        //Draws piechart address distribution by country
        draw_piechart_with_dataset(frm, jp, dataset, "TOP 10
ADDRESS DISTRIBUTION BY COUNTRY");
    }

    private static void
draw_bar_chart_address_by_city(Charts_Address frm, JPanel jp){
        jp.setPreferredSize(new Dimension(jp.getWidth(),
jp.getHeight()));

        DefaultCategoryDataset dataset =
create_bar_dataset(Query_Address.get_sql_address_city_dist(),
"Number", "city");

        //Draws barchart city distribution by city
        draw_barchart_with_dataset(frm, jp, dataset, "THE 10 CITY
DISTRIBUTION BY CITY", "CITY", "NUMBER");
    }
```

Step 4

In **Address_Utils** class, define a new method named **jbchart_handler()**.

```
    public static void
jbchart_handler(Charts_Address frm){
        //Draws piechart address
distribution by district

draw_pie_chart_address_by_district(frm,
frm.getJPanel1());

        //Draws piechart address
distribution by country

draw_pie_chart_address_by_country(frm,
frm.getJPanel2());

        //Draws barchart city
distribution by city

draw_bar_chart_address_by_city(frm,
frm.getJPanel3());
    }
```

Step 5	In **AddressForm**, double click on **jbChart** button to define its event listener:
	
	`private void pbChartActionPerformed(java.awt.event.ActionEvent evt) {`
	`  Charts_Address frm1 = new Charts_Address();`
	`  frm1.setLocationRelativeTo(null);`
	`  frm1.setTitle("THREE DISTRIBUTIONS IN ADDRESS TABLE");`
	`  frm1.setVisible(true);`
	`  Address_Utils.jbchart_handler(frm1);`
	`  }`

Step 6	Run the project. Click on CHART button on the form. You will see the three charts displayed on the panels as shown in Figure 10.12.  **Figure 10.12** The top 10 address distribution by district, the top 10 address distribution by country, and the top 10 address distribution by city

This is the full version of **Address_Utils.java**:

```
package sakila;
import java.awt.Dimension;
import java.util.logging.Level;
import java.util.logging.Logger;
import java.sql.*;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashMap;
import java.util.Objects;
import javax.swing.JComboBox;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.event.TableModelEvent;
import javax.swing.table.DefaultTableModel;
import javax.swing.table.TableModel;
import org.jfree.data.category.DefaultCategoryDataset;
import org.jfree.data.general.DefaultPieDataset;
```

```
import static sakila.Utility.create_bar_dataset;
import static sakila.Utility.draw_barchart_with_dataset;

public class Address_Utils extends Utility{
    public static final int FIRST_INDEX = 0;
    public static final int INVALID_INDEX = -1;

    private static int currentIndex = FIRST_INDEX;
    private static final String SQL_ID = Query_Address.get_sql_id();

    //Creates address table
    public static void create_address_table() {
        try (Connection conn = getConnection()) {
            Statement stmt = conn.createStatement();
            stmt.addBatch(Query_Address.get_sql_address());
            stmt.executeBatch();

            String message = String.format("Successfully creates address
table");
            JOptionPane.showMessageDialog(null, message,
            "INFORMATION",JOptionPane.INFORMATION_MESSAGE);

        } catch (SQLException ex) {
            JOptionPane.showMessageDialog(null, ex.getMessage(),
                "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    //Populates address table with some rows of data
    public static void populate_address_table(){
        try(Connection conn = getConnection()){
            String sql = """
                INSERT INTO address(address_id, address, address2, district,
                    city_id, postal_code, phone, last_update)
                VALUES(?, ?, ?, ?, ?, ?, ?, ?)""";

            //Creates a new Address class with default constructor
            PreparedStatement ps1 = conn.prepareStatement(sql);
            Address obj1 = new Address();
            ps1.setInt(1,obj1.getAddressID());
            ps1.setString(2,obj1.getAddress());
            ps1.setString(3,obj1.getAddress2());
            ps1.setString(4,obj1.getDistrict());
            ps1.setInt(5,obj1.getCityID());
            ps1.setString(6,obj1.getPostalCode());
            ps1.setString(7,obj1.getPhone());
            ps1.setTimestamp(8,obj1.getLastUpdate());

            // Creates a new Address class with eight-params constructor
            PreparedStatement ps2 = conn.prepareStatement(sql);
            Address obj2 = new Address(2, "Balige No. 1", "Balige No. 2",
"BLG", 2,
```

```
                    "89222", "0892682322", new
Timestamp(System.currentTimeMillis()));
            ps2.setInt(1,obj2.getAddressID());
            ps2.setString(2,obj2.getAddress());
            ps2.setString(3,obj2.getAddress2());
            ps2.setString(4,obj2.getDistrict());
            ps2.setInt(5,obj2.getCityID());
            ps2.setString(6,obj2.getPostalCode());
            ps2.setString(7,obj2.getPhone());
            ps2.setTimestamp(8,obj2.getLastUpdate());

            ps1.executeUpdate();
            ps2.executeUpdate();

        }catch(SQLException ex){
            JOptionPane.showMessageDialog(null, ex.getMessage(),
                "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    //Reads the content of joined address, city and country tables
    public static void read_address_table(){
        try(Connection conn = getConnection()){
            Statement stmt = conn.createStatement();
            ResultSet rs =
stmt.executeQuery(Query_Address.get_sql_address_joint());

            while(rs.next()){
                int add_id = rs.getInt("address_id");
                String add = rs.getString("address");
                String add2 = rs.getString("address2");
                String dist = rs.getString("district");
                int city_id = rs.getInt("city_id");
                String post = rs.getString("postal_code");
                String ph = rs.getString("phone");
                String city = rs.getString("city");
                String country = rs.getString("country");
                Timestamp lu = rs.getTimestamp("last_update");

                //Creates a Address object using ten-params constructor
                Address obj = new Address(add_id, add, add2, dist, city_id,
post, ph, city, country, lu);
                System.out.println(obj);
            }
            rs.close();
            stmt.close();

        }catch(SQLException ex){
            JOptionPane.showMessageDialog(null, ex.getMessage(),
                "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }
```

```
    private static ArrayList<Address> get_address_list(AddressForm frm, Strin
sql, String item){
        ArrayList<Address> list = new ArrayList<>();

        try(Connection conn = getConnection();
            PreparedStatement ps = conn.prepareStatement(sql)){
            if (item.equalsIgnoreCase("none")==false) {
                 ps.setString(1,item);
            }
            ResultSet rs = ps.executeQuery();

            Address obj;
            while(rs.next()){
                //Using ten-params constructor
                obj = new Address(rs.getInt("address_id"),
                        rs.getString("address"),
                        rs.getString("address2"),
                        rs.getString("district"),
                        rs.getInt("city_id"),
                        rs.getString("postal_code"),
                        rs.getString("phone"),
                        rs.getString("city"),
                        rs.getString("country"),
                        rs.getTimestamp("last_update"));

                list.add(obj);
            }
        }catch (SQLException ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
                "ERROR",JOptionPane.ERROR_MESSAGE);
        }
        return list;
    }

    private static void show_table_address(AddressForm frm, ArrayList<Address
list) throws SQLException{
        DefaultTableModel model = new DefaultTableModel(0,0);

        String header[] = {"Address ID", "Address", "Address 2", "District",
            "City ID", "Postal Code", "Phone", "City", "Country", "Last
Update"};

        model.setColumnIdentifiers(set_column_header(frm.getJTAddress(),
header));
        frm.getJTAddress().setModel(model);

        Object[] row = new Object[10];

        for(int i=0; i<list.size(); i++){
            row[0] = list.get(i).getAddressID();
            row[1] = list.get(i).getAddress();
            row[2] = list.get(i).getAddress2();
            row[3] = list.get(i).getDistrict();
```

```
            row[4] = list.get(i).getCityID();
            row[5] = list.get(i).getPostalCode();
            row[6] = list.get(i).getPhone();
            row[7] = list.get(i).getCity();
            row[8] = list.get(i).getCountry();
            row[9] = list.get(i).getLastUpdate();

            model.addRow(row);
        }
    }

    public static void refresh_controls(AddressForm frm){
        frm.setLocationRelativeTo(null);
        frm.setTitle("ADDRESS FORM");

        //Shows the content of address table and populates combobox
        try{
            //Makes alternating color for table rows
            table_renderer(frm.getJTAddress());

            //Populates table
            ArrayList<Address> list = get_address_list(frm,
Query_Address.get_sql_address_joint() + " ORDER BY address_id", "none");
            show_table_address(frm, list);

            //Populates jcbAddressID
            String sql_add_id = "SELECT address_id FROM address ORDER BY
address_id";
            populate_combobox(sql_add_id, frm.getJCBAddressID(), frm);

            //Populates jcbDistrictID
            String sql_dist = "SELECT DISTINCT district FROM address ORDER BY
district";
            populate_combobox(sql_dist, frm.getJCBDistrict(), frm);

            //Populates jcbCityID
            String sql_city_id = "SELECT city_id FROM city ORDER BY city_id";
            populate_combobox(sql_city_id, frm.getJCBCityID(), frm);

            //Populates jcbCity
            String sql_city = "SELECT DISTINCT city FROM city ORDER BY city";
            populate_combobox(sql_city, frm.getJCBCity(), frm);

            //Populates getJCBCountry()
            String sql_ct = "SELECT DISTINCT country FROM country ORDER BY
country";
            populate_combobox(sql_ct, frm.getJCBCountry(), frm);

        }catch (SQLException ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
            "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }
```

```
   //Displays joined country and city data result row by row
    private static <T> void display_country_city_data(AddressForm frm, String
sql, T item){
        try(Connection conn = getConnection()){
            PreparedStatement ps = conn.prepareStatement(sql);
            ps.setObject(1,item);
            ResultSet rs = ps.executeQuery();

            if (!rs.next()) {
                // no row found, clear the form fields
                frm.getJCBCityID().setSelectedIndex(-1);
                frm.getJCBCity().setSelectedIndex(-1);
                frm.getJCBCountry().setSelectedIndex(-1);
                return;
            }

            do{
                // Determines item selected from jcbCityID
                find_combo_value_selected(frm.getJCBCityID(),
rs.getInt("city_id"));

                // Determines item selected from jcbCity
                find_combo_value_selected(frm.getJCBCity(),
rs.getString("city"));

                // Determines item selected from jcbCountry
                find_combo_value_selected(frm.getJCBCountry(),
rs.getString("country"));
            }while(rs.next());

            rs.close();
            ps.close();
        }catch(SQLException ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    public static void jcbCity_handler(AddressForm frm, JComboBox<String> jcb
{
        Object item = jcb.getSelectedItem();
        String sql = "";
        if (jcb.equals(frm.getJCBCityID())) {
            sql = Query_City.get_sql_city_joint() + " WHERE city_id = ?";
        } else if (jcb.equals(frm.getJCBCity())) {
            sql = Query_City.get_sql_city_joint() + " WHERE city = ?";
        }

        display_country_city_data(frm, sql, item);
    }

    private static void clear_controls(AddressForm frm){
```

```
        frm.getJTFAddressID().setText("");
        frm.getJTFAddress().setText("");
        frm.getJTFLastUpdate().setText("");
        frm.getJTFAddress2().setText("");
        frm.getJTFDistrict().setText("");
        frm.getJTFPhone().setText("");
        frm.getJTFPostalCode().setText("");
        frm.getJTFAddress2().setText("");

        frm.getJCBCountry().setSelectedIndex(-1);
        frm.getJCBCityID().setSelectedIndex(-1);
        frm.getJCBCity().setSelectedIndex(-1);
        frm.getJCBCountry().setSelectedIndex(-1);
    }

   //Displays address data result row by row
    private static <T> void display_address_data(AddressForm frm, String sql,
item){
        try(Connection conn = getConnection()){
            PreparedStatement ps = conn.prepareStatement(sql);
            ps.setObject(1,item);
            ResultSet rs = ps.executeQuery();

            if (!rs.next()) {
                // no row found, clear the form fields
                clear_controls(frm);
                return;
            }

            do{

frm.getJTFAddressID().setText(String.valueOf(rs.getInt("address_id")));
                frm.getJTFAddress().setText(rs.getString("address"));
                frm.getJTFAddress2().setText(rs.getString("address2"));
                frm.getJTFDistrict().setText(rs.getString("district"));
                frm.getJTFPhone().setText(rs.getString("phone"));
                frm.getJTFPostalCode().setText(rs.getString("postal_code"));

frm.getJTFLastUpdate().setText(String.valueOf(rs.getTimestamp("last_update"))

                // Determines item selected from jcbAddressID
                find_combo_value_selected(frm.getJCBAddressID(),
rs.getInt("address_id"));

                // Determines item selected from jcbDistrict
                find_combo_value_selected(frm.getJCBDistrict(),
rs.getString("district"));

                // Determines item selected from jcbCityID
                find_combo_value_selected(frm.getJCBCityID(),
rs.getInt("city_id"));

                // Determines item selected from jcbCity
```

```
                find_combo_value_selected(frm.getJCBCity(),
rs.getString("city"));

            }while(rs.next());

            rs.close();
            ps.close();
        }catch(SQLException ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    public static void jcbAddress_handler(AddressForm frm) {
        Object item = frm.getJCBAddressID().getSelectedItem();
        display_address_data(frm, SQL_ID, item);
    }

    public static void show_first_row(AddressForm frm){
        String item =
String.valueOf(frm.getJCBAddressID().getItemAt(FIRST_INDEX));
        display_address_data(frm, SQL_ID, item);
        currentIndex = FIRST_INDEX;
    }

    public static void show_last_row(AddressForm frm){
        int endIndex = frm.getJCBAddressID().getItemCount() - 1;
        String item =
String.valueOf(frm.getJCBAddressID().getItemAt(endIndex));
        display_address_data(frm, SQL_ID, item);
        currentIndex = endIndex;
    }

    public static void show_prev_row(AddressForm frm){
        currentIndex--;
        if(currentIndex < FIRST_INDEX){
            currentIndex = FIRST_INDEX;
            return;
        }
        String item =
String.valueOf(frm.getJCBAddressID().getItemAt(currentIndex));
        display_address_data(frm, SQL_ID, item);
    }

    public static void show_next_row(AddressForm frm){
        int endIndex = frm.getJCBAddressID().getItemCount() - 1;
        currentIndex++;
        if(currentIndex > endIndex){
            currentIndex = endIndex;
            return;
        }
        String item =
String.valueOf(frm.getJCBAddressID().getItemAt(currentIndex));
```

```
        display_address_data(frm, SQL_ID, item);
    }

    public static void mouse_pressed_handler(AddressForm frm) {
        Objects.requireNonNull(frm, "frm must not be null");

        int selectedIndex = frm.getJTAddress().getSelectedRow();
        if (selectedIndex == -1) {
            JOptionPane.showMessageDialog(frm, "Please select a row to view 
data.",
                "No row selected", JOptionPane.INFORMATION_MESSAGE);
            return;
        }

        try (Connection conn = getConnection()) {
            String id =
String.valueOf(frm.getJTAddress().getModel().getValueAt(selectedIndex, 0));

            // Displays address data
            display_address_data(frm, SQL_ID, id);

        } catch (SQLException ex) {
            Logger.getLogger(AddressForm.class.getName()).log(Level.SEVERE,
"Error displaying address data", ex);
            String message = "Error displaying address data: " +
ex.getMessage();
            String stackTrace = Arrays.toString(ex.getStackTrace());
            JOptionPane.showMessageDialog(frm, message + "\n\n" + stackTrace,
"ERROR", JOptionPane.ERROR_MESSAGE);
        }
    }

    //Updates row of data in address tabel by address_id
    public static void update_row_by_address_id(AddressForm frm, int add_id,
String add, String add2, String dist,
            int city_id, String post, String ph) throws SQLException{
        Connection conn = getConnection();
        ResultSet rs = null;
        String query_id = "SELECT address_id FROM address WHERE address_id =
?";
        String update_query = """
            UPDATE address SET address = ?, address2 = ?, district = ?,
            city_id = ?, postal_code = ?, phone = ? WHERE address_id = ?""";
        try(PreparedStatement idPs = conn.prepareStatement(query_id,

ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_READ_ONLY);
        PreparedStatement updatePS = conn.prepareStatement(update_query,
                ResultSet.TYPE_SCROLL_SENSITIVE, ResultSet.CONCUR_UPDATABLE))
        {
            idPs.setInt(1,add_id);
            if(!idPs.execute()){
                String message = "Can't find address_id " + add_id;
```

```
                JOptionPane.showMessageDialog(frm, message,
                    "ERROR",JOptionPane.ERROR_MESSAGE);
            } else{
                rs = idPs.getResultSet();
                rs.next();

                //Creates a Address object using eight-params constructor
                Address obj = new Address(add_id, add, add2, dist, city_id,
post, ph, new Timestamp(System.currentTimeMillis()));
                updatePS.setString(1, obj.getAddress());
                updatePS.setString(2, obj.getAddress2());
                updatePS.setString(3, obj.getDistrict());
                updatePS.setInt(4, obj.getCityID());
                updatePS.setString(5, obj.getPostalCode());
                updatePS.setString(6, obj.getPhone());
                updatePS.setInt(7, obj.getAddressID());

                updatePS.executeUpdate();
                rs.close();
                updatePS.close();
                idPs.close();
                conn.close();
            }
        }catch(SQLException ex){
            Logger.getLogger(AddressForm.class.getName()).log(Level.SEVERE,
"Error updating address data", ex);
            String message = "Error updating address data: " + ex.getMessage(
            String stackTrace = Arrays.toString(ex.getStackTrace());
            JOptionPane.showMessageDialog(null, message + "\n\n" + stackTrace
"ERROR", JOptionPane.ERROR_MESSAGE);
        }catch(java.lang.NumberFormatException ex){
            Logger.getLogger(AddressForm.class.getName()).log(Level.SEVERE,
"Invalid Input", ex);
            String message = "Invalid Input: " + ex.getMessage();
            String stackTrace = Arrays.toString(ex.getStackTrace());
            JOptionPane.showMessageDialog(null, message + "\n\n" + stackTrace
"ERROR", JOptionPane.ERROR_MESSAGE);
        }
    }

    private static HashMap<String, String> read_inputs(AddressForm frm) {
        HashMap<String, String> input_data = new HashMap<>();
        String add_id =
String.valueOf(frm.getJCBAddressID().getSelectedItem());
        String city_id = String.valueOf(frm.getJCBCityID().getSelectedItem()
        String add = frm.getJTFAddress().getText();
        String add2 = frm.getJTFAddress2().getText();
        String dist = frm.getJTFDistrict().getText();
        String ph = frm.getJTFPhone().getText();
        String post = frm.getJTFPostalCode().getText();

        // Validate user input
        int add_id_int = 0;
```

```
        try {
            add_id_int = Integer.parseInt(add_id);
            if (add_id_int <= 0) {
                throw new IllegalArgumentException("Address ID cannot be
negative or zero");
            }
        } catch (NumberFormatException ex) {
            JOptionPane.showMessageDialog(frm, "Invalid Address ID: " + add_i
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        } catch (IllegalArgumentException ex) {
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        }

        int city_id_int = 0;
        try {
            city_id_int = Integer.parseInt(city_id);
            if (city_id_int <= 0) {
                throw new IllegalArgumentException("City ID cannot be negati
or zero");
            }
        } catch (NumberFormatException ex) {
            JOptionPane.showMessageDialog(frm, "Invalid City ID: " + city_id,
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        } catch (IllegalArgumentException ex) {
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        }

        if (add == null || add.isEmpty()) {
            JOptionPane.showMessageDialog(frm, "Address cannot be empty",
            "Error", JOptionPane.ERROR_MESSAGE);
            throw new IllegalArgumentException("Address cannot be empty");
        }

        if (dist == null || dist.isEmpty()) {
            JOptionPane.showMessageDialog(frm, "District cannot be empty",
            "Error", JOptionPane.ERROR_MESSAGE);
            throw new IllegalArgumentException("District cannot be empty");
        }

        if (ph == null || ph.isEmpty()) {
            JOptionPane.showMessageDialog(frm, "Phone cannot be empty",
            "Error", JOptionPane.ERROR_MESSAGE);
            throw new IllegalArgumentException("Phone cannot be empty");
        }

        input_data.put("add_id", add_id);
        input_data.put("city_id", city_id);
```

```
        input_data.put("add", add);
        input_data.put("add2", add2);
        input_data.put("dist", dist);
        input_data.put("ph", ph);
        input_data.put("post", post);

        return input_data;
    }

    private static void edit_actual(AddressForm frm){
        try{
            HashMap<String, String> input_data = read_inputs(frm);
            int add_id = Integer.parseInt(input_data.get("add_id"));
            int city_id = Integer.parseInt(input_data.get("city_id"));
            String add = input_data.get("add");
            String add2 = input_data.get("add2");
            String dist = input_data.get("dist");
            String post = input_data.get("post");
            String ph = input_data.get("ph");

            update_row_by_address_id(frm, add_id, add, add2, dist, city_id,
post, ph);

            //Refreshes all objects on form
            refresh_controls(frm);

        }catch(SQLException ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    private static void enable_controls(boolean state, AddressForm frm){
        frm.getJBFirst().setEnabled(state);
        frm.getJBPrev().setEnabled(state);
        frm.getJBNext().setEnabled(state);
        frm.getJBLast().setEnabled(state);
        frm.getJBInsert().setEnabled(state);
        frm.getJBDelete().setEnabled(state);
        frm.getJTFAddressID().setEnabled(state);
        frm.getJCBCountry().setEnabled(state);
    }

    public static void edit_handler(AddressForm frm){
        if(frm.getJBEdit().getText().equals("EDIT")){
            frm.getJBEdit().setText("CONFIRM");

            // Disables controls
            enable_controls(false, frm);
        }

        else {
            frm.getJBEdit().setText("EDIT");
```

```
            // Actual editing
            edit_actual(frm);

            //Enables controls
            enable_controls(true, frm);
        }
    }

    public static void edit_database_from_jtable(TableModelEvent e, AddressFo
frm){
        if (e.getType() == TableModelEvent.UPDATE) {
            int row = e.getFirstRow();
            TableModel model = (TableModel)e.getSource();
            int add_id = Integer.parseInt(String.valueOf(model.getValueAt(rov
0)));
            String add = String.valueOf(model.getValueAt(row, 1));
            String add2 = String.valueOf(model.getValueAt(row, 2));
            String dist = String.valueOf(model.getValueAt(row, 3));
            int city_id = Integer.parseInt(String.valueOf(model.getValueAt(ro
4)));
            String post = String.valueOf(model.getValueAt(row, 5));
            String ph = String.valueOf(model.getValueAt(row, 6));

            try{
                update_row_by_address_id(frm, add_id, add, add2, dist, city_i
post, ph);

                //Refreshes all objects on form
                refresh_controls(frm);

            } catch (SQLIntegrityConstraintViolationException ex) {
                Logger.getLogger(AddressForm.class.getName()).log(Level.SEVER
"Duplicate entry", ex);
                JOptionPane.showMessageDialog(frm, "Error: Duplicate entry\n"
ex.getMessage());
            } catch (SQLSyntaxErrorException ex) {
                Logger.getLogger(AddressForm.class.getName()).log(Level.SEVER
"Invalid SQL syntax", ex);
                JOptionPane.showMessageDialog(frm, "Error: Invalid SQL
syntax\n" + ex.getMessage());
            } catch (SQLException ex) {
                Logger.getLogger(AddressForm.class.getName()).log(Level.SEVER
"Database error", ex);
                JOptionPane.showMessageDialog(frm, "Error: Database error\n"
ex.getMessage());
            }
        }
    }

    //Inserts new row into address table
    private static void insert_row(AddressForm frm) throws SQLException{
        HashMap<String, String> input_data = read_inputs(frm);
```

```
        int city_id = Integer.parseInt(input_data.get("city_id"));
        String add = input_data.get("add");
        String add2 = input_data.get("add2");
        String dist = input_data.get("dist");
        String post = input_data.get("post");
        String ph = input_data.get("ph");

        // SQL insert statement
        String sql = """
            INSERT INTO address(address, address2, district,
                city_id, postal_code, phone)
                VALUES(?, ?, ?, ?, ?, ?)""";

        try(Connection conn = getConnection();
         PreparedStatement pstmt = conn.prepareStatement(sql)){

            //Creates a Address object seven-params constructor
            Address obj = new Address(add, add2, dist, city_id, post, ph, new
Timestamp(System.currentTimeMillis()));
            pstmt.setString(1,obj.getAddress());
            pstmt.setString(2,obj.getAddress2());
            pstmt.setString(3,obj.getDistrict());
            pstmt.setInt(4,obj.getCityID());
            pstmt.setString(5,obj.getPostalCode());
            pstmt.setString(6,obj.getPhone());

            //Executes the sql insert statement
            pstmt.executeUpdate();
        } catch (SQLException ex) {
            Logger.getLogger(AddressForm.class.getName()).log(Level.SEVERE,
"Database error", ex);
            JOptionPane.showMessageDialog(frm, "Error: Database error\n" +
ex.getMessage());
        }
    }

    private static void insert_actual(AddressForm frm){
        try{
            insert_row(frm);

            //Refreshes table and comboboxes
            refresh_controls(frm);

        }catch(SQLException ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    public static void insert_handler(AddressForm frm){
        if(frm.getJBInsert().getText().equals("INSERT")){
            frm.getJBInsert().setText("CONFIRM");
```

```
            //Disables jbEdit
            frm.getJBEdit().setEnabled(false);

            // Disables controls
            enable_controls(false, frm);
            frm.getJCBAddressID().setEnabled(false);

            // Clears controls
            clear_controls(frm);

            // Enables
            frm.getJBInsert().setEnabled(true);
        }

        else {
            frm.getJBInsert().setText("INSERT");

            // Actual insertion
            insert_actual(frm);

            //Enables jbEdit
            frm.getJBEdit().setEnabled(true);

            //Enables controls
            enable_controls(true, frm);
            frm.getJCBAddressID().setEnabled(true);
        }
    }

    public static void delete_handler(AddressForm frm){
        int dialogButton = JOptionPane.YES_NO_OPTION;
        int add_id =
Integer.parseInt(String.valueOf(frm.getJCBAddressID().getSelectedItem()));

        String message = String.format("Are you sure you want to delete the r
Address ID: %d)", add_id);
        int answer = JOptionPane.showConfirmDialog(frm, message, "DELETING RO
OF DATA", dialogButton);

        if(answer == JOptionPane.YES_OPTION){
            String query = """
                DELETE FROM address WHERE address_id = ?""";
            try(Connection conn = getConnection();
                PreparedStatement ps = conn.prepareStatement(query)){
                // Use PreparedStatement to avoid SQL injection attacks
                ps.setInt(1, add_id);
                ps.executeUpdate();

                // Refresh table and comboboxes
                refresh_controls(frm);

            } catch (SQLException ex){
                JOptionPane.showMessageDialog(frm, ex.getMessage(),
```

```
                    "ERROR",JOptionPane.ERROR_MESSAGE);
            }
        }
    }

    private static void draw_pie_chart_address_by_district(Charts_Address frm,
JPanel jp){
        jp.setPreferredSize(new Dimension(jp.getWidth(), jp.getHeight()));
        DefaultPieDataset dataset =
create_pie_dataset(Query_Address.get_sql_address_district_dist(), "Number",
"district");

        //Draws piechart address distribution by district
        draw_piechart_with_dataset(frm, jp, dataset, "TOP 10 ADDRESS
DISTRIBUTION BY DISTRICT");
    }

    private static void draw_pie_chart_address_by_country(Charts_Address frm,
JPanel jp){
        jp.setPreferredSize(new Dimension(jp.getWidth(), jp.getHeight()));
        DefaultPieDataset dataset =
create_pie_dataset(Query_Address.get_sql_address_country_dist(), "Number",
"country");

        //Draws piechart address distribution by country
        draw_piechart_with_dataset(frm, jp, dataset, "TOP 10 ADDRESS
DISTRIBUTION BY COUNTRY");
    }

    private static void draw_bar_chart_address_by_city(Charts_Address frm,
JPanel jp){
        jp.setPreferredSize(new Dimension(jp.getWidth(), jp.getHeight()));

        DefaultCategoryDataset dataset =
create_bar_dataset(Query_Address.get_sql_address_city_dist(), "Number",
"city");

        //Draws barchart city distribution by city
        draw_barchart_with_dataset(frm, jp, dataset, "THE 10 CITY DISTRIBUTI(
BY CITY", "CITY", "NUMBER");
    }

    public static void jbchart_handler(Charts_Address frm){
        //Draws piechart address distribution by district
        draw_pie_chart_address_by_district(frm, frm.getJPanel1());

        //Draws piechart address distribution by country
        draw_pie_chart_address_by_country(frm, frm.getJPanel2());

        //Draws barchart city distribution by city
        draw_bar_chart_address_by_city(frm, frm.getJPanel3());
    }
}
```

This is the full version of **AddressForm.java**:

```
package sakila;

import java.awt.Toolkit;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JComboBox;
```

```
import javax.swing.JMenuItem;
import javax.swing.JOptionPane;
import javax.swing.JPopupMenu;
import javax.swing.JTable;
import javax.swing.JTextField;
import javax.swing.event.TableModelEvent;
import javax.swing.event.TableModelListener;

public class AddressForm extends javax.swing.JFrame {
    public AddressForm() {
        initComponents();
        Utility.setLookAndFeel(this);
        Address_Utils.refresh_controls(this);

 this.setIconImage(Toolkit.getDefaultToolkit().getImage(getClass().getResourc
;
        this.setDefaultCloseOperation(this.HIDE_ON_CLOSE);
    }

    //Getter method for jtfAddressID
    public JTextField getJTFAddressID(){
        return this.jtfAddressID;
    }

    //Getter method for jcbAddressID
    public JComboBox getJCBAddressID(){
        return this.jcbAddressID;
    }

    //Getter method for jtfAddress
    public JTextField getJTFAddress(){
        return this.jtfAddress;
    }

    //Getter method for jtfAddress2
    public JTextField getJTFAddress2(){
        return this.jtfAddress2;
    }

    //Getter method for jtfDistrict
    public JTextField getJTFDistrict(){
        return this.jtfDistrict;
    }

    //Getter method for jcbDistrict
    public JComboBox getJCBDistrict(){
        return this.jcbDistrict;
    }

    //Getter method for jtfPostalCode
    public JTextField getJTFPostalCode(){
        return this.jtfPostalCode;
    }
```

```
    //Getter method for jtfLastUpdate
    public JTextField getJTFLastUpdate(){
        return this.jtfLastUpdate;
    }

    //Getter method for jcbCityID
    public JComboBox getJCBCityID(){
        return this.jcbCityID;
    }

    //Getter method for jcbCity
    public JComboBox getJCBCity(){
        return this.jcbCity;
    }

    //Getter method for jcbCountry
    public JComboBox getJCBCountry(){
        return this.jcbCountry;
    }

    //Getter method for jtfPhone
    public JTextField getJTFPhone(){
        return this.jtfPhone;
    }

    //Getter method for jbEdit
    public JButton getJBEdit(){
        return this.jbEdit;
    }

    //Getter method for jbInsert
    public JButton getJBInsert(){
        return this.jbInsert;
    }

    //Getter method for jbDelete
    public JButton getJBDelete(){
        return this.jbDelete;
    }

    //Getter method for pbChart
    public JButton getJBChart(){
        return this.pbChart;
    }

    //Getter method for jbFirst
    public JButton getJBFirst(){
        return this.jbFirst;
    }

    //Getter method for jbPrev
    public JButton getJBPrev(){
```

```
        return this.jbPrev;
    }

    //Getter method for jbNext
    public JButton getJBNext(){
        return this.jbNext;
    }

    //Getter method for jbLast
    public JButton getJBLast(){
        return this.jbLast;
    }

    //Getter method for jtAddress
    public JTable getJTAddress(){
        return this.jtAddress;
    }

    //Getter method for jbCountryForm
    public JButton getJBCountryForm(){
        return this.jbCountryForm;
    }

    //Getter method for jbCityForm
    public JButton getJBCityForm(){
        return this.jbCityForm;
    }


    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">
    private void initComponents() {
        //...
        pack();
    }// </editor-fold>

    private void jcbAddressIDActionPerformed(java.awt.event.ActionEvent evt)
        Address_Utils.jcbAddress_handler(this);
    }

    private void jbFirstActionPerformed(java.awt.event.ActionEvent evt) {
        Address_Utils.show_first_row(this);
    }

    private void jbPrevActionPerformed(java.awt.event.ActionEvent evt) {
        Address_Utils.show_prev_row(this);
    }

    private void jtAddressMousePressed(java.awt.event.MouseEvent evt) {
        Address_Utils.mouse_pressed_handler(this);
    }
```

```
    private void jbNextActionPerformed(java.awt.event.ActionEvent evt) {
        Address_Utils.show_next_row(this);
    }

    private void jbLastActionPerformed(java.awt.event.ActionEvent evt) {
        Address_Utils.show_last_row(this);
    }

    private void jbDeleteActionPerformed(java.awt.event.ActionEvent evt) {
        Address_Utils.delete_handler(this);
    }

    private void jbInsertActionPerformed(java.awt.event.ActionEvent evt) {
        Address_Utils.insert_handler(this);
    }

    private void jbEditActionPerformed(java.awt.event.ActionEvent evt) {
        Address_Utils.edit_handler(this);
    }

    private void jbCountryFormActionPerformed(java.awt.event.ActionEvent evt)
        CountryForm ct_form = new CountryForm();
        ct_form.setVisible(true);
    }

    private void pbChartActionPerformed(java.awt.event.ActionEvent evt) {
        Charts_Address frm1 = new Charts_Address();
        frm1.setLocationRelativeTo(null);
        frm1.setTitle("THREE DISTRIBUTIONS IN ADDRESS TABLE");
        frm1.setVisible(true);
        Address_Utils.jbchart_handler(frm1);
    }

    private void jcbCityActionPerformed(java.awt.event.ActionEvent evt) {
        Address_Utils.jcbCity_handler(this, this.jcbCity);
    }

    private void jbCityFormActionPerformed(java.awt.event.ActionEvent evt) {
        CityForm cty_form = new CityForm();
        cty_form.setVisible(true);
    }

    private void jcbCityIDActionPerformed(java.awt.event.ActionEvent evt) {
        Address_Utils.jcbCity_handler(this, this.jcbCityID);
    }

    private void jtAddressMouseClicked(java.awt.event.MouseEvent evt) {
        // instantiate AddressTableModelListener and add it as a listener to
        AddressTableModelListener tableModelListener = new
AddressTableModelListener(this.getJTAddress(), this);
        this.getJTAddress().getModel().addTableModelListener(tableModelLister
    }
```

```
    /**
     * @param args the command line arguments
     */
    public static void main(String args[]) {
        try {
            for (javax.swing.UIManager.LookAndFeelInfo info :
javax.swing.UIManager.getInstalledLookAndFeels()) {
                if ("Nimbus".equals(info.getName())) {
                    javax.swing.UIManager.setLookAndFeel(info.getClassName())
                    break;
                }
            }
        } catch (ClassNotFoundException ex) {

java.util.logging.Logger.getLogger(AddressForm.class.getName()).log(java.util
null, ex);
        } catch (InstantiationException ex) {

java.util.logging.Logger.getLogger(AddressForm.class.getName()).log(java.util
null, ex);
        } catch (IllegalAccessException ex) {

java.util.logging.Logger.getLogger(AddressForm.class.getName()).log(java.util
null, ex);
        } catch (javax.swing.UnsupportedLookAndFeelException ex) {

java.util.logging.Logger.getLogger(AddressForm.class.getName()).log(java.util
null, ex);
        }
        /* Create and display the form */
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new AddressForm().setVisible(true);
            }
        });
    }

    // Variables declaration - do not modify
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel10;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JLabel jLabel4;
    private javax.swing.JLabel jLabel5;
    private javax.swing.JLabel jLabel6;
    private javax.swing.JLabel jLabel7;
    private javax.swing.JLabel jLabel8;
    private javax.swing.JLabel jLabel9;
    private javax.swing.JScrollPane jScrollPane1;
    private javax.swing.JButton jbCityForm;
    private javax.swing.JButton jbCountryForm;
    private javax.swing.JButton jbDelete;
    private javax.swing.JButton jbEdit;
```

```
    private javax.swing.JButton jbFirst;
    private javax.swing.JButton jbInsert;
    private javax.swing.JButton jbLast;
    private javax.swing.JButton jbNext;
    private javax.swing.JButton jbPrev;
    private javax.swing.JComboBox<String> jcbAddressID;
    private javax.swing.JComboBox<String> jcbCity;
    private javax.swing.JComboBox<String> jcbCityID;
    private javax.swing.JComboBox<String> jcbCountry;
    private javax.swing.JComboBox<String> jcbDistrict;
    private javax.swing.JTable jtAddress;
    private javax.swing.JTextField jtfAddress;
    private javax.swing.JTextField jtfAddress2;
    private javax.swing.JTextField jtfAddressID;
    private javax.swing.JTextField jtfDistrict;
    private javax.swing.JTextField jtfLastUpdate;
    private javax.swing.JTextField jtfPhone;
    private javax.swing.JTextField jtfPostalCode;
    private javax.swing.JButton pbChart;
    // End of variables declaration
}
```

STORE FORM

STORE FORM

CREATING AND POPULATING STORE TABLE

CREATING AND POPULATING STORE TABLE

Step 1	Create a new class named **Query_Store**. It includes a set of pre-defined SQL queries that are related to a database schema named sakila. Here's a breakdown of each query: 1. **sql_min**: This query retrieves the minimum value of **store_id** column from the **store** table. 2. **sql_max**: This query retrieves the maximum value of **store_id** column from the **store** table. 3. **sql_id**: This query is a parameterized SQL statement that retrieves information about a store with a specific **store_id**. The value of **store_id** is specified as a parameter. 4. **sql_store_district_dist**: This query retrieves the count of stores in each district, and returns the top 10 districts with the highest number of stores. 5. **sql_store_country_dist**: This query retrieves the count of stores in each country, and returns the top 10 countries with the highest number of stores. 6. **sql_store_city_dist**: This query retrieves the count of stores in each city, and returns the top 10 cities with the highest number of stores. 7. **sql_store_joint**: This query retrieves detailed information about all the stores, including their store_id, manager_staff_id, address_id, last_update, address, address2, district, city_id, postal_code, phone, city, and country. 8. **sql_store**: This query creates a new table named store with columns store_id, manager_staff_id,

address_id, and last_update. It also sets up various constraints and keys for the table.

All these queries are static fields of the class **Query_Store**, and can be accessed using the public getter methods provided for each query.

```
1  package sakila;
2
3  public class Query_Store {
4      private static final String
5  sql_min = "SELECT MIN(store_id)
6  FROM store";
7      private static final String
8  sql_max = "SELECT MAX(store_id)
9  FROM store";
10     private static final String
11 sql_id = get_sql_store_joint() + "
12 WHERE st.store_id  = ?";
13
14     private static final String
15 sql_store_district_dist = """
16         SELECT ad.district,
17 COUNT(*) AS Number
18         FROM store st
19         JOIN address ad ON
20 ad.address_id = st.address_id
21         GROUP BY ad.district
22         ORDER BY Count(*) DESC
23         LIMIT 10""";
24
25     private static final String
26 sql_store_country_dist = """
27         SELECT co.country, COUNT(*)
28 AS Number
29         FROM store st
30         JOIN address ad ON
31 ad.address_id = st.address_id
32         JOIN city ci ON ci.city_id
33 = ad.city_id
34         JOIN country co ON
35 co.country_id = ci.country_id
36         GROUP BY co.country
37         ORDER BY Count(*) DESC
38         LIMIT 10""";
39
40     private static final String
41 sql_store_city_dist = """
42         SELECT ci.city, COUNT(*) AS
43 Number
44         FROM store st
45         JOIN address ad ON
46 ad.address_id = st.address_id
```

```
        JOIN city ci ON ci.city_id
= ad.city_id
        GROUP BY ci.city
        ORDER BY Count(*) DESC
        LIMIT 10""";

    private static final String
sql_store_joint = """
    SELECT st.store_id,
st.manager_staff_id,
        st.address_id,
st.last_update,
        ad.address, ad.address2,
        ad.district, ad.city_id,
ad.postal_code,
        ad.phone, ci.city,
        co.country
    FROM store st
    JOIN address ad ON
ad.address_id = st.address_id
    JOIN city ci ON ci.city_id =
ad.city_id
    JOIN country co ON
co.country_id = ci.country_id""";

    private static final String
sql_store = """
        CREATE TABLE store (
          store_id TINYINT UNSIGNED
NOT NULL AUTO_INCREMENT,
          manager_staff_id TINYINT
UNSIGNED NOT NULL,
          address_id SMALLINT
UNSIGNED NOT NULL,
          last_update TIMESTAMP NOT
NULL DEFAULT CURRENT_TIMESTAMP ON
UPDATE CURRENT_TIMESTAMP,
          PRIMARY KEY  (store_id),
          UNIQUE KEY
idx_unique_manager
(manager_staff_id),
          KEY idx_fk_address_id
(address_id),
          CONSTRAINT
fk_store_address FOREIGN KEY
(address_id) REFERENCES address
(address_id) ON DELETE RESTRICT ON
UPDATE CASCADE
        ) ENGINE=InnoDB DEFAULT
CHARSET=utf8mb4;""";

    //Getter methods
    public static String
get_sql_min() {
```

```
        return sql_min;
    }

    public static String
get_sql_max() {
        return sql_max;
    }

    public static String
get_sql_id() {
        return sql_id;
    }

    public static String
get_sql_store() {
        return sql_store;
    }

    public static String
get_sql_store_joint() {
        return sql_store_joint;
    }

    public static String
get_sql_store_city_dist() {
        return sql_store_city_dist;
    }

    public static String
get_sql_store_country_dist() {
        return
sql_store_country_dist;
    }

    public static String
get_sql_store_district_dist() {
        return
sql_store_district_dist;
    }
}
```

Step 2	Then, create a public class named **Store**. It represents a store entity in a database for a fictional movie rental company called Sakila. The class has 12 instance variables representing different attributes of a store, such as store_id, manager_id, address_id, address, address2, district, city_id, postal_code, phone, city, country, and last_update. The class has four constructors: a default constructor that sets some default values, a three-params constructor that

takes in manager_id, address_id, and last_update, a four-params constructor that takes in store_id, manager_id, address_id, and last_update, and a twelve-params constructor that takes in all the instance variables of the class.

The class also has getter and setter methods for all its instance variables. The setter methods include validation checks to ensure that the input values are valid.

The class overrides the **hashCode()** and **equals()** methods to ensure that instances of the class can be compared and sorted correctly. The class also overrides the **toString()** method to provide a string representation of an instance of the class.

```
1   package sakila;
2   import java.util.Objects;
3   import java.sql.Timestamp;
4
5   public class Store {
6       //12 instance variables
7       private int store_id;
8       private int manager_id;
9       private int address_id;
10      private String address;
11      private String address2;
12      private String district;
13      private int city_id;
14      private String postal_code;
15      private String phone;
16      private String city;
17      private String country;
18      private Timestamp last_update;
19
20      //Default constructor
21      Store(){
22          this(1, 1, 1, new
23  Timestamp(System.currentTimeMillis()));
24      }
25
26      //Three-params constructor
27      Store(int man_id, int add_id,
28  Timestamp lu){
29          setAddressID(add_id);
30          setManagerID(man_id);
31          setLastUpdate(lu);
32      }
33
34      //Four-params constructor
35
```

```
36      Store(int store_id, int man_id, int
37  add_id, Timestamp lu){
38          this(man_id, add_id, lu);
39          setStoreID(store_id);
40      }
41
42      //Twelve-params constructor
43      Store(int store_id, int man_id, int
44  add_id, String add,
45          String add2, String dist, int
46  city_id, String post,
47          String ph, String city, String
48  country, Timestamp lu){
49          this(store_id, man_id, add_id,
50  lu);
51          this.address = add;
52          this.address2 = add2;
53          this.district = dist;
54          this.city_id = city_id;
55          this.postal_code = post;
56          this.phone = ph;
57          this.city = city;
58          this.country = country;
59      }
60
61      //Getter methods
62      public int getStoreID() {return
63  store_id;}
64      public int getManagerID() {return
65  manager_id;}
66      public int getAddressID() {return
67  address_id;}
68      public String getAddress() {return
69  address;}
70      public String getAddress2() {return
71  address2;}
72      public String getDistrict() {return
73  district;}
74      public int getCityID() {return
75  city_id;}
76      public String getPostalCode()
77  {return postal_code;}
78      public String getPhone() {return
79  phone;}
80      public Timestamp getLastUpdate()
81  {return last_update;}
82      public String getCity() {return
83  city;}
84      public String getCountry() {return
85  country;}
86
87      //Setter methods
88      public void setAddressID(int id) {
89
```

```
90          if (id <= 0) {
91              throw new
92  IllegalArgumentException("Address ID must be greater
93  than zero.");
94          }
95          this.address_id = id;
96      }
97
98      public void setStoreID(int id) {
99          if (id <= 0) {
100             throw new
101 IllegalArgumentException("Store ID must be greater
102 than zero.");
103         }
104         this.store_id = id;
105     }
106
107     public void setManagerID(int id) {
108         if (id <= 0) {
109             throw new
110 IllegalArgumentException("Manager ID must be greater
111 than zero.");
112         }
113         this.manager_id = id;
114     }
115
116     public void setLastUpdate(Timestamp
117 date){
118         if (date == null) {
119         throw new
120 IllegalArgumentException("Date cannot
121 be null");
122         }
123         this.last_update = date;
124     }
125
126127
    // Override the hashCode() method
    @Override
    public int hashCode() {
        return Objects.hash(store_id,
manager_id, last_update);
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() !=
o.getClass()) return false;
        Store st = (Store) o;
        return store_id == st.store_id
&&
```

```
                manager_id ==
st.manager_id &&

Objects.equals(last_update,
st.last_update);
    }

    @Override
    public String toString(){
        return "\nStore ID    : " +
getStoreID() +
                "\nManager ID    : " +
getManagerID() +
                 "\nAddress ID    : " +
getAddressID() +
                "\nAddress       : " +
getAddress() +
                "\nAddress2      : " +
getAddress2() +
                "\nDistrict      : " +
getDistrict() +
                "\nCity ID       : " +
getCityID() +
                "\nPostal Code   : " +
getPostalCode() +
                "\nPhone         : " +
getPhone() +
                "\nCity          : " +
getCity() +
                "\nCountry       : " +
getCountry() +
                "\nLast Update   : " +
getLastUpdate();
    }
}
```

Step 3

Create a new public class named **Store_Utils**. It extends the **Utility** class. It contains three methods that interact with a database to create and populate **store** table and read its contents using **Store** class. Here's an explanation of each method:

1. **create_store_table()**: This method creates a new table in the connected database using a SQL query defined in **Query_Store.get_sql_store()**. It uses a **Statement** object to execute the query and shows a message dialog to indicate whether the table was successfully created or not. If an **SQLException**

occurs, the error message is shown in an error message dialog.

2. **populate_store_table()**: This method inserts two rows of data into the **store** table. It uses a prepared statement to execute a SQL query that inserts data into the table. The data is retrieved from two instances of the **Store** class, one with the default constructor and another with an eight-params constructor. If an **SQLException** occurs, the error message is shown in an error message dialog.
3. **read_store_table()**: This method reads the contents of the **store** table and related tables using a SQL query defined in **Query_Store.get_sql_store_joint()**. It uses a **Statement** object to execute the query and retrieves the result set using a **ResultSet** object. The data is retrieved row by row using the **next()** method of the result set, and a new instance of the **Store** class is created for each row of data retrieved. If an **SQLException** occurs, the error message is shown in an error message dialog.

```
1   package sakila;
2   import java.awt.Dimension;
3   import java.util.logging.Level;
4   import java.util.logging.Logger;
5   import java.sql.*;
6   import java.util.ArrayList;
7   import java.util.Arrays;
8   import java.util.HashMap;
9   import java.util.Objects;
10  import javax.swing.JComboBox;
11  import javax.swing.JOptionPane;
12  import javax.swing.JPanel;
13  import javax.swing.event.TableModelEvent;
14  import javax.swing.table.DefaultTableModel;
15  import javax.swing.table.TableModel;
16  import
17  org.jfree.data.category.DefaultCategoryDataset;
18  import org.jfree.data.general.DefaultPieDataset;
19  import static sakila.Utility.create_bar_dataset;
20  import static
21  sakila.Utility.draw_barchart_with_dataset;
22
23  public class Store_Utils extends Utility{
24      public static final int FIRST_INDEX = 0;
25      public static final int INVALID_INDEX = -1;
26
27      private static int currentIndex = FIRST_INDEX;
28      private static final String SQL_ID =
29  Query_Store.get_sql_id();
30
```

```
31
32      //Creates store table
33      public static void create_store_table() {
34          try (Connection conn = getConnection()) {
35              Statement stmt = conn.createStatement();
36
37  stmt.addBatch(Query_Store.get_sql_store());
38              stmt.executeBatch();
39
40              String message =
41  String.format("Successfully creates store table");
42              JOptionPane.showMessageDialog(null,
43  message,
44
45   "INFORMATION",JOptionPane.INFORMATION_MESSAGE);
46
47          } catch (SQLException ex) {
48              JOptionPane.showMessageDialog(null,
49  ex.getMessage(),
50                  "ERROR",JOptionPane.ERROR_MESSAGE);
51          }
52      }
53
54      //Populates store table with some rows of data
55      public static void populate_store_table(){
56          try(Connection conn = getConnection()){
57              String sql = """
58                  INSERT INTO store(store_id,
59  manager_staff_id, address_id, last_update)
60                  VALUES(?, ?, ?, ?)""";
61
62              //Creates a new Store class with default
63  constructor
64              PreparedStatement ps1 =
65  conn.prepareStatement(sql);
66              Store obj1 = new Store();
67              ps1.setInt(1,obj1.getStoreID());
68              ps1.setInt(2,obj1.getManagerID());
69              ps1.setInt(3,obj1.getAddressID());
70              ps1.setTimestamp(4,obj1.getLastUpdate());
71
72              // Creates a new Store class with four-
73  params constructor
74              PreparedStatement ps2 =
75  conn.prepareStatement(sql);
76              Store obj2 = new Store(2, 2, 2, new
77  Timestamp(System.currentTimeMillis()));
78              ps2.setInt(1,obj2.getStoreID());
79              ps2.setInt(2,obj2.getManagerID());
80              ps2.setInt(3,obj2.getAddressID());
81              ps2.setTimestamp(4,obj2.getLastUpdate());
82
83              ps1.executeUpdate();
84
```

```
            ps2.executeUpdate();

        }catch(SQLException ex){
            JOptionPane.showMessageDialog(null,
ex.getMessage(),
                "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    //Reads the content of joined store, address,
city and country tables
    public static void read_store_table(){
        try(Connection conn = getConnection()){
            Statement stmt = conn.createStatement();
            ResultSet rs =
stmt.executeQuery(Query_Store.get_sql_store_joint());

            while(rs.next()){
                int store_id = rs.getInt("store_id");
                int man_id =
rs.getInt("manager_staff_id");
                int add_id = rs.getInt("address_id");
                String add = rs.getString("address");
                String add2 =
rs.getString("address2");
                String dist =
rs.getString("district");
                int city_id = rs.getInt("city_id");
                String post =
rs.getString("postal_code");
                String ph = rs.getString("phone");
                String city = rs.getString("city");
                String country =
rs.getString("country");
                Timestamp lu =
rs.getTimestamp("last_update");

                //Creates a Store object using
twelve-params constructor
                Store obj = new Store(store_id,
man_id, add_id, add, add2, dist, city_id, post, ph,
city, country, lu);
                System.out.println(obj);
            }
            rs.close();
            stmt.close();

        }catch(SQLException ex){
            JOptionPane.showMessageDialog(null,
ex.getMessage(),
                "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }
```

```
    }
```

Step 4

In the driver class, **Sakila.java**, invoke **create_store_table()**, **populate_store_table()**, and **read_store_table()** as shown in line 60 – 62:

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
```

```
32 package sakila;
33
34 public class Sakila {
35     public static void main(String[] args) {
36 //         Utility.testConnection();
37 //         Actor_Utils.create_actor_table();
38 //         Actor_Utils.populate_actor_table();
39 //         Actor_Utils.read_actor_table();
40 //         ActorForm frm = new ActorForm();
41 //         frm.setVisible(true);
42
43 //         Language_Utils.create_language_table();
44 //
45 Language_Utils.populate_language_table();
46 //         Language_Utils.read_language_table();
47 //         LanguageForm frm = new LanguageForm();
48 //         frm.setVisible(true);
49
50 //         Category_Utils.create_category_table();
51 //
52 Category_Utils.populate_category_table();
53 //         Category_Utils.read_category_table();
54 //         CategoryForm frm = new CategoryForm();
55 //         frm.setVisible(true);
56
57 //         Film_Utils.create_film_table();
58 //         Film_Utils.populate_film_table();
59 //         Film_Utils.read_film_table();
60 //         FilmForm frm = new FilmForm();
61 //         frm.setVisible(true);
62
63 //
64 FilmActor_Utils.create_film_actor_table();
   //
   FilmActor_Utils.populate_film_actor_table();
   //         FilmActor_Utils.read_film_actor_table();
   //         FilmActorForm frm = new FilmActorForm();
   //         frm.setVisible(true);

   //
   FilmCategory_Utils.create_film_category_table();
   //
   FilmCategory_Utils.populate_film_category_table();
   //
   FilmCategory_Utils.read_film_category_table();
   //         FilmCategoryForm frm = new
   FilmCategoryForm();
   //         frm.setVisible(true);

   //         Country_Utils.create_country_table();
   //         Country_Utils.populate_country_table();
   //         Country_Utils.read_country_table();
```

```
//        CountryForm frm = new CountryForm();
//        frm.setVisible(true);

//        City_Utils.create_city_table();
//        City_Utils.populate_city_table();
//        City_Utils.read_city_table();
//        CityForm frm = new CityForm();
//        frm.setVisible(true);

//        Address_Utils.create_address_table();
//        Address_Utils.populate_address_table();
//        Address_Utils.read_address_table();
//        AddressForm frm = new AddressForm();
//        frm.setVisible(true);

        Store_Utils.create_store_table();
        Store_Utils.populate_store_table();
        Store_Utils.read_store_table();
    }
}
```

Run project to see the result in console:

```
Store ID    : 1
Manager ID    : 1
Address ID    : 1
Address       : Address xxx
Address2      :
District      : District xxx
City ID       : 1
Postal Code   : Post xxx
Phone         : Phone xxx
City          : Balige
Country       : Toba State
Last Update   : 2023-04-28 18:21:57.0

Store ID    : 2
Manager ID    : 2
Address ID    : 2
Address       : Balige No. 1
Address2      : Balige No. 2
District      : BLG
City ID       : 2
Postal Code   : 89222
Phone         : 0892682322
City          : Tobasa
Country       : Toba State
Last Update   : 2023-04-28 18:21:57.0
```

DESIGNING GUI
DESIGNING GUI

Step 1	In the project, create a new **JFrame Form** and name it as **StoreForm.java**. In the Design tab, add eleven **JLabels** to the form and set their corresponding **text** properties as STORE ID, MAN. STAFF ID, ADDRESS ID, ADDRESS, ADDRESS 2, DISTRICT, POSTAL CODE, CITY, COUNTRY, PHONE, and LAST UPDATE.
Step 2	Then, add eleven **JTextField** to the form and set their corresponding **Variable Name** as **jtfStoreID**, **jtfManStaffID**, **jtfAddressID**, **jtfAddress**, **jtfAddress2**, **jtfPostalCode**, **jtfPhone**, **jtfDistrict**, **jtfCity**, **jtfCountry**, and **jtfLastUpdate**.
Step 3	Then, add eleven **JButton** to the form and set their corresponding **Variable Name** as **jbFirst**, **jbPrev**, **jbNext**, **jbLast**, **jbEdit**, **jbInsert**, **jbDelete**, **jbAddress**, **jbCountryForm**, **jbCityForm**, and **jbChart**. Set their corresponding **text** properties as FIRST, PREV, NEXT, LAST, EDIT, INSERT, DELETE, ADDRESS FORM, COUNTRY FORM, CITY FORM, and CHART.
Step 4	Then, add three **JComboBoxes** to the form and set their corresponding **Variable Name** as **jcbStoreID**, **jcbAddressID**, and **jcbManStaffID**.
Step 5	Lastly, add a new **JTable** to the form set set its **Variable Name** as **jtStore**. Then, right-click on it, then choose **Table Contents...** and set the number of columns to 12 and the number of rows to 25.
Step 6	In the driver class, **Sakila.java**, create a new object of **StoreForm** class using its default constructor as shown in 63 – 64:

```
1  package sakila;
2
3  public class Sakila {
4      public static void main(String[] args) {
5  //         Utility.testConnection();
6  //         Actor_Utils.create_actor_table();
7  //         Actor_Utils.populate_actor_table();
8  //         Actor_Utils.read_actor_table();
9  //         ActorForm frm = new ActorForm();
```

```
//        frm.setVisible(true);

//        Language_Utils.create_language_table();
//
Language_Utils.populate_language_table();
//        Language_Utils.read_language_table();
//        LanguageForm frm = new LanguageForm();
//        frm.setVisible(true);

//        Category_Utils.create_category_table();
//
Category_Utils.populate_category_table();
//        Category_Utils.read_category_table();
//        CategoryForm frm = new CategoryForm();
//        frm.setVisible(true);

//        Film_Utils.create_film_table();
//        Film_Utils.populate_film_table();
//        Film_Utils.read_film_table();
//        FilmForm frm = new FilmForm();
//        frm.setVisible(true);

//
FilmActor_Utils.create_film_actor_table();
//
FilmActor_Utils.populate_film_actor_table();
//        FilmActor_Utils.read_film_actor_table();
//        FilmActorForm frm = new FilmActorForm();
//        frm.setVisible(true);

//
FilmCategory_Utils.create_film_category_table();
//
FilmCategory_Utils.populate_film_category_table();
//
FilmCategory_Utils.read_film_category_table();
//        FilmCategoryForm frm = new
FilmCategoryForm();
//        frm.setVisible(true);

//        Country_Utils.create_country_table();
//        Country_Utils.populate_country_table();
//        Country_Utils.read_country_table();
//        CountryForm frm = new CountryForm();
//        frm.setVisible(true);

//        City_Utils.create_city_table();
//        City_Utils.populate_city_table();
//        City_Utils.read_city_table();
//        CityForm frm = new CityForm();
//        frm.setVisible(true);

//        Address_Utils.create_address_table();
```

```
64          //        Address_Utils.populate_address_table();
65          //        Address_Utils.read_address_table();
66          //        AddressForm frm = new AddressForm();
            //        frm.setVisible(true);

            //        Store_Utils.create_store_table();
            //        Store_Utils.populate_store_table();
            //        Store_Utils.read_store_table();
                    StoreForm frm = new StoreForm();
                    frm.setVisible(true);
                }
            }
```

Step 8

In **StoreForm**'s constructor, invoke **setLookAndFeel()** to set the look and feel of the form as shown in line 17.

```
1   package sakila;
2
3   import java.awt.Toolkit;
4   import java.awt.event.ActionEvent;
5   import
6   java.awt.event.ActionListener;
7   import javax.swing.JButton;
8   import javax.swing.JComboBox;
9   import javax.swing.JMenuItem;
10  import javax.swing.JOptionPane;
11  import javax.swing.JPopupMenu;
12  import javax.swing.JTable;
13  import javax.swing.JTextField;
14
15  public class StoreForm extends
16  javax.swing.JFrame {
17      public StoreForm() {
18          initComponents();
19
20  Utility.setLookAndFeel(this);
        }
        //...
    }
```

Run the project to see the store form as shown in Figure 11.1.

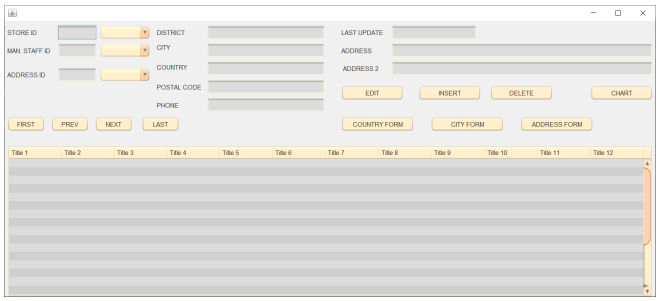


Figure 11.1 The layout of store form

Step 9

In **StoreForm.java**, define **getter()** method for every object in the form and place them outside and beneath its default constructor:

```
1       //Getter method for jtfStoreID
2       public JTextField
3   getJTFStoreID(){
4           return this.jtfStoreID;
5       }
6
7       //Getter method for jcbStoreID
8       public JComboBox
9   getJCBStoreID(){
10          return this.jcbStoreID;
11      }
12
13      //Getter method for
14  jtfManStaffID
15      public JTextField
16  getJTFManStaffID(){
17          return this.jtfManStaffID;
18      }
19
20      //Getter method for
21  jcbManStaffID
22      public JComboBox
23  getJCBManStaffID(){
24          return this.jcbManStaffID;
25      }
26
27      //Getter method for
28  jtfAddressID
29      public JTextField
30  getJTFAddressID(){
31          return this.jtfAddressID;
32      }
33
34      //Getter method for
35  jcbAddressID
36
```

```
    public JComboBox
getJCBAddressID(){
        return this.jcbAddressID;
    }

    //Getter method for jtfDistrict
    public JTextField
getJTFDistrict(){
        return this.jtfDistrict;
    }

    //Getter method for jtfCity
    public JTextField getJTFCity(){
        return this.jtfCity;
    }

    //Getter method for jtfCountry
    public JTextField
getJTFCountry(){
        return this.jtfCountry;
    }

    //Getter method for
jtfPostalCode
    public JTextField
getJTFPostalCode(){
        return this.jtfPostalCode;
    }

    //Getter method for jtfPhone
    public JTextField getJTFPhone()
{
        return this.jtfPhone;
    }

    //Getter method for jtfAddress
    public JTextField
getJTFAddress(){
        return this.jtfAddress;
    }

    //Getter method for jtfAddress2
    public JTextField
getJTFAddress2(){
        return this.jtfAddress2;
    }

    //Getter method for
jtfLastUpdate
    public JTextField
getJTFLastUpdate(){
        return this.jtfLastUpdate;
    }
```

```
91
92      //Getter method for jtAddress
93      public JTable getJTStore(){
94          return this.jtStore;
95      }
96
97      //Getter method for jbEdit
98      public JButton getJBEdit(){
99          return this.jbEdit;
100     }
101
102     //Getter method for jbInsert
103     public JButton getJBInsert(){
104         return this.jbInsert;
105     }
106
107     //Getter method for jbDelete
108     public JButton getJBDelete(){
109         return this.jbDelete;
110     }
111
112     //Getter method for jbChart
113     public JButton getJBChart(){
114         return this.jbChart;
115     }
116
117     //Getter method for jbFirst
118     public JButton getJBFirst(){
119         return this.jbFirst;
120     }
121
122     //Getter method for jbPrev
123     public JButton getJBPrev(){
124         return this.jbPrev;
125     }
126
127     //Getter method for jbNext
128     public JButton getJBNext(){
129         return this.jbNext;
    }

    //Getter method for jbLast
    public JButton getJBLast(){
        return this.jbLast;
    }

    //Getter method for
jbAddressForm
    public JButton
getJBAddressForm(){
        return this.jbAddressForm;
    }
```

```
    //Getter method for
jbCountryForm
    public JButton
getJBCountryForm(){
        return this.jbCountryForm;
    }

    //Getter method for jbCityForm
    public JButton getJBCityForm(){
        return this.jbCityForm;
    }
```

POPULATING TABLE AND COMBOBOXES
POPULATING TABLE AND COMBOBOXES

Step 1

In **Store_Utils.java**, add two new methods: **get_store_list()** and **sh** methods related to displaying store data in a GUI application. Here's what

1. **get_store_list()**: This method takes in a **StoreForm** object, a SQI establishes a connection to a database, creates a prepared statem executes the statement to retrieve data from the database. The r **Store** objects which are added to an **ArrayList**. The method returr
2. **show_table_store()**: This method takes in a **StoreForm** object a creates a **DefaultTableModel** with 12 columns and sets the **DefaultTableModel** as the model for a table component in the **S** iterates through the **ArrayList** of **Store** objects, retrieves the dat row to the **DefaultTableModel**. Finally, the method returns the up

```
1   private static ArrayList<Store> get_store_list(StoreForm
2  frm, String sql, String item){
3          ArrayList<Store> list = new ArrayList<>();
4
5          try(Connection conn = getConnection();
6              PreparedStatement ps = conn.prepareStatement(sql))
7  {
8              if (item.equalsIgnoreCase("none")==false) {
9                   ps.setString(1,item);
10             }
11             ResultSet rs = ps.executeQuery();
12
13             Store obj;
14             while(rs.next()){
15                 //Using twelve-params constructor
16                 obj = new Store(rs.getInt("store_id"),
17                         rs.getInt("manager_staff_id"),
```

```
                        rs.getInt("address_id"),
                        rs.getString("address"),
                        rs.getString("address2"),
                        rs.getString("district"),
                        rs.getInt("city_id"),
                        rs.getString("postal_code"),
                        rs.getString("phone"),
                        rs.getString("city"),
                        rs.getString("country"),
                        rs.getTimestamp("last_update"));

                list.add(obj);
            }
        }catch (SQLException ex){
            JOptionPane.showMessageDialog(frm,
ex.getMessage(),
                "ERROR",JOptionPane.ERROR_MESSAGE);
        }
        return list;
    }

    private static void show_table_store(StoreForm frm,
ArrayList<Store> list) throws SQLException{
        DefaultTableModel model = new DefaultTableModel(0,0);

        String header[] = {"Store ID", "Manager Staff ID",
"Address ID", "Address", "Address 2", "District", "City ID",
"Postal Code", "Phone", "City", "Country", "Last Update"};


model.setColumnIdentifiers(set_column_header(frm.getJTStore(),
header));
        frm.getJTStore().setModel(model);

        Object[] row = new Object[12];

        for(int i=0; i<list.size(); i++){
            row[0] = list.get(i).getStoreID();
            row[1] = list.get(i).getManagerID();
            row[2] = list.get(i).getAddressID();
            row[3] = list.get(i).getAddress();
            row[4] = list.get(i).getAddress2();
            row[5] = list.get(i).getDistrict();
            row[6] = list.get(i).getCityID();
            row[7] = list.get(i).getPostalCode();
            row[8] = list.get(i).getPhone();
            row[9] = list.get(i).getCity();
            row[10] = list.get(i).getCountry();
            row[11] = list.get(i).getLastUpdate();

            model.addRow(row);
        }
    }
```

Step 2	In **Store_Utils.java**, define **refresh_controls()** method. It updates t StoreForm. Here's what this method does: 1. **frm.setLocationRelativeTo(null)** sets the location of the **Stor** screen. 2. **frm.setTitle("STORE FORM")** sets the title of the **StoreForm** ob 3. **table_renderer(frm.getJTStore())** applies a custom table rend **StoreForm** object, to make alternating row colors. 4. **ArrayList<Store> list = get_store_list(frm, Query_Store.get_ store_id", "none")** retrieves the list of **Store** objects from the method, with a SQL query that retrieves store data joined with **ad** retrieved data is ordered by store ID. 5. **show_table_store(frm, list)** populates the table component in t from the **ArrayList** of **Store** objects. 6. populate_combobox(sql_store_id, frm.getJCBStoreID(), frm), frm.getJCBManStaffID(), frm), and populate_combobox(sql_add_id populate a **JComboBox** in the **StoreForm** object with data retrie query. The SQL queries retrieve store IDs, manager staff IDs, and a 7. If an **SQLException** is caught in the try block, a message dialog is Overall, this method updates the content of various controls in the **Store** data from a database and populating **JComboBoxes** and a **JTable**.

```
1     public static void
2  refresh_controls(StoreForm frm){
3
4  frm.setLocationRelativeTo(null);
5          frm.setTitle("STORE FORM");
6
7          //Shows the content of
8  address table and populates
9  combobox
10         try{
11             //Makes alternating
12 color for table rows
13
14 table_renderer(frm.getJTStore());
15
16             //Populates table
17             ArrayList<Store> list =
18 get_store_list(frm,
19 Query_Store.get_sql_store_joint() +
20 " ORDER BY store_id", "none");
21             show_table_store(frm,
22 list);
23
24             //Populates jcbStoreID
25
```

```
            String sql_store_id =
 "SELECT store_id FROM store ORDER
BY store_id";

populate_combobox(sql_store_id,
frm.getJCBStoreID(), frm);

            //Populates
jcbManStaffID
            String sql_staff_id =
 "SELECT manager_staff_id FROM
store ORDER BY manager_staff_id";

populate_combobox(sql_staff_id,
frm.getJCBManStaffID(), frm);

            //Populates
jcbAddressID
            String sql_add_id =
 "SELECT address_id FROM address
ORDER BY address_id";

populate_combobox(sql_add_id,
frm.getJCBAddressID(), frm);

        }catch (SQLException ex){

JOptionPane.showMessageDialog(frm,
ex.getMessage(),

 "ERROR",JOptionPane.ERROR_MESSAGE)
;
        }
    }
```

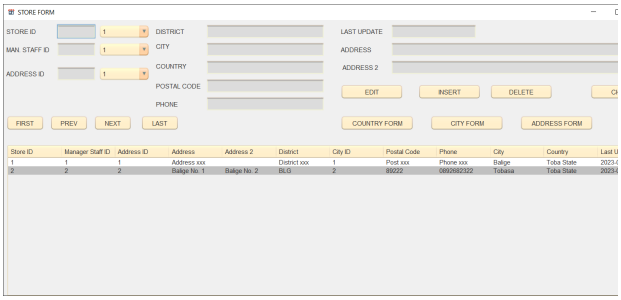


Figure 11.2 The content of joined **store**, **address**, **country** and **ci**

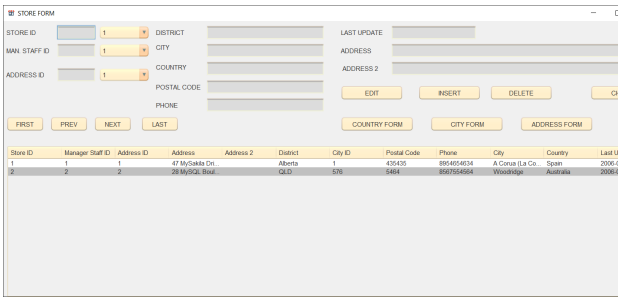


Figure 11.3 The the content of joined **store**, **address**, **country** and **city** available in the internet displayed in **jtSt**

Step 3

In **StoreForm**'s default constructor, the **Store_Utils.refresh_controls(th** the controls in the **StoreForm** with data from a database using the **re Store_Utils** class.

The **this.setIconImage()** method sets the icon of the **S this.setDefaultCloseOperation(this.HIDE_ON_CLOSE)** method sets th to hide the form instead of exiting the application when the close button is

```
1  package sakila;
2  import java.awt.Toolkit;
3  import java.awt.event.ActionEvent;
4  import java.awt.event.ActionListener;
5  import java.io.IOException;
6  import java.text.ParseException;
7  import java.util.logging.Level;
8  import java.util.logging.Logger;
9  import javax.swing.JButton;
10 import javax.swing.JComboBox;
11 import javax.swing.JLabel;
12 import javax.swing.JTable;
13 import javax.swing.JTextField;
14
15 public class StoreForm extends javax.swing.JFrame {
16     public StoreForm() {
17         initComponents();
18         Utility.setLookAndFeel(this);
19         Store_Utils.refresh_controls(this);
20
21  this.setIconImage(Toolkit.getDefaultToolkit().getImage(getClass().
22 ;
23         this.setDefaultCloseOperation(this.HIDE_ON_CLOSE);
24     }
25     //...
26 }
```

Step

Run the project to see the content of joined **store**, **address**, **country** and

4	shown in Figure 11.2. If you use the data from **Sakila** MySQL database available in the inter joined **store**, **address**, **country** and **city** tables displayed in **jtStore** as sh

DISPLAYING AND NAVIGATING DATA ROW BY ROW
DISPLAYING AND NAVIGATING DATA ROW BY ROW

Step 1

In **Store_Utils**, define two new methods named **clear_address_co** **display_address_data()**. The **clear_address_controls()** method is a pr that takes a **StoreForm** object as its parameter. It is used to clear the tex form that display address data. This is useful in cases where no data is database query.

The **display_address_data()** method is another private method t **StoreForm** object, a SQL query string, and an item object as its par **getConnection()** method is called to establish a connection to the datab a **PreparedStatement** object is created to execute the SQL query. The set as a parameter to the query using **ps.setObject(1,item)**.

The **ResultSet** object rs is used to retrieve the results of the query. If **rs.n** **false**, it means that no rows were found in the query. In th **clear_address_controls()** method is called to clear the form fields.

If rows are found in the query, the do-while loop is used to iterate throug the result set. Within the loop, the form fields are populated with the va result set using methods like **rs.getString()** and **rs.ge** **find_combo_value_selected()** method is called to determine which iter box was selected based on the **address_id** value from the result set.

If an SQL exception is caught, a message dialog is displayed with the erro

```
1     private static void clear_address_controls(StoreForm frm){
2         frm.getJTFAddressID().setText("");
3         frm.getJTFAddress().setText("");
4         frm.getJTFAddress2().setText("");
5         frm.getJTFDistrict().setText("");
6         frm.getJTFPhone().setText("");
7         frm.getJTFPostalCode().setText("");
8         frm.getJTFCity().setText("");
9         frm.getJTFCountry().setText("");
10    }
11
12   //Displays address data result row by row
13
```

```
14      private static <T> void display_address_data(StoreForm frm, Str
15  sql, T item){
16          try(Connection conn = getConnection()){
17              PreparedStatement ps = conn.prepareStatement(sql);
18              ps.setObject(1,item);
19              ResultSet rs = ps.executeQuery();
20
21              if (!rs.next()) {
22                  // no row found, clear the form fields
23                  clear_address_controls(frm);
24                  return;
25              }
26
27              do{
28
29  frm.getJTFAddressID().setText(String.valueOf(rs.getInt("address_id"
30                  frm.getJTFAddress().setText(rs.getString("address")
31                  frm.getJTFAddress2().setText(rs.getString("address2
32                  frm.getJTFDistrict().setText(rs.getString("district
33                  frm.getJTFPhone().setText(rs.getString("phone"));
34
35  frm.getJTFPostalCode().setText(rs.getString("postal_code"));
36                  frm.getJTFCity().setText(rs.getString("city"));
37                  frm.getJTFCountry().setText(rs.getString("country")
38
39                  // Determines item selected from jcbAddressID
40                  find_combo_value_selected(frm.getJCBAddressID(),
41  rs.getInt("address_id"));
42
43              }while(rs.next());
44
45              rs.close();
46              ps.close();
47          }catch(SQLException ex){
48              JOptionPane.showMessageDialog(frm, ex.getMessage(),
49                      "ERROR",JOptionPane.ERROR_MESSAGE);
            }
        }
```

Step 2

Still in the same class, define another method named **jcbAddress_ha**
jcbAddress_handler() method is a public static method that takes a **Stor**
as its parameter. It is used as an event handler for the
frm.getJCBAddressID().

The **Object** item variable is assigned the currently selected item in th
using the **getSelectedItem()** method.

The **display_address_data()** method is called with the frm parameter,
string from **Query_Address.get_sql_address_joint()**, and the item
parameters.

```
    public static void
jcbAddress_handler(StoreForm frm) {
        Object item =
frm.getJCBAddressID().getSelectedItem();
        display_address_data(frm,
Query_Address.get_sql_address_joint() +
" WHERE address_id = ?", item);
    }
```

Step 3

In **StoreForm**, double click on **jcbAddressID** combobox and define its c event handler as follows:

```
    private void
jcbAddressIDActionPerformed(java.awt.event.ActionEvent
evt) {
        Store_Utils.jcbAddress_handler(this);
    }
```

It is a private method that takes an **ActionEvent** object as its parameter. I when an action is performed on the **jcbAddressID** combo box, such as wh selected.

The **jcbAddress_handler()** method from **Store_Utils** is called with parameter. The **this** keyword refers to the current instance of the **StoreFo**

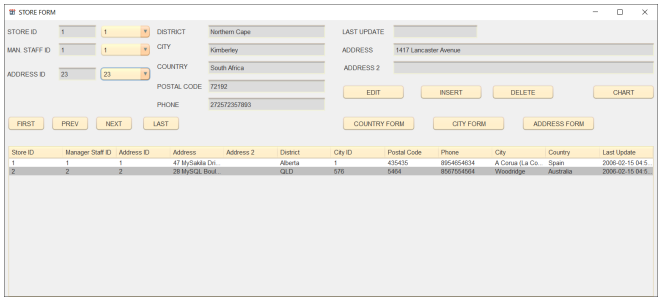


Figure 11.4 Displaying row by row the content of joined **address**, **city** a tables

Step 4

Run the project. Choose one of items in **jcbAddressID** combobox to see rc content of joined **address**, **city** and **country** tables as shown in Figure 11.

Step 5

In **Store_Utils** class, define a new method named **jcbManStaff_handler()**

```
    public static void jcbManStaff_handler(StoreForm
frm) {

```

```
4           Object item =
frm.getJCBManStaffID().getSelectedItem();

frm.getJTFManStaffID().setText(String.valueOf(item));
    }
```

It is a public static method that takes a **StoreForm** object as its parameter an event handler for the combo box **frm.getJCBManStaffID()**.

The **Object** item variable is assigned the currently selected item in th using the **getSelectedItem()** method. The **String.valueOf(item)** metho convert the selected item to a string. The string value of the selected item text of the **frm.getJTFManStaffID()** text field using the **setText()** method

Step 6

In **StoreForm**, double click on **jcbManStaffID** combobox and define its c event handler as follows:

```
1     private void
2 jcbManStaffIDActionPerformed(java.awt.event.ActionEvent
3 evt) {
          Store_Utils.jcbManStaff_handler(this);
    }
```

Step 7

Run the project. Choose one of items in **jcbManStaffID** combobox to se the selected item from the **jcbManStaffID** combo box and sets its string text of the **frm.getJTFManStaffID()** text field. as shown in Figure 11.5.

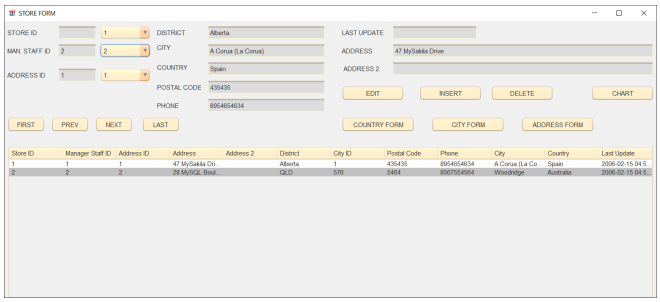


Figure 11.5 Retrieving the selected item from the **jcbManStaffID** combo its string value as the text of the **frm.getJTFManStaffID()** text f

Step 8

In **Store_Utils** class, define a new method named **display_store_data()**. parameters: a **StoreForm** object, an SQL query string, and an item of type

The method first obtains a database connection using the **getConnection** then prepares a **PreparedStatement** using the supplied SQL query and parameter of the prepared statement to the item parameter.

The **PreparedStatement** is executed using the **executeQuery()** m returns a **ResultSet** object containing the results of the query.

If the **ResultSet** does not contain any rows, the method clears the fo returns. Otherwise, the method iterates over the rows in the **ResultSe** while loop.
For each row, the method updates the text fields in the **StoreForm** objec from the **ResultSet** using the **setText()** method. It also **find_combo_value_selected()** method to determine the selected it **JComboBoxes** in the StoreForm object based on values from the **ResultS**

After iterating over all rows in the **ResultSet**, the method closes the **R** **PreparedStatement** and handles any **SQLExceptions** that may oc **JOptionPane** message dialog.

```
1    //Displays store data result row by row
2     private static <T> void display_store_data(StoreForm frm, Strin
3 item){
4         try(Connection conn = getConnection()){
5             PreparedStatement ps = conn.prepareStatement(sql);
6             ps.setObject(1,item);
7             ResultSet rs = ps.executeQuery();
8 
9             if (!rs.next()) {
10                // no row found, clear the form fields
11                frm.getJCBManStaffID().setSelectedIndex(-1);
12                frm.getJCBAddressID().setSelectedIndex(-1);
13                frm.getJTFLastUpdate().setText("");
14                return;
15            }
16 
17            do{
18 
19 frm.getJTFStoreID().setText(String.valueOf(rs.getInt("store_id")));
20 
21 frm.getJTFLastUpdate().setText(String.valueOf(rs.getTimestamp("last
22 
23                // Determines item selected from jcbStoreID
24                find_combo_value_selected(frm.getJCBStoreID(),
25 rs.getInt("store_id"));
26 
27                // Determines item selected from jcbManStaffID
28                find_combo_value_selected(frm.getJCBManStaffID(),
29 rs.getInt("manager_staff_id"));
30 
31                // Determines item selected from jcbAddressID
32                find_combo_value_selected(frm.getJCBAddressID(),
33 rs.getInt("address_id"));
34 
35            }while(rs.next());
36 
```

```
            rs.close();
            ps.close();
        }catch(SQLException ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }
```

Step 9

In the same class, define another method named **jcbStore_handler()**.

```
    public static void
jcbStore_handler(StoreForm frm) {
        Object item =
frm.getJCBStoreID().getSelectedItem();
        display_store_data(frm,
SQL_ID, item);
    }
```

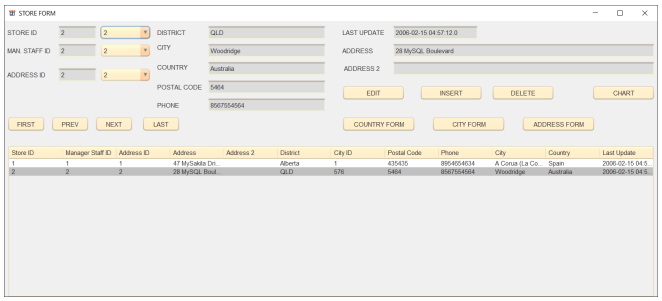


Figure 11.6 Displaying row by row the content of joined **store**, **addres**
country tables

Step 10

In **StoreForm**, double click on **jcbStoreID** combobox and define its c
event handler as follows:

```
    private void
jcbStoreIDActionPerformed(java.awt.event.ActionEvent
evt) {
        Store_Utils.jcbStore_handler(this);
    }
```

Step 11

Run the project. Choose one of items in **jcbStoreID** combobox to see ro
content of joined **store**, **address**, **city** and **country** tables as shown in Fig

Step 12

Define four navigating methods in **Store_Utils** class. These are methods
the rows in a **JComboBox** in a **StoreForm** object. They take a **StoreFor**

parameter.

The methods use a **currentIndex** variable to keep track of the current selected item in the **JComboBox**. The **currentIndex** variable is initially s updated by each navigation method as needed.

The **show_first_row()** method sets the **currentIndex** variable to 0, gets the item at the first index of the **JComboBox** using the **getItemAt()** met it to a String, and calls the display_store_data() method with the **StoreF** SQL query string constant called SQL_ID, and the selected item.

The **show_last_row()** method gets the index of the last item in the **JCom** the **getItemCount()** method, subtracts 1 to get the last index, sets the **c** variable to the last index, gets the value of the item at the last ind **getItemAt()** method, converts it to a **String**, and calls the **display_** method with the StoreForm object, SQL_ID constant, and the selected item

The **show_prev_row()** method decrements the currentIndex variable by is less than the first index, and returns if it is. If not, it gets the value of th current index using the **getItemAt()** method, converts it to a **String**, **display_store_data()** method with the **StoreForm** object, SQL_ID cons selected item.

The **show_next_row()** method increments the **currentIndex** variable b index of the last item in the **JComboBox** using the **getItemCount()** meth 1 to get the last index, checks if the currentIndex variable is greater index, and returns if it is. If not, it gets the value of the item at the curren the **getItemAt()** method, converts it to a String, and calls the **display_** method with the **StoreForm** object, SQL_ID constant, and the selected ite

```
1     public static void show_first_row(StoreForm frm){
2         String item =
3 String.valueOf(frm.getJCBStoreID().getItemAt(FIRST_INDEX));
4         display_store_data(frm, SQL_ID, item);
5         currentIndex = FIRST_INDEX;
6     }
7
8     public static void show_last_row(StoreForm frm){
9         int endIndex = frm.getJCBStoreID().getItemCount() -
10 1;
11        String item =
12 String.valueOf(frm.getJCBStoreID().getItemAt(endIndex));
13        display_store_data(frm, SQL_ID, item);
14        currentIndex = endIndex;
15    }
16
17    public static void show_prev_row(StoreForm frm){
18        currentIndex--;
19        if(currentIndex < FIRST_INDEX){
```

```
            currentIndex = FIRST_INDEX;
            return;
        }
        String item =
String.valueOf(frm.getJCBStoreID().getItemAt(currentIndex));
        display_store_data(frm, SQL_ID, item);
    }

    public static void show_next_row(StoreForm frm){
        int endIndex = frm.getJCBStoreID().getItemCount() -
1;
        currentIndex++;
        if(currentIndex > endIndex){
            currentIndex = endIndex;
            return;
        }
        String item =
String.valueOf(frm.getJCBStoreID().getItemAt(currentIndex));
        display_store_data(frm, SQL_ID, item);
    }
```

Step 13

Then in **StoreForm**, double click on each navigation buttons to corresponding event handler:

```
    private void
jbFirstActionPerformed(java.awt.event.ActionEvent
evt) {
        Store_Utils.show_first_row(this);
    }

    private void
jbPrevActionPerformed(java.awt.event.ActionEvent
evt) {
        Store_Utils.show_prev_row(this);
    }

    private void
jbNextActionPerformed(java.awt.event.ActionEvent
evt) {
        Store_Utils.show_next_row(this);
    }

    private void
jbLastActionPerformed(java.awt.event.ActionEvent
evt) {
        Store_Utils.show_last_row(this);
    }
```

These are event handler methods for the "First", "Prev", "Next", and "Las
the **StoreForm** GUI. When one of these buttons is clicked, the correspond

called, which in turn calls a method from the **Store_Utils** class to display previous row, next row, or last row of data in the store table, respectiv keyword in the method calls refers to the current **StoreForm** instance, wh as a parameter to the **Store_Utils** methods.

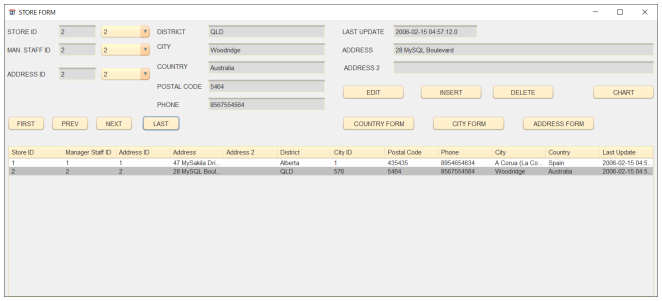


Figure 11.7 User clicks on one or more navigation buttons on stor

Step 14 | Run the project. Click on one or more navigation buttons to see the resul Figure 11.7.

Step 15 | Define **mouse_pressed_handler()** method in **Store_Utils** class. This is handling a mouse press event in the **JTable** component in the **StoreForm** first checks if a row in the **JTable** is selected. If not, it displays a message user to select a row. If a row is selected, the method gets the value of the in the selected row (the store ID) and calls the **display_store_data()** met **Store_Utils** class to display the data for that store ID.

The **getConnection()** method is used to establish a connection to the da try-with-resources block is used to ensure the connection is properly clos code is finished executing. If an **SQLException** is caught, an error displayed, including the exception message and stack trace.

```
1     public static void mouse_pressed_handler(StoreForm frm) {
2         Objects.requireNonNull(frm, "frm must not be null");
3
4         int selectedIndex = frm.getJTStore().getSelectedRow();
5         if (selectedIndex == -1) {
6             JOptionPane.showMessageDialog(frm, "Please select a row to view its
7 data.",
8                 "No row selected", JOptionPane.INFORMATION_MESSAGE)
9             return;
10        }
11
12        try (Connection conn = getConnection()) {
13            String id =
14 String.valueOf(frm.getJTStore().getModel().getValueAt(selectedIndex
15 0));
16
17            // Displays store data
18
```

```
            display_store_data(frm, SQL_ID, id);

        } catch (SQLException ex) {

Logger.getLogger(StoreForm.class.getName()).log(Level.SEVERE, "Erro
displaying store data", ex);
            String message = "Error displaying store data: " +
ex.getMessage();
            String stackTrace = Arrays.toString(ex.getStackTrace())
            JOptionPane.showMessageDialog(frm, message + "\n\n" +
stackTrace, "ERROR", JOptionPane.ERROR_MESSAGE);
        }
    }
```

Step 16

Right click on **jtStore**. Then, choose **Events > Mouse > mousePresse** event handler:

```
    private void
jtStoreMousePressed(java.awt.event.MouseEvent
evt) {

 Store_Utils.mouse_pressed_handler(this);
    }
```

Step 17

Run the project. Double click on any row in **jtStore** table. You corresponding row of joined **store**, **address**, **country** and **city** tables textfields and comboboxes as shown in Figure 11.8.

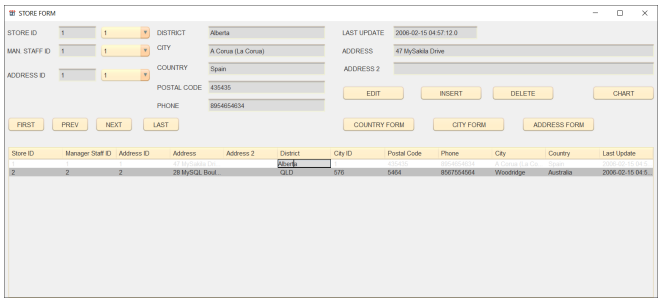


Figure 11.8 User double-clicks on any row in **jtStore**

UPDATING RECORD
UPDATING RECORD

Step 1

In **Store_Utils** class, define a new method named **update_row_by_store_id()**. This method updates a row of data in the **store** table in the database by **store_id**. It takes in a **StoreForm** object, as well as three integers: **store_id**, **man_id**, and **add_id**, which represent the values to update for the **manager_staff_id** and **address_id** columns, respectively.

First, it creates a database connection using the **getConnection()** method. It then prepares two SQL statements: one to select the **store_id** from the **store** table using the given **store_id**, and another to update the corresponding row in the **store** table with the new **manager_staff_id** and **address_id** values.

The method executes the select statement using the **address_id** parameter as input, and checks whether the result set is empty. If the result set is empty, it displays an error message indicating that the given **store_id** does not exist in the database.

If the result set is not empty, the method creates a new **Store** object using the given **store_id**, **man_id**, **add_id**, and a timestamp representing the current time. It then sets the parameters of the update statement to the values of the corresponding columns in the Store object and executes the update statement.

Finally, the method closes the result set, statements, and database connection. If an SQL exception or number format exception occurs during execution, the method logs an error message and displays a corresponding error message dialog box.

```
1     //Updates row of data in store tabel by store_id
2     public static void update_row_by_store_id(StoreForm frm,
3 int store_id, int man_id, int add_id) throws SQLException{
4         Connection conn = getConnection();
5         ResultSet rs = null;
6         String query_id = "SELECT store_id FROM store WHERE
7 store_id = ?";
8         String update_query = """
9             UPDATE store SET manager_staff_id = ?,
10            address_id = ? WHERE store_id = ?""";
11        try(PreparedStatement idPs =
12 conn.prepareStatement(query_id,
13
14 ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_READ_ONLY);
15        PreparedStatement updatePS =
16 conn.prepareStatement(update_query,
17                 ResultSet.TYPE_SCROLL_SENSITIVE,
18 ResultSet.CONCUR_UPDATABLE))
19        {
20            idPs.setInt(1, store_id);
```

```
            if(!idPs.execute()){
                String message = "Can't find store_id " +
store_id;

                JOptionPane.showMessageDialog(frm, message,
                    "ERROR",JOptionPane.ERROR_MESSAGE);
            } else{
                rs = idPs.getResultSet();
                rs.next();

                //Creates a Store object using four-params
constructor
                Store obj = new Store(store_id, man_id,
add_id, new Timestamp(System.currentTimeMillis()));
                updatePS.setInt(1, obj.getManagerID());
                updatePS.setInt(2, obj.getAddressID());
                updatePS.setInt(3, obj.getStoreID());

                updatePS.executeUpdate();
                rs.close();
                updatePS.close();
                idPs.close();
                conn.close();
            }
        }catch(SQLException ex){

Logger.getLogger(StoreForm.class.getName()).log(Level.SEVERE,
"Error updating store data", ex);
            String message = "Error updating store data: " +
ex.getMessage();
            String stackTrace =
Arrays.toString(ex.getStackTrace());
            JOptionPane.showMessageDialog(null, message +
"\n\n" + stackTrace, "ERROR", JOptionPane.ERROR_MESSAGE);
        }catch(java.lang.NumberFormatException ex){

Logger.getLogger(StoreForm.class.getName()).log(Level.SEVERE,
"Invalid Input", ex);
            String message = "Invalid Input: " +
ex.getMessage();
            String stackTrace =
Arrays.toString(ex.getStackTrace());
            JOptionPane.showMessageDialog(null, message +
"\n\n" + stackTrace, "ERROR", JOptionPane.ERROR_MESSAGE);
        }
    }
```

Step 2	Then in the same class, define a new method **read_inputs()**. It takes a **StoreForm** object as an input parameter and returns a **HashMap** of String keys and values.

The method first initializes an empty **HashMap** called **input_data**. Then, it reads the user input from the **StoreForm** object by extracting the selected store ID, manager staff ID, and address ID. It then validates the input data by checking if the integer values of each input are greater than zero. If any of the input data is invalid, the method displays an error message in a dialog box and throws an exception.

Finally, the method adds the validated input data to the **input_data HashMap** using the corresponding key-value pairs and returns the **HashMap**.

```
1   private static HashMap<String, String>
2  read_inputs(StoreForm frm) {
3          HashMap<String, String> input_data = new
4  HashMap<>();
5          String store_id =
6  String.valueOf(frm.getJCBStoreID().getSelectedItem());
7          String man_id =
8  frm.getJTFManStaffID().getText();
9          String add_id =
10 String.valueOf(frm.getJCBAddressID().getSelectedItem());
11
12         // Validate user input
13         int add_id_int = 0;
14         try {
15             add_id_int = Integer.parseInt(add_id);
16             if (add_id_int <= 0) {
17                 throw new
18 IllegalArgumentException("Address ID cannot be negative
19 or zero");
20             }

21         } catch (NumberFormatException ex) {
22             JOptionPane.showMessageDialog(frm, "Invalid
23 Address ID: " + add_id,
24             "Error", JOptionPane.ERROR_MESSAGE);
25             throw ex;
26         } catch (IllegalArgumentException ex) {
27             JOptionPane.showMessageDialog(frm,
28 ex.getMessage(),
29             "Error", JOptionPane.ERROR_MESSAGE);
30             throw ex;
31         }
32
33         int store_id_int = 0;
34         try {
35             store_id_int = Integer.parseInt(store_id);
36             if (store_id_int <= 0) {
37                 throw new
38 IllegalArgumentException("Store ID cannot be negative or
```

```
zero");
            }
        } catch (NumberFormatException ex) {
            JOptionPane.showMessageDialog(frm, "Invalid
Store ID: " + store_id,
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        } catch (IllegalArgumentException ex) {
            JOptionPane.showMessageDialog(frm,
ex.getMessage(),
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        }

        int man_id_int = 0;
        try {
            man_id_int = Integer.parseInt(man_id);

            if (man_id_int <= 0) {
                throw new
IllegalArgumentException("Manager Staff ID cannot be
negative or zero");
            }
        } catch (NumberFormatException ex) {
            JOptionPane.showMessageDialog(frm, "Invalid
Manager Staff ID: " + man_id,
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        } catch (IllegalArgumentException ex) {
            JOptionPane.showMessageDialog(frm,
ex.getMessage(),
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        }

        input_data.put("add_id", add_id);
        input_data.put("store_id", store_id);
        input_data.put("man_id", man_id);

        return input_data;
    }
```

Step 3

Still in the same class, define another method named **edit_actual()**. It is called when the user clicks the "Edit" button on the **StoreForm**. It reads the input data from the form using the **read_inputs()** method and calls the **update_row_by_store_id()** method to update the store data in the database. If the update is successful, it calls the **refresh_controls()** method to update the form with the new data. If an exception is caught, it displays an error message to the user.

```
    private static void edit_actual(StoreForm
frm){
        try{
            HashMap<String, String>
input_data = read_inputs(frm);
            int add_id =
Integer.parseInt(input_data.get("add_id"));
            int store_id =
Integer.parseInt(input_data.get("store_id"));
            int man_id =
Integer.parseInt(input_data.get("man_id"));

            update_row_by_store_id(frm,
store_id, man_id, add_id);

            //Refreshes all objects on form
            refresh_controls(frm);

        }catch(SQLException ex){

JOptionPane.showMessageDialog(frm,
ex.getMessage(),

 "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }
```

Step 4

Lastly, define two new methods named **enable_controls()** and **edit_handler()**. The **edit_handler()** is called when the user clicks the "EDIT" button on the **StoreForm**.

If the button text is "EDIT", the method changes the text to "CONFIRM" and disables all input controls on the form by calling **enable_controls(false, frm)**.

If the button text is "CONFIRM", the method calls **edit_actual(frm)** to perform the actual editing of the store data, updates the button text to "EDIT", and re-enables all input controls by calling **enable_controls(true, frm)**.

```
    private static void
enable_controls(boolean state, StoreForm
frm){
        frm.getJBFirst().setEnabled(state);
        frm.getJBPrev().setEnabled(state);
        frm.getJBNext().setEnabled(state);
        frm.getJBLast().setEnabled(state);
        frm.getJBInsert().setEnabled(state);
        frm.getJBDelete().setEnabled(state);
```

```
10
11 frm.getJTFAddressID().setEnabled(state);
12
13 frm.getJTFStoreID().setEnabled(state);
14
15 frm.getJTFAddress().setEnabled(state);
16
17 frm.getJTFAddress2().setEnabled(state);
18
19 frm.getJTFDistrict().setEnabled(state);
20         frm.getJTFPhone().setEnabled(state);
21
22 frm.getJTFPostalCode().setEnabled(state);
23         frm.getJTFCity().setEnabled(state);
24
25 frm.getJTFCountry().setEnabled(state);
26     }
27
28     public static void
29 edit_handler(StoreForm frm){
30
31  if(frm.getJBEdit().getText().equals("EDIT")
32 ){
33
34 frm.getJBEdit().setText("CONFIRM");
35
36             // Disables controls
               enable_controls(false, frm);
           }

           else {
               frm.getJBEdit().setText("EDIT");

               // Actual editing
               edit_actual(frm);

               //Enables controls
               enable_controls(true, frm);
           }
       }
```

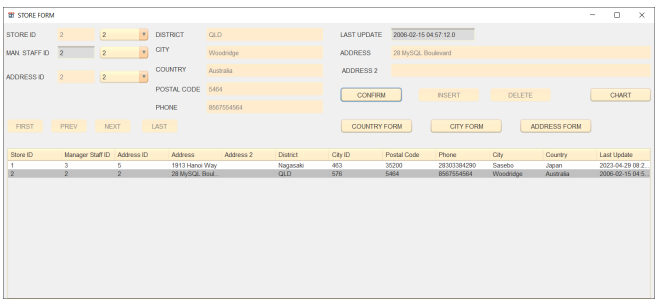
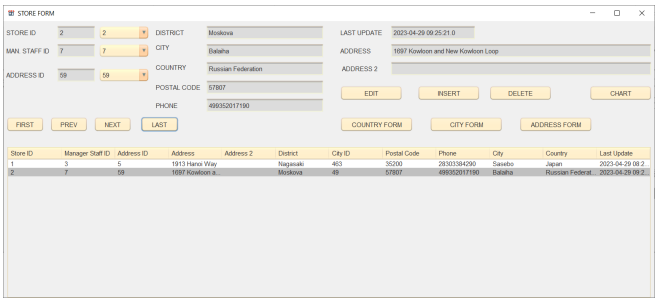


Figure 11.9 The store form is in editing state

Step 5	Run the project. Choose **store_id** using **jcbStoreID** combobox. Or, you can choose one of rows in **jtStore** (in this case, **store_id = 2**). Then, click on EDIT button as shown in Figure 11.9. Choose **address_id** using **jcbAddressID** and type **manager_staff_id** in **jtfManStaffID**. Then, click on CONFIRM button. The edited row had been saved into **store** table as shown in Figure 11.10.  **Figure 11.10** The edited row had been saved into database

UPDATING RECORD DIRECTLY ON JTABLE
UPDATING RECORD DIRECTLY ON JTABLE

Step 1	In **Store_Utility** class, define a new method na **edit_database_from_jtable()**. It handles events that occur when the edits a row in the **JTable** on the **StoreForm**. If the event type is an updat retrieves the row, column, and cell values of the edited cell and uses the update the corresponding row in the database by calling **update_row_by_store_id()** function. If there is a **SQLIntegrityConstraintViolationException** **SQLSyntaxErrorException**, it logs an error and displays an appropr error message to the user using a JOptionPane. Finally, it refreshes all objects on the form by calling the **refresh_contro** function.

```
1     public static void
2 edit_database_from_jtable(TableModelEvent e, StoreForm frm){
3         if (e.getType() == TableModelEvent.UPDATE) {
4             int row = e.getFirstRow();
5             TableModel model = (TableModel)e.getSource();
```

```
            int store_id =
Integer.parseInt(String.valueOf(model.getValueAt(row, 0)));
            int add_id =
Integer.parseInt(String.valueOf(model.getValueAt(row, 2)));
            int man_id =
Integer.parseInt(String.valueOf(model.getValueAt(row, 1)));

            try{
                update_row_by_store_id(frm, store_id, man_id,
add_id);

                //Refreshes all objects on form
                refresh_controls(frm);

            } catch (SQLIntegrityConstraintViolationException
ex) {

Logger.getLogger(StoreForm.class.getName()).log(Level.SEVERE,
"Duplicate entry", ex);
                JOptionPane.showMessageDialog(frm, "Error:
Duplicate entry\n" + ex.getMessage());
            } catch (SQLSyntaxErrorException ex) {

Logger.getLogger(StoreForm.class.getName()).log(Level.SEVERE,
"Invalid SQL syntax", ex);
                JOptionPane.showMessageDialog(frm, "Error:
Invalid SQL syntax\n" + ex.getMessage());
            } catch (SQLException ex) {

Logger.getLogger(StoreForm.class.getName()).log(Level.SEVERE,
"Database error", ex);
                JOptionPane.showMessageDialog(frm, "Error:
Database error\n" + ex.getMessage());
            }
        }
    }
```

Step 2

Create a new public class named **StoreTableModelListener**. It implem the **TableModelListener** interface. It has two fields: **jt**, which is a **JTa** and **frm**, which is a **StoreForm**.

The purpose of this class is to listen for changes in the table model and the **edit_database_from_jtable()** method from **Store_Utils** to update database accordingly. It also stops cell editing if there is an active cell ec in the **JTable**.

```
package sakila;
import
javax.swing.event.TableModelEvent;
```

```
import
javax.swing.event.TableModelListener;
import javax.swing.JTable;

public class StoreTableModelListener
implements TableModelListener {
    private final JTable jt;
    private final StoreForm frm;

    public
StoreTableModelListener(JTable jt,
StoreForm frm) {
        this.jt = jt;
        this.frm = frm;
    }

    @Override
    public void
tableChanged(TableModelEvent e) {

Store_Utils.edit_database_from_jtable(e,
frm);

        if (jt.getCellEditor() != null)
{

jt.getCellEditor().stopCellEditing();
        }
    }
}
```

Step 3

Right click on **jtStore**. Then, choose **Events > Mouse > mouseClicl** Define its event handler:

```
    private void jtStoreMouseClicked(java.awt.event.MouseEvent evt)
        // instantiate StoreTableModelListener and add it as a liste
to the table model
        StoreTableModelListener tableModelListener = new
StoreTableModelListener(this.getJTStore(), this);

 this.getJTStore().getModel().addTableModelListener(tableModelListen
;
    }
```

It is an event handler for when the user clicks on the **JTable** compor named **jtStore** in the **StoreForm**.

The code creates an instance of **StoreTableModelListener** and passes in **jtStore** and this (the current **StoreForm** instance) as arguments. Ther

	adds the **tableModelListener** as a listener to the table model of the **jtSto** This means that whenever the user edits a cell in the **jtStore**, **tableChanged()** method of the **StoreTableModelListener** will be ca which in turn will call the **edit_database_from_jtable()** method in **Store_Utils** class to update the database with the new value. Additionally, the **stopCellEditing()** method is called on the cell editor of table to commit any pending changes and stop editing.
Step 4	Run the project. Click on any cell in second column to third column in any in **jtStore** that you want to edit. Then, change it. Then, click any wl outside the corresponding cell. The edited data had been saved into databa

INSERTING NEW RECORD
INSERTING NEW RECORD

Step 1	In **Store_Utils** class, define a method named **insert_row()**. This method is used to insert a new row into the **store** table in the database. It first reads the input data from the StoreForm, which includes the manager ID and address ID, and then creates an SQL insert statement using those values. It then creates a **Store** object using those values and executes the SQL insert statement using a **PreparedStatement**. If there is a SQL exception, it catches the exception and logs the error message, and then displays an error message dialog to the user.

```
1   //Inserts new row into store table
2   private static void insert_row(StoreForm frm) throws
3   SQLException{
4           HashMap<String, String> input_data =
5   read_inputs(frm);
6           int add_id =
7   Integer.parseInt(input_data.get("add_id"));
8           int man_id =
9   Integer.parseInt(input_data.get("man_id"));
10
11          // SQL insert statement
12          String sql = """
13              INSERT INTO store(manager_staff_id, address_id)
14  VALUES(?, ?)""";
15
16          try(Connection conn = getConnection();
17
```

```
         PreparedStatement pstmt =
conn.prepareStatement(sql)){

            //Creates a Store object three-params constructor
            Store obj = new Store(man_id, add_id, new
Timestamp(System.currentTimeMillis()));
            pstmt.setInt(1,obj.getManagerID());
            pstmt.setInt(2,obj.getAddressID());

            //Executes the sql insert statement
            pstmt.executeUpdate();
        } catch (SQLException ex) {

Logger.getLogger(StoreForm.class.getName()).log(Level.SEVERE,
"Database error", ex);
            JOptionPane.showMessageDialog(frm, "Error:
Database error\n" + ex.getMessage());
        }
    }
```

Step 2

Still in **Store_Utils.java**, define **insert_actual()** and **insert_handler()** methods. The **insert_handler()** method is responsible for handling the click event of the "INSERT" button on the form. When the button is clicked, it checks whether its text is "INSERT" or "CONFIRM". If it is "INSERT", it changes the text to "CONFIRM" and disables the "Edit" button and some controls on the form. It also clears the input fields. If the text is "CONFIRM", it calls the **insert_actual()** method to actually insert the data into the database.

The insert_actual method calls the **insert_row()** method to perform the actual insertion. If successful, it refreshes the table and comboboxes on the form.

```
    private static void insert_actual(StoreForm
frm){
        try{
            insert_row(frm);

            //Refreshes table and comboboxes
            refresh_controls(frm);

        }catch(SQLException ex){
            JOptionPane.showMessageDialog(frm,
ex.getMessage(),

 "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }
```

```
16
17      public static void insert_handler(StoreForm
18  frm){
19
20   if(frm.getJBInsert().getText().equals("INSERT")
21  ){
22
23  frm.getJBInsert().setText("CONFIRM");
24
25              //Disables jbEdit
26              frm.getJBEdit().setEnabled(false);
27
28              // Disables controls
29              enable_controls(false, frm);
30
31  frm.getJCBStoreID().setEnabled(false);
32
33              // Clears controls
34              clear_address_controls(frm);
35
36              // Enables
37              frm.getJBInsert().setEnabled(true);
38
39
40
41
42
43
44
45
```

```
        }

        else {

frm.getJBInsert().setText("INSERT");

            // Actual insertion
            insert_actual(frm);

            //Enables jbEdit

frm.getJBEdit().setEnabled(true);

            //Enables controls
            enable_controls(true, frm);

frm.getJCBStoreID().setEnabled(true);
        }
    }
```

Step 3

In **StoreForm.java**, double click or create its event listener:

```
1  private void
2  jbInsertActionPerformed(java.awt
3  evt) {
       Store_Utils.insert_handl
   }
```

Step 4

Run the project. Click on INSERT b the state of store form when inserti shown in Figure 11.11.

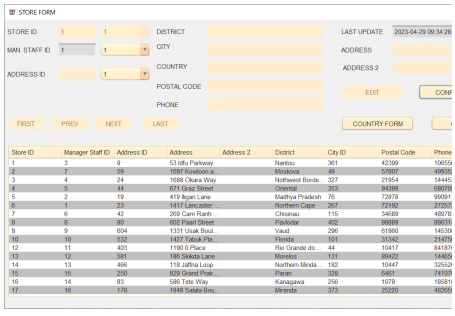


Figure 11.11 When user clicks on store form will be in state

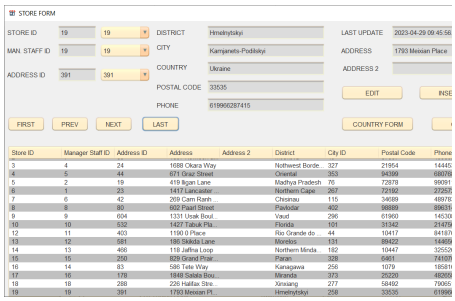


Figure 11.12 The new data had be
table

Then, choose **address_id** using **jcbA**
a new manager staff id in the te
CONFIRM button to save the new
table as shown in Figure 11.12.

DELETING RECORD

DELETING RECORD

Step 1

Then in **Store_Utils** class, define **delete_handler()** method. This m
handles the delete operation for a row in the **Store** table. It first promp
user to confirm if they want to delete the row with the selected Store ID.
user confirms the deletion, the method constructs and executes a SQL DE
statement using a **PreparedStatement** to avoid SQL injection attacks
PreparedStatement is set up to delete the row with the specified Store
the deletion is successful, the method refreshes the table and combobo
calling the **refresh_controls()** method. If there is an exception durin
deletion process, an error message is displayed to the user.

```
    public static void delete_handler(StoreForm frm){
        int dialogButton = JOptionPane.YES_NO_OPTION;
        int store_id =
Integer.parseInt(String.valueOf(frm.getJCBStoreID().getSelectedItem

        String message = String.format("Are you sure you want to de
the row Store ID: %d)", store_id);
        int answer = JOptionPane.showConfirmDialog(frm, message,
"DELETING ROW OF DATA", dialogButton);

        if(answer == JOptionPane.YES_OPTION){
            String query = """
                DELETE FROM store WHERE store_id = ?""";
            try(Connection conn = getConnection();
                PreparedStatement ps = conn.prepareStatement(query)
                // Use PreparedStatement to avoid SQL injection att
                ps.setInt(1, store_id);
                ps.executeUpdate();
```

```
                // Refresh table and comboboxes
                refresh_controls(frm);

            } catch (SQLException ex){
                JOptionPane.showMessageDialog(frm, ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
            }
        }
    }
```

Step 2	In **StoreForm.java**, double click on DELETE button to generate its listener:

```
    private void jbDeleteActionPerformed(java.awt.event.ActionEvent evt) {
        Store_Utils.delete_handler(this);
    }
```

Step 3	Run the project. Choose **store_id** using **jcbStoreID** combobox. Then, Cl DELETE button. The corresponding row of data had been deleted database.

PLOTTING CHART
PLOTTING CHART

Step 1	Create a new **JFrame** and save it as **Charts_Store.java**.
Step 2	In **Charts_Store.java**, add three **JPanels** and set their corresponding **Variable Name** as **jPanel1**, **jPanel2**, and **jPanel3**. Then, add getter method for each object as follows:

```
    //Getter method for jPanel1
    public JPanel getJPanel1(){
        return this.jPanel1;
    }

    //Getter method for jPanel2
    public JPanel getJPanel2(){
        return this.jPanel2;
    }

    //Getter method for jPanel3
```

```
    public JPanel getJPanel3(){
        return this.jPanel3;
    }
```

Step 3

In **Store_Utils** class, define three new methods. These are three methods that draw different types of charts based on store distribution data.

The first method, **draw_pie_chart_store_by_district()**, draws a pie chart that shows the top 10 store distributions by district. It creates a **DefaultPieDataset** using the **create_pie_dataset()** method from the **Query_Store** class, passing in the SQL query string that retrieves the data, as well as the column names for the "Number" and "district" values. It then calls the **draw_piechart_with_dataset()** method to draw the pie chart on the specified **JPanel**.

The second method, **draw_pie_chart_store_by_country()**, is similar to the first method, but draws a pie chart that shows the top 10 store distributions by country. It also creates a **DefaultPieDataset**, but uses a different SQL query string that retrieves data on store distributions by country, and passes in the column names for the "Number" and "country" values.

The third method, **draw_bar_chart_store_by_city()**, draws a bar chart that shows the top 10 store distributions by city. It creates a **DefaultCategoryDataset** using the **create_bar_dataset()** method from the **Query_Store** class, passing in the SQL query string that retrieves the data, as well as the column names for the "Number" and "city" values. It then calls the **draw_barchart_with_dataset()** method to draw the bar chart on the specified JPanel.

```
    private static void
draw_pie_chart_store_by_district(Charts_Store frm, JPanel jp)
{
        jp.setPreferredSize(new Dimension(jp.getWidth(),
jp.getHeight()));
        DefaultPieDataset dataset =
create_pie_dataset(Query_Store.get_sql_store_district_dist(),
"Number", "district");

        //Draws piechart store distribution by district
        draw_piechart_with_dataset(frm, jp, dataset, "TOP 10
STORE DISTRIBUTION BY DISTRICT");
    }
```

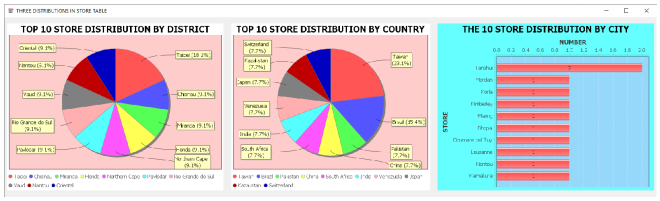
```
    private static void
draw_pie_chart_store_by_country(Charts_Store frm, JPanel jp){
        jp.setPreferredSize(new Dimension(jp.getWidth(),
jp.getHeight()));
        DefaultPieDataset dataset =
create_pie_dataset(Query_Store.get_sql_store_country_dist(),
"Number", "country");

        //Draws piechart store distribution by country
        draw_piechart_with_dataset(frm, jp, dataset, "TOP 10
STORE DISTRIBUTION BY COUNTRY");
    }

    private static void
draw_bar_chart_store_by_city(Charts_Store frm, JPanel jp){
        jp.setPreferredSize(new Dimension(jp.getWidth(),
jp.getHeight()));

        DefaultCategoryDataset dataset =
create_bar_dataset(Query_Store.get_sql_store_city_dist(),
"Number", "city");

        //Draws barchart store distribution by city
        draw_barchart_with_dataset(frm, jp, dataset, "THE 10
STORE DISTRIBUTION BY CITY", "STORE", "NUMBER");
    }
```

Step 4

In **Store_Utils** class, define a new method named **jbchart_handler()**.

```
    public static void
jbchart_handler(Charts_Store frm){
        //Draws piechart store
distribution by district

draw_pie_chart_store_by_district(frm,
frm.getJPanel1());

        //Draws piechart store
distribution by country

draw_pie_chart_store_by_country(frm,
frm.getJPanel2());

        //Draws barchart store
distribution by city

draw_bar_chart_store_by_city(frm,
frm.getJPanel3());
    }
```

Step 5	In **StoreForm**, double click on **jbChart** button to define its event listener:
	
	`private void jbChartActionPerformed(java.awt.event.ActionEvent evt) {`
	`  Charts_Store frm1 = new Charts_Store();`
	`  frm1.setLocationRelativeTo(null);`
	`  frm1.setTitle("THREE DISTRIBUTIONS IN STORE TABLE");`
	`  frm1.setVisible(true);`
	`  Store_Utils.jbchart_handler(frm1);`
	`  }`

Step 6	Run the project. Click on CHART button on the form. You will see the three charts displayed on the panels as shown in Figure 11.13.  **Figure 11.13** The top 10 address distribution by district, the top 10 address distribution by country, and the top 10 address distribution by city

This is the full version of **Store_Utils.java**:

```
package sakila;
import java.awt.Dimension;
import java.util.logging.Level;
import java.util.logging.Logger;
import java.sql.*;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashMap;
import java.util.Objects;
import javax.swing.JComboBox;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.event.TableModelEvent;
import javax.swing.table.DefaultTableModel;
import javax.swing.table.TableModel;
```

```java
import org.jfree.data.category.DefaultCategoryDataset;
import org.jfree.data.general.DefaultPieDataset;
import static sakila.Utility.create_bar_dataset;
import static sakila.Utility.draw_barchart_with_dataset;

public class Store_Utils extends Utility{
    public static final int FIRST_INDEX = 0;
    public static final int INVALID_INDEX = -1;

    private static int currentIndex = FIRST_INDEX;
    private static final String SQL_ID = Query_Store.get_sql_id();

    //Creates store table
    public static void create_store_table() {
        try (Connection conn = getConnection()) {
            Statement stmt = conn.createStatement();
            stmt.addBatch(Query_Store.get_sql_store());
            stmt.executeBatch();

            String message = String.format("Successfully creates store table'
            JOptionPane.showMessageDialog(null, message,
            "INFORMATION",JOptionPane.INFORMATION_MESSAGE);

        } catch (SQLException ex) {
            JOptionPane.showMessageDialog(null, ex.getMessage(),
                "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    //Populates store table with some rows of data
    public static void populate_store_table(){
        try(Connection conn = getConnection()){
            String sql = """
                INSERT INTO store(store_id, manager_staff_id, address_id,
last_update)
                VALUES(?, ?, ?, ?)""";

            //Creates a new Store class with default constructor
            PreparedStatement ps1 = conn.prepareStatement(sql);
            Store obj1 = new Store();
            ps1.setInt(1,obj1.getStoreID());
            ps1.setInt(2,obj1.getManagerID());
            ps1.setInt(3,obj1.getAddressID());
            ps1.setTimestamp(4,obj1.getLastUpdate());

            // Creates a new Store class with four-params constructor
            PreparedStatement ps2 = conn.prepareStatement(sql);
            Store obj2 = new Store(2, 2, 2, new
Timestamp(System.currentTimeMillis()));
            ps2.setInt(1,obj2.getStoreID());
            ps2.setInt(2,obj2.getManagerID());
            ps2.setInt(3,obj2.getAddressID());
            ps2.setTimestamp(4,obj2.getLastUpdate());
```

```
            ps1.executeUpdate();
            ps2.executeUpdate();

        }catch(SQLException ex){
            JOptionPane.showMessageDialog(null, ex.getMessage(),
                "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    //Reads the content of joined store, address, city and country tables
    public static void read_store_table(){
        try(Connection conn = getConnection()){
            Statement stmt = conn.createStatement();
            ResultSet rs =
stmt.executeQuery(Query_Store.get_sql_store_joint());

            while(rs.next()){
                int store_id = rs.getInt("store_id");
                int man_id = rs.getInt("manager_staff_id");
                int add_id = rs.getInt("address_id");
                String add = rs.getString("address");
                String add2 = rs.getString("address2");
                String dist = rs.getString("district");
                int city_id = rs.getInt("city_id");
                String post = rs.getString("postal_code");
                String ph = rs.getString("phone");
                String city = rs.getString("city");
                String country = rs.getString("country");
                Timestamp lu = rs.getTimestamp("last_update");

                //Creates a Store object using twelve-params constructor
                Store obj = new Store(store_id, man_id, add_id, add, add2,
dist, city_id, post, ph, city, country, lu);
                System.out.println(obj);
            }
            rs.close();
            stmt.close();

        }catch(SQLException ex){
            JOptionPane.showMessageDialog(null, ex.getMessage(),
                "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    private static ArrayList<Store> get_store_list(StoreForm frm, String sql,
String item){
        ArrayList<Store> list = new ArrayList<>();

        try(Connection conn = getConnection();
            PreparedStatement ps = conn.prepareStatement(sql)){
            if (item.equalsIgnoreCase("none")==false) {
                 ps.setString(1,item);
```

```
            }
            ResultSet rs = ps.executeQuery();

            Store obj;
            while(rs.next()){
                //Using twelve-params constructor
                obj = new Store(rs.getInt("store_id"),
                        rs.getInt("manager_staff_id"),
                        rs.getInt("address_id"),
                        rs.getString("address"),
                        rs.getString("address2"),
                        rs.getString("district"),
                        rs.getInt("city_id"),
                        rs.getString("postal_code"),
                        rs.getString("phone"),
                        rs.getString("city"),
                        rs.getString("country"),
                        rs.getTimestamp("last_update"));

                list.add(obj);
            }
        }catch (SQLException ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
                "ERROR",JOptionPane.ERROR_MESSAGE);
        }
        return list;
    }

    private static void show_table_store(StoreForm frm, ArrayList<Store> list
throws SQLException{
        DefaultTableModel model = new DefaultTableModel(0,0);

        String header[] = {"Store ID", "Manager Staff ID", "Address ID",
"Address", "Address 2", "District",
            "City ID", "Postal Code", "Phone", "City", "Country", "Last
Update"};

        model.setColumnIdentifiers(set_column_header(frm.getJTStore(),
header));
        frm.getJTStore().setModel(model);

        Object[] row = new Object[12];

        for(int i=0; i<list.size(); i++){
            row[0] = list.get(i).getStoreID();
            row[1] = list.get(i).getManagerID();
            row[2] = list.get(i).getAddressID();
            row[3] = list.get(i).getAddress();
            row[4] = list.get(i).getAddress2();
            row[5] = list.get(i).getDistrict();
            row[6] = list.get(i).getCityID();
            row[7] = list.get(i).getPostalCode();
            row[8] = list.get(i).getPhone();
            row[9] = list.get(i).getCity();
```

```
            row[10] = list.get(i).getCountry();
            row[11] = list.get(i).getLastUpdate();

            model.addRow(row);
        }
    }

    public static void refresh_controls(StoreForm frm){
        frm.setLocationRelativeTo(null);
        frm.setTitle("STORE FORM");

        //Shows the content of address table and populates combobox
        try{
            //Makes alternating color for table rows
            table_renderer(frm.getJTStore());

            //Populates table
            ArrayList<Store> list = get_store_list(frm,
Query_Store.get_sql_store_joint() + " ORDER BY store_id", "none");
            show_table_store(frm, list);

            //Populates jcbStoreID
            String sql_store_id = "SELECT store_id FROM store ORDER BY
store_id";
            populate_combobox(sql_store_id, frm.getJCBStoreID(), frm);

            //Populates jcbManStaffID
            String sql_staff_id = "SELECT manager_staff_id FROM store ORDER B
manager_staff_id";
            populate_combobox(sql_staff_id, frm.getJCBManStaffID(), frm);

            //Populates jcbAddressID
            String sql_add_id = "SELECT address_id FROM address ORDER BY
address_id";
            populate_combobox(sql_add_id, frm.getJCBAddressID(), frm);

        }catch (SQLException ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
            "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    private static void clear_address_controls(StoreForm frm){
        frm.getJTFAddressID().setText("");
        frm.getJTFAddress().setText("");
        frm.getJTFAddress2().setText("");
        frm.getJTFDistrict().setText("");
        frm.getJTFPhone().setText("");
        frm.getJTFPostalCode().setText("");
        frm.getJTFCity().setText("");
        frm.getJTFCountry().setText("");
    }
```

```
   //Displays address data result row by row
    private static <T> void display_address_data(StoreForm frm, String sql, T
item){
        try(Connection conn = getConnection()){
            PreparedStatement ps = conn.prepareStatement(sql);
            ps.setObject(1,item);
            ResultSet rs = ps.executeQuery();

            if (!rs.next()) {
                // no row found, clear the form fields
                clear_address_controls(frm);
                return;
            }

            do{

frm.getJTFAddressID().setText(String.valueOf(rs.getInt("address_id")));
                frm.getJTFAddress().setText(rs.getString("address"));
                frm.getJTFAddress2().setText(rs.getString("address2"));
                frm.getJTFDistrict().setText(rs.getString("district"));
                frm.getJTFPhone().setText(rs.getString("phone"));
                frm.getJTFPostalCode().setText(rs.getString("postal_code"));
                frm.getJTFCity().setText(rs.getString("city"));
                frm.getJTFCountry().setText(rs.getString("country"));

                // Determines item selected from jcbAddressID
                find_combo_value_selected(frm.getJCBAddressID(),
rs.getInt("address_id"));

            }while(rs.next());

            rs.close();
            ps.close();
        }catch(SQLException ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    public static void jcbAddress_handler(StoreForm frm) {
        Object item = frm.getJCBAddressID().getSelectedItem();
        display_address_data(frm, Query_Address.get_sql_address_joint() + "
WHERE address_id = ?", item);
    }

    public static void jcbManStaff_handler(StoreForm frm) {
        Object item = frm.getJCBManStaffID().getSelectedItem();
        frm.getJTFManStaffID().setText(String.valueOf(item));
    }

   //Displays store data result row by row
    private static <T> void display_store_data(StoreForm frm, String sql, T
item){
```

```
        try(Connection conn = getConnection()){
            PreparedStatement ps = conn.prepareStatement(sql);
            ps.setObject(1,item);
            ResultSet rs = ps.executeQuery();

            if (!rs.next()) {
                // no row found, clear the form fields
                frm.getJCBManStaffID().setSelectedIndex(-1);
                frm.getJCBAddressID().setSelectedIndex(-1);
                frm.getJTFLastUpdate().setText("");
                return;
            }

            do{

frm.getJTFStoreID().setText(String.valueOf(rs.getInt("store_id")));

frm.getJTFLastUpdate().setText(String.valueOf(rs.getTimestamp("last_update"));

                // Determines item selected from jcbStoreID
                find_combo_value_selected(frm.getJCBStoreID(),
rs.getInt("store_id"));

                // Determines item selected from jcbManStaffID
                find_combo_value_selected(frm.getJCBManStaffID(),
rs.getInt("manager_staff_id"));

                // Determines item selected from jcbAddressID
                find_combo_value_selected(frm.getJCBAddressID(),
rs.getInt("address_id"));

            }while(rs.next());

            rs.close();
            ps.close();
        }catch(SQLException ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    public static void jcbStore_handler(StoreForm frm) {
        Object item = frm.getJCBStoreID().getSelectedItem();
        display_store_data(frm, SQL_ID, item);
    }

    public static void show_first_row(StoreForm frm){
        String item =
String.valueOf(frm.getJCBStoreID().getItemAt(FIRST_INDEX));
        display_store_data(frm, SQL_ID, item);
        currentIndex = FIRST_INDEX;
    }
```

```
    public static void show_last_row(StoreForm frm){
        int endIndex = frm.getJCBStoreID().getItemCount() - 1;
        String item = String.valueOf(frm.getJCBStoreID().getItemAt(endIndex));
        display_store_data(frm, SQL_ID, item);
        currentIndex = endIndex;
    }

    public static void show_prev_row(StoreForm frm){
        currentIndex--;
        if(currentIndex < FIRST_INDEX){
            currentIndex = FIRST_INDEX;
            return;
        }
        String item =
String.valueOf(frm.getJCBStoreID().getItemAt(currentIndex));
        display_store_data(frm, SQL_ID, item);
    }

    public static void show_next_row(StoreForm frm){
        int endIndex = frm.getJCBStoreID().getItemCount() - 1;
        currentIndex++;
        if(currentIndex > endIndex){
            currentIndex = endIndex;
            return;
        }
        String item =
String.valueOf(frm.getJCBStoreID().getItemAt(currentIndex));
        display_store_data(frm, SQL_ID, item);
    }

    public static void mouse_pressed_handler(StoreForm frm) {
        Objects.requireNonNull(frm, "frm must not be null");

        int selectedIndex = frm.getJTStore().getSelectedRow();
        if (selectedIndex == -1) {
            JOptionPane.showMessageDialog(frm, "Please select a row to view 
data.",
                "No row selected", JOptionPane.INFORMATION_MESSAGE);
            return;
        }

        try (Connection conn = getConnection()) {
            String id =
String.valueOf(frm.getJTStore().getModel().getValueAt(selectedIndex, 0));

            // Displays store data
            display_store_data(frm, SQL_ID, id);

        } catch (SQLException ex) {
            Logger.getLogger(StoreForm.class.getName()).log(Level.SEVERE,
"Error displaying store data", ex);
            String message = "Error displaying store data: " + ex.getMessage
            String stackTrace = Arrays.toString(ex.getStackTrace());
```

```
            JOptionPane.showMessageDialog(frm, message + "\n\n" + stackTrace,
"ERROR", JOptionPane.ERROR_MESSAGE);
        }
    }


    //Updates row of data in store tabel by store_id
    public static void update_row_by_store_id(StoreForm frm, int store_id, in
man_id, int add_id) throws SQLException{
        Connection conn = getConnection();
        ResultSet rs = null;
        String query_id = "SELECT store_id FROM store WHERE store_id = ?";
        String update_query = """
            UPDATE store SET manager_staff_id = ?, address_id = ? WHERE
store_id = ?""";
        try(PreparedStatement idPs = conn.prepareStatement(query_id,

ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_READ_ONLY);
        PreparedStatement updatePS = conn.prepareStatement(update_query,
                ResultSet.TYPE_SCROLL_SENSITIVE, ResultSet.CONCUR_UPDATABLE))
        {
            idPs.setInt(1,store_id);
            if(!idPs.execute()){
                String message = "Can't find store_id " + store_id;

                JOptionPane.showMessageDialog(frm, message,
                    "ERROR",JOptionPane.ERROR_MESSAGE);
            } else{
                rs = idPs.getResultSet();
                rs.next();

                //Creates a Store object using four-params constructor
                Store obj = new Store(store_id, man_id, add_id, new
Timestamp(System.currentTimeMillis()));
                updatePS.setInt(1, obj.getManagerID());
                updatePS.setInt(2, obj.getAddressID());
                updatePS.setInt(3, obj.getStoreID());

                updatePS.executeUpdate();
                rs.close();
                updatePS.close();
                idPs.close();
                conn.close();
            }
        }catch(SQLException ex){
            Logger.getLogger(StoreForm.class.getName()).log(Level.SEVERE,
"Error updating store data", ex);
            String message = "Error updating store data: " + ex.getMessage();
            String stackTrace = Arrays.toString(ex.getStackTrace());
            JOptionPane.showMessageDialog(null, message + "\n\n" + stackTrace
"ERROR", JOptionPane.ERROR_MESSAGE);
        }catch(java.lang.NumberFormatException ex){
            Logger.getLogger(StoreForm.class.getName()).log(Level.SEVERE,
"Invalid Input", ex);
```

```
            String message = "Invalid Input: " + ex.getMessage();
            String stackTrace = Arrays.toString(ex.getStackTrace());
            JOptionPane.showMessageDialog(null, message + "\n\n" + stackTrace
"ERROR", JOptionPane.ERROR_MESSAGE);
        }
    }

    private static HashMap<String, String> read_inputs(StoreForm frm) {
        HashMap<String, String> input_data = new HashMap<>();
        String store_id =
String.valueOf(frm.getJCBStoreID().getSelectedItem());
        String man_id = frm.getJTFManStaffID().getText();
        String add_id =
String.valueOf(frm.getJCBAddressID().getSelectedItem());

        // Validate user input
        int add_id_int = 0;
        try {
            add_id_int = Integer.parseInt(add_id);
            if (add_id_int <= 0) {
                throw new IllegalArgumentException("Address ID cannot be
negative or zero");
            }
        } catch (NumberFormatException ex) {
            JOptionPane.showMessageDialog(frm, "Invalid Address ID: " + add_
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        } catch (IllegalArgumentException ex) {
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        }

        int store_id_int = 0;
        try {
            store_id_int = Integer.parseInt(store_id);
            if (store_id_int <= 0) {
                throw new IllegalArgumentException("Store ID cannot be negat
or zero");
            }
        } catch (NumberFormatException ex) {
            JOptionPane.showMessageDialog(frm, "Invalid Store ID: " + store_
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        } catch (IllegalArgumentException ex) {
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        }

        int man_id_int = 0;
        try {
            man_id_int = Integer.parseInt(man_id);
```

```
            if (man_id_int <= 0) {
                throw new IllegalArgumentException("Manager Staff ID cannot b
negative or zero");
            }
        } catch (NumberFormatException ex) {
            JOptionPane.showMessageDialog(frm, "Invalid Manager Staff ID: " +
man_id,
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        } catch (IllegalArgumentException ex) {
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        }

        input_data.put("add_id", add_id);
        input_data.put("store_id", store_id);
        input_data.put("man_id", man_id);

        return input_data;
    }

    private static void edit_actual(StoreForm frm){
        try{
            HashMap<String, String> input_data = read_inputs(frm);
            int add_id = Integer.parseInt(input_data.get("add_id"));
            int store_id = Integer.parseInt(input_data.get("store_id"));
            int man_id = Integer.parseInt(input_data.get("man_id"));

            update_row_by_store_id(frm, store_id, man_id, add_id);

            //Refreshes all objects on form
            refresh_controls(frm);

        }catch(SQLException ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    private static void enable_controls(boolean state, StoreForm frm){
        frm.getJBFirst().setEnabled(state);
        frm.getJBPrev().setEnabled(state);
        frm.getJBNext().setEnabled(state);
        frm.getJBLast().setEnabled(state);
        frm.getJBInsert().setEnabled(state);
        frm.getJBDelete().setEnabled(state);
        frm.getJTFAddressID().setEnabled(state);
        frm.getJTFStoreID().setEnabled(state);
        frm.getJTFAddress().setEnabled(state);
        frm.getJTFAddress2().setEnabled(state);
        frm.getJTFDistrict().setEnabled(state);
        frm.getJTFPhone().setEnabled(state);
```

```
        frm.getJTFPostalCode().setEnabled(state);
        frm.getJTFCity().setEnabled(state);
        frm.getJTFCountry().setEnabled(state);
    }

    public static void edit_handler(StoreForm frm){
        if(frm.getJBEdit().getText().equals("EDIT")){
            frm.getJBEdit().setText("CONFIRM");

            // Disables controls
            enable_controls(false, frm);
        }

        else {
            frm.getJBEdit().setText("EDIT");

            // Actual editing
            edit_actual(frm);

            //Enables controls
            enable_controls(true, frm);
        }
    }

    public static void edit_database_from_jtable(TableModelEvent e, StoreForm
frm){
        if (e.getType() == TableModelEvent.UPDATE) {
            int row = e.getFirstRow();
            TableModel model = (TableModel)e.getSource();
            int store_id =
Integer.parseInt(String.valueOf(model.getValueAt(row, 0)));
            int add_id = Integer.parseInt(String.valueOf(model.getValueAt(row
2)));
            int man_id = Integer.parseInt(String.valueOf(model.getValueAt(row
1)));

            try{
                update_row_by_store_id(frm, store_id, man_id, add_id);

                //Refreshes all objects on form
                refresh_controls(frm);

            } catch (SQLIntegrityConstraintViolationException ex) {
                Logger.getLogger(StoreForm.class.getName()).log(Level.SEVERE,
"Duplicate entry", ex);
                JOptionPane.showMessageDialog(frm, "Error: Duplicate entry\n"
ex.getMessage());
            } catch (SQLSyntaxErrorException ex) {
                Logger.getLogger(StoreForm.class.getName()).log(Level.SEVERE,
"Invalid SQL syntax", ex);
                JOptionPane.showMessageDialog(frm, "Error: Invalid SQL
syntax\n" + ex.getMessage());
            } catch (SQLException ex) {
```

```
                Logger.getLogger(StoreForm.class.getName()).log(Level.SEVERE,
"Database error", ex);
                JOptionPane.showMessageDialog(frm, "Error: Database error\n"
ex.getMessage());
            }
        }
    }

    //Inserts new row into store table
    private static void insert_row(StoreForm frm) throws SQLException{
        HashMap<String, String> input_data = read_inputs(frm);
        int add_id = Integer.parseInt(input_data.get("add_id"));
        int man_id = Integer.parseInt(input_data.get("man_id"));

        // SQL insert statement
        String sql = """
            INSERT INTO store(manager_staff_id, address_id) VALUES(?, ?)""";

        try(Connection conn = getConnection();
         PreparedStatement pstmt = conn.prepareStatement(sql)){

            //Creates a Store object three-params constructor
            Store obj = new Store(man_id, add_id, new
Timestamp(System.currentTimeMillis()));
            pstmt.setInt(1,obj.getManagerID());
            pstmt.setInt(2,obj.getAddressID());

            //Executes the sql insert statement
            pstmt.executeUpdate();
        } catch (SQLException ex) {
            Logger.getLogger(StoreForm.class.getName()).log(Level.SEVERE,
"Database error", ex);
            JOptionPane.showMessageDialog(frm, "Error: Database error\n" +
ex.getMessage());
        }
    }

    private static void insert_actual(StoreForm frm){
        try{
            insert_row(frm);

            //Refreshes table and comboboxes
            refresh_controls(frm);

        }catch(SQLException ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    public static void insert_handler(StoreForm frm){
        if(frm.getJBInsert().getText().equals("INSERT")){
            frm.getJBInsert().setText("CONFIRM");
```

```
            //Disables jbEdit
            frm.getJBEdit().setEnabled(false);

            // Disables controls
            enable_controls(false, frm);
            frm.getJCBStoreID().setEnabled(false);

            // Clears controls
            clear_address_controls(frm);

            // Enables
            frm.getJBInsert().setEnabled(true);
        }

        else {
            frm.getJBInsert().setText("INSERT");

            // Actual insertion
            insert_actual(frm);

            //Enables jbEdit
            frm.getJBEdit().setEnabled(true);

            //Enables controls
            enable_controls(true, frm);
            frm.getJCBStoreID().setEnabled(true);
        }
    }

    public static void delete_handler(StoreForm frm){
        int dialogButton = JOptionPane.YES_NO_OPTION;
        int store_id =
Integer.parseInt(String.valueOf(frm.getJCBStoreID().getSelectedItem()));

        String message = String.format("Are you sure you want to delete the r
Store ID: %d)", store_id);
        int answer = JOptionPane.showConfirmDialog(frm, message, "DELETING RO
OF DATA", dialogButton);

        if(answer == JOptionPane.YES_OPTION){
            String query = """
                DELETE FROM store WHERE store_id = ?""";
            try(Connection conn = getConnection();
                PreparedStatement ps = conn.prepareStatement(query)){
                // Use PreparedStatement to avoid SQL injection attacks
                ps.setInt(1, store_id);
                ps.executeUpdate();

                // Refresh table and comboboxes
                refresh_controls(frm);

            } catch (SQLException ex){
```

```
                JOptionPane.showMessageDialog(frm, ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
            }
        }
    }

    private static void draw_pie_chart_store_by_district(Charts_Store frm,
JPanel jp){
        jp.setPreferredSize(new Dimension(jp.getWidth(), jp.getHeight()));
        DefaultPieDataset dataset =
create_pie_dataset(Query_Store.get_sql_store_district_dist(), "Number",
"district");

        //Draws piechart store distribution by district
        draw_piechart_with_dataset(frm, jp, dataset, "TOP 10 STORE DISTRIBUTI
BY DISTRICT");
    }

    private static void draw_pie_chart_store_by_country(Charts_Store frm,
JPanel jp){
        jp.setPreferredSize(new Dimension(jp.getWidth(), jp.getHeight()));
        DefaultPieDataset dataset =
create_pie_dataset(Query_Store.get_sql_store_country_dist(), "Number",
"country");

        //Draws piechart store distribution by country
        draw_piechart_with_dataset(frm, jp, dataset, "TOP 10 STORE DISTRIBUTI
BY COUNTRY");
    }

    private static void draw_bar_chart_store_by_city(Charts_Store frm, JPanel
jp){
        jp.setPreferredSize(new Dimension(jp.getWidth(), jp.getHeight()));

        DefaultCategoryDataset dataset =
create_bar_dataset(Query_Store.get_sql_store_city_dist(), "Number", "city");

        //Draws barchart store distribution by city
        draw_barchart_with_dataset(frm, jp, dataset, "THE 10 STORE DISTRIBUTI
BY CITY", "STORE", "NUMBER");
    }

    public static void jbchart_handler(Charts_Store frm){
        //Draws piechart store distribution by district
        draw_pie_chart_store_by_district(frm, frm.getJPanel1());

        //Draws piechart store distribution by country
        draw_pie_chart_store_by_country(frm, frm.getJPanel2());

        //Draws barchart store distribution by city
        draw_bar_chart_store_by_city(frm, frm.getJPanel3());
    }
}
```

This is the full version of **StoreForm.java**:

```
package sakila;

import java.awt.Toolkit;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JComboBox;
import javax.swing.JMenuItem;
import javax.swing.JOptionPane;
import javax.swing.JPopupMenu;
import javax.swing.JTable;
import javax.swing.JTextField;
import javax.swing.event.TableModelEvent;
import javax.swing.event.TableModelListener;

public class StoreForm extends javax.swing.JFrame {
    public StoreForm() {
        initComponents();
        Utility.setLookAndFeel(this);
        Store_Utils.refresh_controls(this);

 this.setIconImage(Toolkit.getDefaultToolkit().getImage(getClass().getResourc
;
//        this.setDefaultCloseOperation(this.HIDE_ON_CLOSE);
    }

    //Getter method for jtfStoreID
    public JTextField getJTFStoreID(){
        return this.jtfStoreID;
    }

    //Getter method for jcbStoreID
    public JComboBox getJCBStoreID(){
        return this.jcbStoreID;
    }

    //Getter method for jtfManStaffID
    public JTextField getJTFManStaffID(){
        return this.jtfManStaffID;
    }

    //Getter method for jcbManStaffID
    public JComboBox getJCBManStaffID(){
        return this.jcbManStaffID;
    }

    //Getter method for jtfAddressID
    public JTextField getJTFAddressID(){
```

```
        return this.jtfAddressID;
    }

    //Getter method for jcbAddressID
    public JComboBox getJCBAddressID(){
        return this.jcbAddressID;
    }

    //Getter method for jtfDistrict
    public JTextField getJTFDistrict(){
        return this.jtfDistrict;
    }

    //Getter method for jtfCity
```

```
    public JTextField getJTFCity(){
        return this.jtfCity;
    }

    //Getter method for jtfCountry
    public JTextField getJTFCountry(){
        return this.jtfCountry;
    }

    //Getter method for jtfPostalCode
    public JTextField getJTFPostalCode(){
        return this.jtfPostalCode;
    }

    //Getter method for jtfPhone
    public JTextField getJTFPhone(){
        return this.jtfPhone;
    }

    //Getter method for jtfAddress
    public JTextField getJTFAddress(){
        return this.jtfAddress;
    }

    //Getter method for jtfAddress2
    public JTextField getJTFAddress2(){
        return this.jtfAddress2;
    }

    //Getter method for jtfLastUpdate
    public JTextField getJTFLastUpdate(){
        return this.jtfLastUpdate;
    }

    //Getter method for jtAddress
    public JTable getJTStore(){
        return this.jtStore;
    }

    //Getter method for jbEdit
    public JButton getJBEdit(){
        return this.jbEdit;
    }

    //Getter method for jbInsert
    public JButton getJBInsert(){
        return this.jbInsert;
    }

    //Getter method for jbDelete
    public JButton getJBDelete(){
        return this.jbDelete;
```

```
    }

    //Getter method for jbChart
    public JButton getJBChart(){
        return this.jbChart;
    }

    //Getter method for jbFirst
    public JButton getJBFirst(){
        return this.jbFirst;
    }

    //Getter method for jbPrev
    public JButton getJBPrev(){
        return this.jbPrev;
    }

    //Getter method for jbNext
    public JButton getJBNext(){
        return this.jbNext;
    }

    //Getter method for jbLast
    public JButton getJBLast(){
        return this.jbLast;
    }

    //Getter method for jbAddressForm
    public JButton getJBAddressForm(){
        return this.jbAddressForm;
    }

    //Getter method for jbCountryForm
    public JButton getJBCountryForm(){
        return this.jbCountryForm;
    }

    //Getter method for jbCityForm
    public JButton getJBCityForm(){
        return this.jbCityForm;
    }

    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">
    private void initComponents() {
        //...
        pack();
    }// </editor-fold>

    private void jtStoreMousePressed(java.awt.event.MouseEvent evt) {
        Store_Utils.mouse_pressed_handler(this);
    }
```

```
    private void jbFirstActionPerformed(java.awt.event.ActionEvent evt) {
        Store_Utils.show_first_row(this);
    }

    private void jbPrevActionPerformed(java.awt.event.ActionEvent evt) {
        Store_Utils.show_prev_row(this);
    }

    private void jbNextActionPerformed(java.awt.event.ActionEvent evt) {
        Store_Utils.show_next_row(this);
    }

    private void jbLastActionPerformed(java.awt.event.ActionEvent evt) {
        Store_Utils.show_last_row(this);
    }

    private void jbCountryFormActionPerformed(java.awt.event.ActionEvent evt)
        CountryForm ct_form = new CountryForm();
        ct_form.setVisible(true);
    }

    private void jbCityFormActionPerformed(java.awt.event.ActionEvent evt) {
        CityForm cty_form = new CityForm();
        cty_form.setVisible(true);
    }

    private void jbDeleteActionPerformed(java.awt.event.ActionEvent evt) {
        Store_Utils.delete_handler(this);
    }

    private void jbInsertActionPerformed(java.awt.event.ActionEvent evt) {
        Store_Utils.insert_handler(this);
    }

    private void jbEditActionPerformed(java.awt.event.ActionEvent evt) {
        Store_Utils.edit_handler(this);
    }

    private void jcbAddressIDActionPerformed(java.awt.event.ActionEvent evt)
        Store_Utils.jcbAddress_handler(this);
    }

    private void jbAddressFormActionPerformed(java.awt.event.ActionEvent evt)
        // TODO add your handling code here:
    }

    private void jcbStoreIDActionPerformed(java.awt.event.ActionEvent evt) {
        Store_Utils.jcbStore_handler(this);
    }

    private void jcbManStaffIDActionPerformed(java.awt.event.ActionEvent evt)
```

```
        Store_Utils.jcbManStaff_handler(this);
    }

    private void jbChartActionPerformed(java.awt.event.ActionEvent evt) {
        Charts_Store frm1 = new Charts_Store();
        frm1.setLocationRelativeTo(null);
        frm1.setTitle("THREE DISTRIBUTIONS IN STORE TABLE");
        frm1.setVisible(true);
        Store_Utils.jbchart_handler(frm1);
    }

    private void jtStoreMouseClicked(java.awt.event.MouseEvent evt) {
        // instantiate StoreTableModelListener and add it as a listener to th
        StoreTableModelListener tableModelListener = new
StoreTableModelListener(this.getJTStore(), this);
        this.getJTStore().getModel().addTableModelListener(tableModelListener

    }

    public static void main(String args[]) {
        try {
            for (javax.swing.UIManager.LookAndFeelInfo info :
javax.swing.UIManager.getInstalledLookAndFeels()) {
                if ("Nimbus".equals(info.getName())) {
                    javax.swing.UIManager.setLookAndFeel(info.getClassName())
                    break;
                }
            }
        } catch (ClassNotFoundException ex) {

java.util.logging.Logger.getLogger(StoreForm.class.getName()).log(java.util.l
null, ex);
        } catch (InstantiationException ex) {

java.util.logging.Logger.getLogger(StoreForm.class.getName()).log(java.util.l
null, ex);
        } catch (IllegalAccessException ex) {

java.util.logging.Logger.getLogger(StoreForm.class.getName()).log(java.util.l
null, ex);
        } catch (javax.swing.UnsupportedLookAndFeelException ex) {

java.util.logging.Logger.getLogger(StoreForm.class.getName()).log(java.util.l
null, ex);
        }
        //</editor-fold>
        //</editor-fold>

        /* Create and display the form */
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new StoreForm().setVisible(true);
            }
        });
```

```
    }

    // Variables declaration - do not modify
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel10;
    private javax.swing.JLabel jLabel11;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JLabel jLabel3;
    private javax.swing.JLabel jLabel4;
    private javax.swing.JLabel jLabel5;
    private javax.swing.JLabel jLabel6;
    private javax.swing.JLabel jLabel7;
    private javax.swing.JLabel jLabel8;
    private javax.swing.JLabel jLabel9;
    private javax.swing.JScrollPane jScrollPane1;
    private javax.swing.JButton jbAddressForm;
    private javax.swing.JButton jbChart;
    private javax.swing.JButton jbCityForm;
    private javax.swing.JButton jbCountryForm;
    private javax.swing.JButton jbDelete;
    private javax.swing.JButton jbEdit;
    private javax.swing.JButton jbFirst;
    private javax.swing.JButton jbInsert;
    private javax.swing.JButton jbLast;
    private javax.swing.JButton jbNext;
    private javax.swing.JButton jbPrev;
    private javax.swing.JComboBox<String> jcbAddressID;
    private javax.swing.JComboBox<String> jcbManStaffID;
    private javax.swing.JComboBox<String> jcbStoreID;
    private javax.swing.JTable jtStore;
    private javax.swing.JTextField jtfAddress;
    private javax.swing.JTextField jtfAddress2;
    private javax.swing.JTextField jtfAddressID;
    private javax.swing.JTextField jtfCity;
    private javax.swing.JTextField jtfCountry;
    private javax.swing.JTextField jtfDistrict;
    private javax.swing.JTextField jtfLastUpdate;
    private javax.swing.JTextField jtfManStaffID;
    private javax.swing.JTextField jtfPhone;
    private javax.swing.JTextField jtfPostalCode;
    private javax.swing.JTextField jtfStoreID;
    // End of variables declaration
}
```

INVENTORY FORM

INVENTORY FORM

CREATING AND POPULATING INVENTORY TABLE

CREATING AND POPULATING INVENTORY TABLE

Step 1	Create a new class named **Query_Inventory**. It contains a series of SQL queries for accessing data related to inventory in a database. The class has several private static final **String** variables that contain the SQL queries, including:

- **sql_min**: a query to get the minimum value of the **inventory_id** column from the **inventory** table.
- **sql_max**: a query to get the maximum value of the **inventory_id** column from the **inventory** table.
- **sql_id**: a query to get information about a specific inventory item based on its **inventory_id**.
- **sql_inventory_district_dist**: a query to get a list of the top 10 districts by the number of inventory items in each district.
- **sql_inventory_country_dist**: a query to get a list of the top 10 countries by the number of inventory items in each country.
- **sql_inventory_city_dist**: a query to get a list of the top 10 cities by the number of inventory items in each city.
- **sql_inventory_duration_dist**: a query to get a list of the number of inventory items for each rental duration value.
- **sql_inventory_lang_dist**: a query to get a list of the number of inventory items for each language.
- **sql_inventory_rating_dist**: a query to get a list of the number of inventory items for each rating value.
- **sql_inventory_year_dist**: a query to get a list of the top 10 years by the number of inventory items released in each year.
- **sql_inventory_joint**: a SELECT statement that joins several tables in the inventory database, including inventory, film, language, store, address, city, and country. It selects multiple columns from these tables, including the inventory_id, film_id, store_id, last_update, title, description, release_year, rental_duration, rental_rate, length, replacement_cost, rating, special_features, language_name, address_id, address, district, postal_code, phone, city, and country columns.
- **sql_inventory**: a CREATE TABLE statement that creates a table named inventory in the inventory database. The

table has several columns, including inventory_id, film_id, store_id, and last_update. It also defines some constraints and keys for the table, including foreign keys that reference the store and film tables. The table is created with the InnoDB engine and the utf8mb4 character set.

The class also includes several getter methods for accessing the SQL queries from other classes.

```
1       package sakila;
2
3       public class Query_Inventory {
4           private static final String
5       sql_min = "SELECT
6       MIN(inventory_id) FROM
7       inventory";
8           private static final String
9       sql_max = "SELECT
10      MAX(inventory_id) FROM
11      inventory";
12          private static final String
13      sql_id =
14      get_sql_inventory_joint() + "
15      WHERE i.inventory_id  = ?";
16
17          private static final String
18      sql_inventory_district_dist = """
19              SELECT ad.district,
20      COUNT(*) AS Number
21              FROM inventory i
22              JOIN store st ON
23      st.store_id = i.store_id
24              JOIN address ad ON
25      ad.address_id = st.address_id
26              GROUP BY ad.district
27              ORDER BY Count(*) DESC
28              LIMIT 10""";
29
30          private static final String
31      sql_inventory_country_dist = """
32              SELECT co.country,
33      COUNT(*) AS Number
34              FROM inventory i
35              JOIN store st ON
36      st.store_id = i.store_id
37              JOIN address ad ON
38      ad.address_id = st.address_id
39              JOIN city ci ON
40      ci.city_id = ad.city_id
```

```
41          JOIN country co ON
42      co.country_id = ci.country_id
43          GROUP BY co.country
44          ORDER BY Count(*) DESC
45          LIMIT 10""";
46
47          private static final String
48      sql_inventory_city_dist = """
49          SELECT ci.city, COUNT(*)
50      AS Number
51          FROM inventory i
52          JOIN store st ON
53      st.store_id = i.store_id
54          JOIN address ad ON
55      ad.address_id = st.address_id
56          JOIN city ci ON
57      ci.city_id = ad.city_id
58          GROUP BY ci.city
59          ORDER BY Count(*) DESC
60          LIMIT 10""";
61
62          private static final String
63      sql_inventory_duration_dist = """
64          SELECT f.rental_duration
65      AS duration, COUNT(*) AS Number
66          FROM inventory i
67          JOIN film f ON f.film_id
68      = i.film_id
69          GROUP BY duration
70          ORDER BY Count(*) DESC,
71      duration ASC;""";
72
73          private static final String
74      sql_inventory_lang_dist = """
75          SELECT l1.name, COUNT(*)
76      AS Number
77          FROM inventory i
78          JOIN film f ON f.film_id
79      = i.film_id
80          JOIN language l1 ON
81      l1.language_id = f.language_id
82          GROUP BY l1.name
83          ORDER BY Count(*) DESC,
84      l1.name ASC;""";
85
86          private static final String
87      sql_inventory_rating_dist = """
88          SELECT rating, COUNT(*)
89      AS Number
90          FROM inventory i
91          JOIN film f ON f.film_id
92      = i.film_id
93          GROUP BY rating
94
```

```
95          ORDER BY Count(*) DESC,
96      rating ASC""";
97
98          private static final String
99      sql_inventory_year_dist = """
100         SELECT YEAR(release_year)
101     AS year, Count(*) AS Number
102         FROM inventory i
103         JOIN film f ON f.film_id
104     = i.film_id
105         GROUP BY year
106         ORDER BY Count(*) DESC,
107     year ASC
108         LIMIT 10""";
109
110         private static final String
111     sql_inventory_joint = """
112         SELECT i.inventory_id,
113     i.film_id, i.store_id,
114     i.last_update,
115                 f.title,
116     f.description, f.release_year,
117                 f.rental_duration,
118                 f.rental_rate,
119     f.length, f.replacement_cost,
120     f.rating,
121                 f.special_features,
122     l1.name AS language_name,
123                 ad.address_id,
124     ad.address, ad.district,
125     ad.postal_code,
126127               ad.phone, ci.city,
128     co.country
129         FROM inventory i
130         JOIN film f ON f.film_id
131     = i.film_id
132         JOIN language l1 ON
133     l1.language_id = f.language_id
134         JOIN store st ON
135     st.store_id = i.store_id
136         JOIN address ad ON
137     ad.address_id = st.address_id
138         JOIN city ci ON
139     ci.city_id = ad.city_id
140         JOIN country co ON
141     co.country_id = ci.country_id""";
142
143         private static final String
144     sql_inventory = """
144         CREATE TABLE inventory (
145           inventory_id MEDIUMINT
146     UNSIGNED NOT NULL AUTO_INCREMENT,
147
148
```

```
149             film_id SMALLINT
150     UNSIGNED NOT NULL,
151             store_id TINYINT
        UNSIGNED NOT NULL,
                last_update TIMESTAMP
        NOT NULL DEFAULT
        CURRENT_TIMESTAMP ON UPDATE
        CURRENT_TIMESTAMP,
                PRIMARY KEY
        (inventory_id),
                KEY idx_fk_film_id
        (film_id),
                KEY
        idx_store_id_film_id
        (store_id,film_id),
                CONSTRAINT
        fk_inventory_store FOREIGN KEY
        (store_id) REFERENCES store
        (store_id) ON DELETE RESTRICT ON
        UPDATE CASCADE,
                CONSTRAINT
        fk_inventory_film FOREIGN KEY
        (film_id) REFERENCES film
        (film_id) ON DELETE RESTRICT ON
        UPDATE CASCADE
                ) ENGINE=InnoDB DEFAULT
        CHARSET=utf8mb4;""";


            //Getter methods
            public static String
        get_sql_min() {
                return sql_min;
            }

            public static String
        get_sql_max() {
                return sql_max;
            }

            public static String
        get_sql_id() {
                return sql_id;
            }

            public static String
        get_sql_inventory() {
                return sql_inventory;
            }

            public static String
        get_sql_inventory_joint() {
```

```
        return
sql_inventory_joint;
    }

    public static String
get_sql_inventory_year_dist() {
        return
sql_inventory_year_dist;
    }

    public static String
get_sql_inventory_rating_dist() {
        return
sql_inventory_rating_dist;
    }

    public static String
get_sql_inventory_lang_dist() {
        return
sql_inventory_lang_dist;
    }

    public static String
get_sql_inventory_duration_dist()
{
        return
sql_inventory_duration_dist;
    }

    public static String
get_sql_inventory_city_dist() {
        return
sql_inventory_city_dist;
    }

    public static String
get_sql_inventory_country_dist()
{
        return
sql_inventory_country_dist;
    }

    public static String
get_sql_inventory_district_dist()
{
        return
sql_inventory_district_dist;
    }
}
```

Step	Then, create a public class named **Inventory** with

2

instance variables and getter/setter methods. It has instance variables for inventory_id, film_id, store_id, last_update, address_id, address, district, postal_code, phone, city, country, title, description, release_year, rental_duration, rental_rate, length, replacement_cost, rating, special_features, and lang_name.

The class has four constructors: a default constructor, a three-params constructor, a four-params constructor, and a twenty-one-params constructor that takes all the instance variables as parameters.

The class also has getter and setter methods for all instance variables, as well as an overridden **hashCode()** method, an overridden **equals()** method, and an overridden **toString()** method. The **hashCode()** method returns a hash code based on the inventory_id, store_id, film_id, and last_update instance variables.

The **equals()** method checks if the object being compared is the same as the current object, if it has the same class, and if its instance variables are equal to the current object's instance variables. The **toString()** method returns a string representation of the Inventory object.

```
1   package sakila;
2   import java.util.Objects;
3   import java.sql.Timestamp;
4
5   public class Inventory {
6       //21 instance variables
7       private int inventory_id;
8       private int film_id;
9       private int store_id;
10      private Timestamp last_update;
11
12      private int address_id;
13      private String address;
14      private String district;
15      private String postal_code;
16      private String phone;
17      private String city;
18      private String country;
19
20      private String title;
```

```
21      private String description;
22      private int release_year;
23      private int rental_duration;
24      private double rental_rate;
25      private int length;
26      private double replacement_cost;
27      private String rating;
28      private String special_features;
29      private String lang_name;
30
31      //Default constructor
32      Inventory(){
33          this(1, 1, 1, new
34  Timestamp(System.currentTimeMillis()));
35      }
36
37      //Three-params constructor
38      Inventory(int film_id, int
39  store_id, Timestamp lu){
40          setFilmID(film_id);
41          setStoreID(store_id);
42          setLastUpdate(lu);
43      }
44
45      //Four-params constructor
46      Inventory(int inv_id, int film_id,
47  int store_id, Timestamp lu){
48          this(film_id, store_id, lu);
49          setInventoryID(inv_id);
50      }
51
52      //Twenty-one-params constructor
53      Inventory(int inv_id, int film_id,
54  int store_id, int add_id, String add,
55          String dist, String post,
56          String ph, String city, String
57  country, String title,
58          String description, int year,
59  int duration, double rate,
60          int length, double cost, String
61  rating,
62          String features, String
63  lang_name, Timestamp lu){
64          this(inv_id, film_id, store_id,
65  lu);
66          this.address_id = add_id;
67          this.address = add;
68          this.district = dist;
69          this.postal_code = post;
70          this.phone = ph;
71          this.city = city;
72          this.country = country;
73          this.title = title;
74
```

```
75          this.description = description;
76          this.release_year = year;
77          this.rental_duration =
78      duration;
79          this.rental_rate = rate;
80          this.length = length;
81          this.replacement_cost = cost;
82          this.rating = rating;
83          this.special_features =
84      features;
85          this.lang_name = lang_name;
86      }
87
88      //Getter methods
89      public int getInventoryID() {return
90      inventory_id;}
91      public int getFilmID() {return
92      film_id;}
93      public int getStoreID() {return
94      store_id;}
95      public Timestamp getLastUpdate()
96      {return last_update;}
97
98      public int getAddressID() {return
99      address_id;}
100     public String getAddress() {return
101     address;}
102     public String getDistrict() {return
103     district;}
104     public String getPostalCode()
105     {return postal_code;}
106     public String getPhone() {return
107     phone;}
108     public String getCity() {return
109     city;}
110     public String getCountry() {return
111     country;}
112
113
114     public String getTitle() {return
115     title;}
116     public String getDescription()
117     {return description;}
118     public int getReleaseYear() {return
119     release_year;}
120     public int getRentalDuration()
121     {return rental_duration;}
122     public double getRentalRate()
123     {return rental_rate;}
124     public int getLength() {return
125     length;}
126127  public double getReplacementCost()
128     {return replacement_cost;}
129
```

```
    public String getRating() {return rating;}
    public String getSpecialFeatures() {return special_features;}
    public String getLanguageName() {return lang_name;}

    //Setter methods
    public void setInventoryID(int id) {
        if (id <= 0) {
            throw new IllegalArgumentException("Inventory ID must be greater than zero.");
        }
        this.inventory_id = id;
    }

    public void setStoreID(int id) {
        if (id <= 0) {
            throw new IllegalArgumentException("Store ID must be greater than zero.");
        }
        this.store_id = id;
    }

    public void setFilmID(int id) {
        if (id <= 0) {
            throw new IllegalArgumentException("Film ID must be greater than zero.");
        }
        this.film_id = id;
    }

    public void setLastUpdate(Timestamp date){
        if (date == null) {
        throw new IllegalArgumentException("Date cannot be null");
        }
        this.last_update = date;
    }

    // Override the hashCode() method
    @Override
    public int hashCode() {
        return Objects.hash(inventory_id, store_id, film_id, last_update);
```

```
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() !=
o.getClass()) return false;
        Inventory in = (Inventory) o;
        return inventory_id ==
in.inventory_id &&
                store_id == in.store_id
&&
                film_id == in.film_id &&

Objects.equals(last_update,
in.last_update);
    }

    @Override
    public String toString(){
        return  "\nInventory ID
: " + getInventoryID() +
                "\nFilm ID
: " + getFilmID() +
                "\nTitle
: " + getTitle() +
                "\nDescription
: " + getDescription() +
                "\nRelease Year
: " + getReleaseYear() +
                "\nLanguage Name
: " + getLanguageName() +
                "\nRental Duration
: " + getRentalDuration()+
                "\nRental Rate
: " + getRentalRate()+
                "\nLength
: " + getLength()+
                "\nReplacement Cose
: " + getReplacementCost()+
                "\nRating
: " + getRating()+
                "\nSpecial Features
: " + getSpecialFeatures()+
                "\nStore ID
: " + getStoreID() +
                "\nAddress ID
: " + getAddressID() +
                "\nAddress
: " + getAddress() +
                "\nDistrict
: " + getDistrict() +
```

```
                    "\nPostal Code
: " + getPostalCode() +
                    "\nPhone
: " + getPhone() +
                    "\nCity
: " + getCity() +
                    "\nCountry
: " + getCountry() +
                    "\nLast Update
: " + getLastUpdate();
    }
}
```

Step 3

Create a new public class named **Inventory_Utils**. The class extends **Utility** and contains several methods for creating, populating, and reading an inventory table.

The class has a constant FIRST_INDEX with a value of 0 and an INVALID_INDEX constant with a value of -1. It also has a private static variable called currentIndex initialized to FIRST_INDEX. There is a constant string SQL_ID initialized to the result of the **get_sql_id()** method in the **Query_Inventory** class.

The class has a method called **create_inventory_table()** that creates an inventory table by executing a SQL script obtained from the **Query_Inventory** class. If successful, a message dialog is displayed to indicate that the table was successfully created. If an exception occurs, an error message is displayed.

The class also has a method called **populate_inventory_table()** that populates the inventory table with data. Two **Inventory** objects are created, one with a default constructor and the other with a four-parameter constructor. The object data is used to set the parameters of a prepared statement that is executed to insert the data into the inventory table. If an exception occurs, an error message is displayed.

The class also has a method called **read_inventory_table()** that reads the content of

a joined inventory, film, language, store, address, city, and country table by executing a SQL script obtained from the **Query_Inventory** class. The retrieved data is used to create an Inventory object with a twenty-one-parameter constructor, which is printed to the console. If an exception occurs, an error message is displayed.

The class imports several classes and interfaces, including Dimension, Logger, Connection, SQLException, ArrayList, Arrays, HashMap, JComboBox, JOptionPane, JPanel, TableModelEvent, DefaultTableModel, TableModel, DefaultCategoryDataset, and DefaultPieDataset. It also imports static methods from the Utility, Query_Inventory, and draw_barchart_with_dataset classes.

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
```

```
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
```

```
72  package sakila;
73  import java.awt.Dimension;
74  import java.util.logging.Level;
75  import java.util.logging.Logger;
76  import java.sql.*;
77  import java.util.ArrayList;
78  import java.util.Arrays;
79  import java.util.HashMap;
80  import java.util.Objects;
81  import javax.swing.JComboBox;
82  import javax.swing.JOptionPane;
83  import javax.swing.JPanel;
84  import javax.swing.event.TableModelEvent;
85  import javax.swing.table.DefaultTableModel;
86  import javax.swing.table.TableModel;
87  import org.jfree.data.category.DefaultCategoryDataset;
88  import org.jfree.data.general.DefaultPieDataset;
89  import static sakila.Utility.create_bar_dataset;
90  import static sakila.Utility.draw_barchart_with_dataset;
91
92  public class Inventory_Utils extends Utility{
93      public static final int FIRST_INDEX = 0;
94      public static final int INVALID_INDEX = -1;
95
96      private static int currentIndex = FIRST_INDEX;
97      private static final String SQL_ID =
98  Query_Inventory.get_sql_id();
99
100     //Creates inventory table
101     public static void create_inventory_table() {
102         try (Connection conn = getConnection()) {
103             Statement stmt = conn.createStatement();
104
105 stmt.addBatch(Query_Inventory.get_sql_inventory());
106             stmt.executeBatch();
107
108             String message = String.format("Successfully
109 creates inventory table");
110             JOptionPane.showMessageDialog(null, message,
111             "INFORMATION",JOptionPane.INFORMATION_MESSAGE);
112
113         } catch (SQLException ex) {
114             JOptionPane.showMessageDialog(null,
115 ex.getMessage(),
116                 "ERROR",JOptionPane.ERROR_MESSAGE);
117         }
118     }
119
120     //Populates inventory table with some rows of data
121     public static void populate_inventory_table(){
122         try(Connection conn = getConnection()){
123             String sql = """
124
```

```
                INSERT INTO inventory(inventory_id, film_id,
store_id, last_update)
                VALUES(?, ?, ?, ?)""";

            //Creates a new Inventory class with default
constructor
            PreparedStatement ps1 =
conn.prepareStatement(sql);
            Inventory obj1 = new Inventory();
            ps1.setInt(1,obj1.getInventoryID());
            ps1.setInt(2,obj1.getFilmID());
            ps1.setInt(3,obj1.getStoreID());
            ps1.setTimestamp(4,obj1.getLastUpdate());

            // Creates a new Inventory class with four-params
constructor
            PreparedStatement ps2 =
conn.prepareStatement(sql);
            Inventory obj2 = new Inventory(2, 2, 2, new
Timestamp(System.currentTimeMillis()));
            ps2.setInt(1,obj2.getInventoryID());
            ps2.setInt(2,obj2.getFilmID());
            ps2.setInt(3,obj2.getStoreID());
            ps2.setTimestamp(4,obj2.getLastUpdate());

            ps1.executeUpdate();
            ps2.executeUpdate();

        }catch(SQLException ex){
            JOptionPane.showMessageDialog(null,
ex.getMessage(),
                "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    //Reads the content of joined inventory, film, language,
store, address, city and country tables
    public static void read_inventory_table(){
        try(Connection conn = getConnection()){
            Statement stmt = conn.createStatement();
            ResultSet rs =
stmt.executeQuery(Query_Inventory.get_sql_inventory_joint());

            while(rs.next()){
                int inv_id = rs.getInt("inventory_id");
                int film_id = rs.getInt("film_id");
                int store_id = rs.getInt("store_id");
                int add_id = rs.getInt("address_id");
                String add = rs.getString("address");
                String dist = rs.getString("district");
                String post = rs.getString("postal_code");
                String ph = rs.getString("phone");
                String city = rs.getString("city");
```

```
                String country = rs.getString("country");
                String title = rs.getString("title");
                String description =
rs.getString("description");
                int year = rs.getInt("release_year");
                int duration = rs.getInt("rental_duration");
                double rate = rs.getDouble("rental_rate");
                int length = rs.getInt("length");
                double cost =
rs.getDouble("replacement_cost");
                String rating = rs.getString("rating");
                String features =
rs.getString("special_features");
                String lang_name =
rs.getString("language_name");

                Timestamp lu =
rs.getTimestamp("last_update");

                //Creates a Inventory object using twenty-
one-params constructor
                Inventory obj = new Inventory(inv_id,
film_id, store_id, add_id, add, dist, post, ph, city,
country, title, description, year, duration, rate,
                    length, cost, rating, features,
lang_name, lu);
                System.out.println(obj);
            }
            rs.close();
            stmt.close();

        }catch(SQLException ex){
            JOptionPane.showMessageDialog(null,
ex.getMessage(),
                "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }
}
```

Step 4

In the
**create_i
populate
read_inv**

```
1
2
3
4
5
6
```

```
//
//
//
//

//
//
Langu
//
//
//

//
//
Categ
//
//
//

//
//
//
//
//

//
Film/
//
Film/
//
//
//

//
Film(
//
Film(
//
Film(
//
Film(
//

//
//
//
//
//

//
//
//
//
```

```
61 //
62
63 //
64 //
65 //
66 //
67 //
68
69 //
70 //
   //
   //
   //
   Inver
       }
   }
```

Run proj

```
Inventory
Film ID
Title
Descripti
Release Y
Language
Rental Du
Rental Ra
Length
Replaceme
Rating
Special F
Store ID
Address I
Address
District
Postal Co
Phone
City
Country
Last Upda

Inventory
Film ID
Title
Descripti
Release Y
Language
Rental Du
```

DESIGNING GUI
DESIGNING GUI

Step 1	In the project, create a new **JFrame Form** and name it as **InventoryForm.java**. In the Design tab, add twenty-one **JLabel** to the form and set their corresponding **text** properties as INVENTORY ID, FILM ID, STORE ID, TITLE, DESCRIPTION, RELEASE YEAR, RENTAL RATE, RENTAL DURATION, LENGTH, REPLACEMENT COST, RATING, LANGUAGE, SPECIAL FEATURES, ADDRESS ID, DISTRICT, CITY, COUNTRY, PHONE, POSTAL CODE, and LAST UPDATE.
Step 2	Then, add twelve **JTextField** to the form and set set their corresponding **Variable Name** as jtfInventoryID, **jtfFilmID**, **jtfStoreID**, **jtfTitle**, **jtfDescription**, **jtfReleaseYear**, **jtfRentalRate**, **jtfLength**, **jtfRentalDuration**, **jtfReplacementCost**, **jtfLanguage**, **jtfSpecialFeatures**, **jtfAddressID**, **jtfDistrict**, **jtfCity**, **jtfCountry**, **jtfPhone**, **jtfPostalCode**, and **jtfLastUpdate**.
Step 3	Then, add twelve **JButton** to the form and set their corresponding **Variable Name** as **jbFirst**, **jbPrev**, **jbNext**, **jbLast**, **jbEdit**, **jbInsert**, **jbDelete**, **jbChart**, **jbStoreForm**, **jbAddressForm**, **jbCountryForm**, and

	jbCityForm. Set their corresponding **text** properties as FIRST, PREV, NEXT, LAST, EDIT, INSERT, DELETE, CHART, STORE FORM, ADDRESS FORM, COUNTRY FORM, and CITY FORM.
Step 4	Then, add three **JComboBox** to the form and set set their corresponding **Variable Name** as **jcbInventoryID**, **jcbFilmID**, and **jcbStoreID**.
Step 5	Lastly, add a new **JTable** to the form set set its **Variable Name** as **jtInventory**. Then, right-click on it, then choose **Table Contents...** and set the number of columns to 21 and the number of rows to 50.

Step 6	In the driver class, **Sakila.java**, create a new object of **InventoryForm** class using its default constructor as shown in 15 – 16:
	
	1 package sakila;
	2
	3 public class Sakila {
	4 public static void main(String[] args)
	5 {
	6 //...
	7 // Store_Utils.create_store_table();
	8 //
	9 Store_Utils.populate_store_table();
	10 // Store_Utils.read_store_table();
	11 // StoreForm frm = new StoreForm();
	12 // frm.setVisible(true);
	13
	14 //
	15 Inventory_Utils.create_inventory_table();
	16 //
	17 Inventory_Utils.populate_inventory_table();
	18 //
	Inventory_Utils.read_inventory_table();
	InventoryForm frm = new
	InventoryForm();
	frm.setVisible(true);
	}
	}

Step 8	In **InventoryForm**'s constructor, invoke

setLookAndFeel() to set the look and feel of the form as shown in line 17.

```
1  package sakila;
2
3  import java.awt.Toolkit;
4  import java.awt.event.ActionEvent;
5  import
6  java.awt.event.ActionListener;
7  import javax.swing.JButton;
8  import javax.swing.JComboBox;
9  import javax.swing.JMenuItem;
10 import javax.swing.JOptionPane;
11 import javax.swing.JPopupMenu;
12 import javax.swing.JTable;
13 import javax.swing.JTextField;
14
15 public class InventoryForm extends
16 javax.swing.JFrame {
17     public InventoryForm() {
18         initComponents();
19
20 Utility.setLookAndFeel(this);
       }
       //...
   }
```

Run the project to see the inventory form as shown in Figure 12.1.

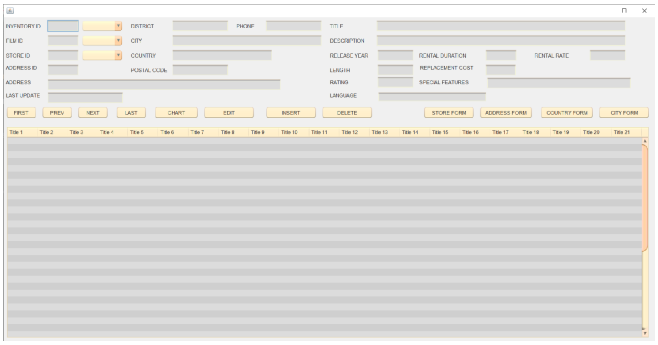


Figure 12.1 The layout of inventory form

Step 9

In **InventoryForm.java**, define **getter()** method for every object in the form and place them outside and beneath its default constructor:

```
1        //Getter method for
2    jtfInventoryID
3
```

```
    public JTextField
getJTFInventoryID(){
        return
this.jtfInventoryID;
    }

    //Getter method for
jcbInventoryID
    public JComboBox
getJCBInventoryID(){
        return
this.jcbInventoryID;
    }

    //Getter method for
jtfFilmID
    public JTextField
getJTFFilmID(){
        return this.jtfFilmID;
    }

    //Getter method for
jcbFilmID
    public JComboBox
getJCBFilmID(){
        return this.jcbFilmID;
    }

    //Getter method for
jtfStoreID
    public JTextField
getJTFStoreID(){
        return this.jtfStoreID;
    }

    //Getter method for
jcbStoreID
    public JComboBox
getJCBStoreID(){
        return this.jcbStoreID;
    }

   //Getter method for
jtInventory
    public JTable
getJTInventory(){
        return this.jtInventory;
    }

    //Getter method for
jtfAddressID
    public JTextField
getJTFAddressID(){
```

```
58          return
59      this.jtfAddressID;
60          }
61
62          //Getter method for
63      jtfAddress
64          public JTextField
65      getJTFAddress(){
66              return this.jtfAddress;
67          }
68
69          //Getter method for
70      jtfDistrict
71          public JTextField
72      getJTFDistrict(){
73              return this.jtfDistrict;
74          }
75
76          //Getter method for jtfPhone
77          public JTextField
78      getJTFPhone(){
79              return this.jtfPhone;
80          }
81
82          //Getter method for jtfCity
83          public JTextField
84      getJTFCity(){
85              return this.jtfCity;
86          }
87
88          //Getter method for
89      jtfCountry
90          public JTextField
91      getJTFCountry(){
92              return this.jtfCountry;
93          }
94
95          //Getter method for
96      jtfPostalCode
97          public JTextField
98      getJTFPostalCode(){
99              return
100     this.jtfPostalCode;
101         }
102
103         //Getter method for jtfTitle
104         public JTextField
105     getJTFTitle(){
106             return this.jtfTitle;
107         }
108
109         //Getter method for
110     jtfDescription
111
```

```
    public JTextField
getJTFDescription(){
        return
this.jtfDescription;
    }

    //Getter method for
jtfReleaseYear
    public JTextField
getJTFReleaseYear(){
        return
this.jtfReleaseYear;
    }

    //Getter method for
jtfRentalDuration
    public JTextField
getJTFRentalDuration(){
        return
this.jtfRentalDuration;
    }

    //Getter method for
jtfRentalRate
    public JTextField
getJTFRentalRate(){
        return
this.jtfRentalRate;
    }

    //Getter method for
jtfLength
    public JTextField
getJTFLength(){
        return this.jtfLength;
    }

    //Getter method for
jtfReplacementCost
    public JTextField
getJTFReplacementCost(){
        return
this.jtfReplacementCost;
    }

    //Getter method for
jtfSpecialFeatures
    public JTextField
getJTFSpecialFeatures(){
        return
this.jtfSpecialFeatures;
    }
```

```
    //Getter method for
jtfRating
    public JTextField
getJTFRating(){
        return this.jtfRating;
    }

    //Getter method for
jtfLanguage
    public JTextField
getJTFLanguage(){
        return this.jtfLanguage;
    }

    //Getter method for
jtfLastUpdate
    public JTextField
getJTFLastUpdate(){
        return
this.jtfLastUpdate;
    }

    //Getter method for jbEdit
    public JButton getJBEdit(){
        return this.jbEdit;
    }

    //Getter method for jbInsert
    public JButton getJBInsert()
{
        return this.jbInsert;
    }

    //Getter method for jbDelete
    public JButton getJBDelete()
{
        return this.jbDelete;
    }

    //Getter method for jbChart
    public JButton getJBChart(){
        return this.jbChart;
    }

    //Getter method for jbFirst
    public JButton getJBFirst(){
        return this.jbFirst;
    }

    //Getter method for jbPrev
    public JButton getJBPrev(){
        return this.jbPrev;
    }
```

```
    //Getter method for jbNext
    public JButton getJBNext(){
        return this.jbNext;
    }

    //Getter method for jbLast
    public JButton getJBLast(){
        return this.jbLast;
    }
```

POPULATING TABLE AND COMBOBOXES
POPULATING TABLE AND COMBOBOXES

Step 1

In **Inventory_Utils.java**, add two new methods: **get_inventory_list()** and of these methods is to retrieve and display inventory data from a database

get_inventory_list() takes in an **InventoryForm** object, a SQL query, a parameters to create a **PreparedStatement** and execute a query agains not "none", it sets a parameter in the **PreparedStatement** to the value of and creates Inventory objects from the data. These objects are added to a of the method.

show_table_inventory() takes in an **InventoryForm** object and a parameters. It creates a **DefaultTableModel** with 21 columns and set **TableModel** of the **jtInventory** component of the **InventoryForm** objec method then iterates over the **ArrayList** and adds each Inventory object to

```
1     private static ArrayList<Inventory>
2  get_inventory_list(InventoryForm frm, String sql, String item){
3         ArrayList<Inventory> list = new ArrayList<>();
4
5         try(Connection conn = getConnection();
6             PreparedStatement ps = conn.prepareStatement(sql)){
7             if (item.equalsIgnoreCase("none")==false) {
8                  ps.setString(1,item);
9             }
10            ResultSet rs = ps.executeQuery();
11
12            Inventory obj;
13            while(rs.next()){
14                //Using twenty-one-params constructor
15                obj = new Inventory(rs.getInt("inventory_id"),
16                    rs.getInt("film_id"),
17                    rs.getInt("store_id"),
```

```
                    rs.getInt("address_id"),
                    rs.getString("address"),
                    rs.getString("district"),
                    rs.getString("postal_code"),
                    rs.getString("phone"),
                    rs.getString("city"),
                    rs.getString("country"),
                    rs.getString("title"),
                    rs.getString("description"),
                    rs.getInt("release_year"),
                    rs.getInt("rental_duration"),
                    rs.getDouble("rental_rate"),
                    rs.getInt("length"),
                    rs.getDouble("replacement_cost"),
                    rs.getString("rating"),
                    rs.getString("special_features"),
                    rs.getString("language_name"),
                    rs.getTimestamp("last_update"));

                list.add(obj);
            }
        }catch (SQLException ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
                "ERROR",JOptionPane.ERROR_MESSAGE);
        }
        return list;
    }

    private static void show_table_inventory(InventoryForm frm,
ArrayList<Inventory> list) throws SQLException{
        DefaultTableModel model = new DefaultTableModel(0,0);

        String header[] = {"Inventory ID", "Film ID", "Store ID",
"Address ID", "Address", "District", "Postal Code", "Phone",
"City", "Country",
            "Title", "Description", "Release Year", "Rental
Duration",
            "Rental Rate", "Length", "Replacement Cost", "Rating",
"Special Features", "Language Name", "Last Update"};


model.setColumnIdentifiers(set_column_header(frm.getJTInventory(),
header));
        frm.getJTInventory().setModel(model);

        Object[] row = new Object[21];

        for(int i=0; i<list.size(); i++){
            row[0] = list.get(i).getInventoryID();
            row[1] = list.get(i).getFilmID();
            row[2] = list.get(i).getStoreID();
            row[3] = list.get(i).getAddressID();
            row[4] = list.get(i).getAddress();
            row[5] = list.get(i).getDistrict();
```

```
            row[6] = list.get(i).getPostalCode();
            row[7] = list.get(i).getPhone();
            row[8] = list.get(i).getCity();
            row[9] = list.get(i).getCountry();
            row[10] = list.get(i).getTitle();
            row[11] = list.get(i).getDescription();
            row[12] = list.get(i).getReleaseYear();
            row[13] = list.get(i).getRentalDuration();
            row[14] = list.get(i).getRentalRate();
            row[15] = list.get(i).getLength();
            row[16] = list.get(i).getReplacementCost();
            row[17] = list.get(i).getRating();
            row[18] = list.get(i).getSpecialFeatures();
            row[19] = list.get(i).getLanguageName();
            row[20] = list.get(i).getLastUpdate();

            model.addRow(row);
        }
    }
```

Step 2

In **Inventory_Utils.java**, define **refresh_controls()** method. This is a met that refreshes the controls on an inventory form. Here's what it does:

1. Sets the location and title of the inventory form.
2. Populates the inventory table with data from the database, sorted b
3. Populates the inventory ID, film ID, and store ID combo boxes with
4. If there is a SQL exception, it shows an error message.

```
    public static void refresh_controls(InventoryForm frm){
        frm.setLocationRelativeTo(null);
        frm.setTitle("INVENTORY FORM");

        //Shows the content of inventory table and populates combobox
        try{
            //Makes alternating color for table rows

table_renderer(frm.getJTInventory());

            //Populates table
            ArrayList<Inventory> list = get_inventory_list(frm, Query_Inventory.get_sql_inventory_joint() + " ORDER BY inventory_id", "none");
            show_table_inventory(frm, list);

            //Populates jcbInventoryID
```

```
            String sql_inv_id = "SELECT
inventory_id FROM inventory ORDER BY
inventory_id";
            populate_combobox(sql_inv_id,
frm.getJCBInventoryID(), frm);

            //Populates jcbFilmID
            String sql_film_id = "SELECT
film_id FROM film ORDER BY film_id";

populate_combobox(sql_film_id,
frm.getJCBFilmID(), frm);

            //Populates jcbStoreID
            String sql_store_id = "SELECT
store_id FROM store ORDER BY store_id";

populate_combobox(sql_store_id,
frm.getJCBStoreID(), frm);

        }catch (SQLException ex){

JOptionPane.showMessageDialog(frm,
ex.getMessage(),

 "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }
```

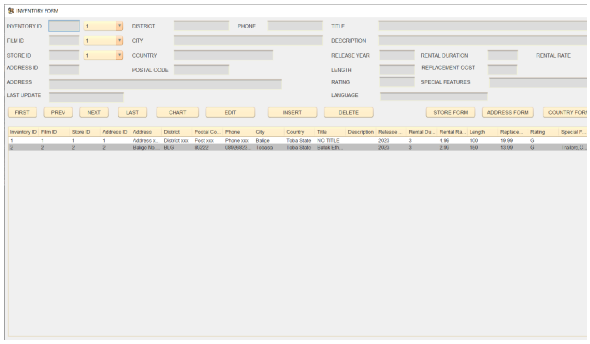


Figure 12.2 The content of **inventory** table disp

Step 3

In **InventoryForm**'s default constructor, the **Inventory_Utils.refresh** populates the controls in the **InventoryForm** with data from a databas from the **Inventory_Utils** class.

The **this.setIconImage()** method sets the icon of the **In this.setDefaultCloseOperation(this.HIDE_ON_CLOSE)** method sets th to hide the form instead of exiting the application when the close button is

```
1  package sakila;
2  import java.awt.Toolkit;
3  import java.awt.event.ActionEvent;
4  import java.awt.event.ActionListener;
5  import java.io.IOException;
6  import java.text.ParseException;
7  import java.util.logging.Level;
8  import java.util.logging.Logger;
9  import javax.swing.JButton;
10 import javax.swing.JComboBox;
11 import javax.swing.JLabel;
12 import javax.swing.JTable;
13 import javax.swing.JTextField;
14
15 public class InventoryForm extends javax.swing.JFrame {
16     public InventoryForm() {
17         initComponents();
18         Utility.setLookAndFeel(this);
19         Inventory_Utils.refresh_controls(this);
20
21  this.setIconImage(Toolkit.getDefaultToolkit().getImage(getClass().
22 ;
23         this.setDefaultCloseOperation(this.HIDE_ON_CLOSE);
24     }
25     //...
26 }
```

Step 4

Run the project to see the content of **inventory** table displayed in **jtInvent**

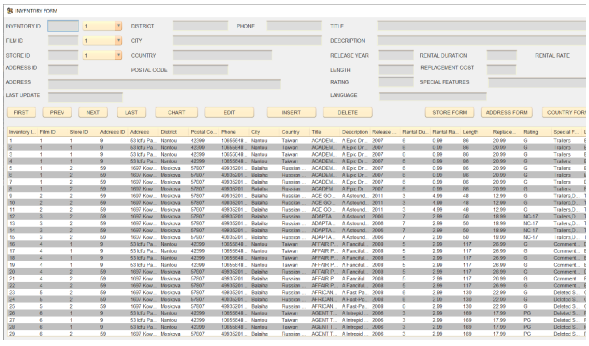


Figure 12.3 The the content of **inventory** table in original **Sakila** databa
jtInventory

If you use the data from **Sakila** MySQL database available in the int
inventory table displayed in **jtInventory** as shown in Figure 12.3.

DISPLAYING AND NAVIGATING DATA ROW BY ROW

DISPLAYING AND NAVIGATING DATA ROW BY ROW

Step 1

In **Inventory_Utils**, define two new methods named **clear_sto** **display_store_data()**. The **clear_store_controls()** method is a private n **InventoryForm** parameter frm. It clears the text fields of the store info The **display_store_data()** method is a generic method that also takes parameter frm, a SQL string sql, and an item item of type T. This method the database using the given SQL string and item, retrieves the result s store, address, country, and city data of the first row in the result set in th fields of the **InventoryForm** form. If the result set is empty, the method of the store information. The method also determines the selected iten combo box based on the store ID retrieved from the database. If an err database query, the method displays an error message in a dialog box.

```
1     private static void clear_store_controls(InventoryForm frm){
2         frm.getJTFAddressID().setText("");
3         frm.getJTFAddress().setText("");
4         frm.getJTFDistrict().setText("");
5         frm.getJTFPhone().setText("");
6         frm.getJTFPostalCode().setText("");
7         frm.getJTFCity().setText("");
8         frm.getJTFCountry().setText("");
9     }
10
11   //Displays store, address, country, and city data result row by
12    private static <T> void display_store_data(InventoryForm frm,
13 String sql, T item){
14        try(Connection conn = getConnection()){
15            PreparedStatement ps = conn.prepareStatement(sql);
16            ps.setObject(1,item);
17            ResultSet rs = ps.executeQuery();
18
19            if (!rs.next()) {
20                // no row found, clear the form fields
21                clear_store_controls(frm);
22                return;
23            }
24
25            do{
26
27 frm.getJTFStoreID().setText(String.valueOf(rs.getInt("store_id")));
28
29 frm.getJTFAddressID().setText(String.valueOf(rs.getInt("address_id"
30                frm.getJTFAddress().setText(rs.getString("address")
31                frm.getJTFDistrict().setText(rs.getString("district
32                frm.getJTFPhone().setText(rs.getString("phone"));
33
34 frm.getJTFPostalCode().setText(rs.getString("postal_code"));
35                frm.getJTFCity().setText(rs.getString("city"));
36                frm.getJTFCountry().setText(rs.getString("country")
37
```

```
                // Determines item selected from jcbStoreID
                find_combo_value_selected(frm.getJCBStoreID(),
rs.getInt("store_id"));

            }while(rs.next());

            rs.close();
            ps.close();
        }catch(SQLException ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }
```

Step 2

Still in the same class, define another method named **jcbStc** **jcbStore_handler()** method is a handler method that is triggered w **jcbStoreID** combo box is selected. It retrieves the selected item from passes it to the **display_store_data()** method along with an **display_store_data()** method then executes the SQL query with the parameter and displays the resulting store data in the relevant fields in object.

```
    public static void
jcbStore_handler(InventoryForm frm) {
        Object item =
frm.getJCBStoreID().getSelectedItem();
        display_store_data(frm,
Query_Store.get_sql_store_joint() + "
WHERE store_id = ?", item);
    }
```

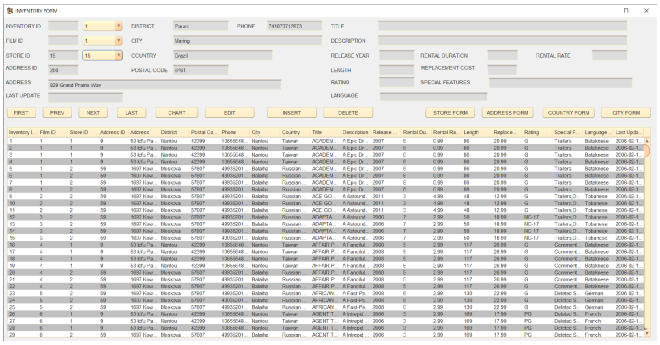


Figure 12.4 Displaying row by row the content of joined **store**, **address** tables

Step 3

In **InventoryForm**, double click on **jcbStoreID** combobox and define its handler as follows:

```
```

```
    private void
jcbStoreIDActionPerformed(java.awt.event.ActionEvent
evt) {
        Inventory_Utils.jcbStore_handler(this);
    }
```

This is an event handling method that gets called when the value of the **jcl** is changed by the user. It calls the **jcbStore_handler()** method of the **In** and passes the **InventoryForm** instance (**this**) as an argument to **jcbStore_handler()** method then retrieves the selected item from the co database for the store data related to that item, and displays that data in t

Step 4

Run the project. Choose one of items in **jcbStoreID** combobox to see rov of joined **store**, **address**, **city** and **country** tables as shown in Figure 12.4

Step 5

In **Inventory_Utils**, define two new methods named **clear_fi display_film_data()**. The **display_film_data()** method is responsible for of the **frm** parameter with data retrieved from the database using the **sql** parameter.

The method starts by creating a connection to the database using th method from the **Inventory_Utils** class. Then, a PreparedStatement is cr statement passed in the **sql** parameter. The item parameter is set as the the **PreparedStatement** using the setObject method.

Next, the **PreparedStatement** is executed to obtain a **ResultSet** obje result of the SQL query. If no row is found in the result set, **clear_film_controls()** to clear the fields in the frm parameter, and returns

If there is at least one row in the result set, the method loops through t while loop. For each row, the method populates the appropriate fields i with the corresponding column values from the **ResultSet**.

Finally, the **ResultSet** and **PreparedStatement** are closed, and any SQ and displayed in a **JOptionPane**.

```
    private static void clear_film_controls(InventoryForm frm){
        frm.getJTFFilmID().setText("");
        frm.getJTFTitle().setText("");
        frm.getJTFDescription().setText("");
        frm.getJTFReleaseYear().setText("");
        frm.getJTFRentalDuration().setText("");
        frm.getJTFRentalRate().setText("");
        frm.getJTFLength().setText("");
        frm.getJTFReplacementCost().setText("");
        frm.getJTFSpecialFeatures().setText("");
        frm.getJTFLastUpdate().setText("");
```

```
    }

   //Displays joined film and language data result row by row
    private static <T> void display_film_data(InventoryForm frm, St
        try(Connection conn = getConnection()){
            PreparedStatement ps = conn.prepareStatement(sql);
            ps.setObject(1,item);
            ResultSet rs = ps.executeQuery();

            if (!rs.next()) {
                // no row found, clear the form fields
                clear_film_controls(frm);
                return;
            }

            do{
                frm.getJTFFilmID().setText(String.valueOf(rs.getInt
                frm.getJTFTitle().setText(rs.getString("title"));
                frm.getJTFDescription().setText(rs.getString("descr

frm.getJTFReleaseYear().setText(String.valueOf(rs.getInt("release_y

frm.getJTFRentalDuration().setText(String.valueOf(rs.getInt("rental

frm.getJTFRentalRate().setText(String.valueOf(rs.getDouble("rental_
                frm.getJTFLength().setText(String.valueOf(rs.getInt

frm.getJTFReplacementCost().setText(String.valueOf(rs.getDouble("re
                frm.getJTFSpecialFeatures().setText(rs.getString("s
                frm.getJTFLanguage().setText(rs.getString("language
                frm.getJTFRating().setText(rs.getString("rating"));

                // Determines item selected from jcbFilmID
                find_combo_value_selected(frm.getJCBFilmID(), rs.ge

            }while(rs.next());

            rs.close();
            ps.close();
        }catch(SQLException ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }
```

Step 6	Still in the same class, define another method named **jcbFilm_handler()**. that takes an **InventoryForm** object as a parameter. It first gets the se

JComboBox component **jcbFilmID** in the form and passes it to the method along with a SQL query to retrieve the film and language data f The retrieved data is then displayed in the form.

Next, the method retrieves the inventory data for the selected film generated by the **Query_Inventory.get_sql_inventory_joint()** method a **get_inventory_list()** method to retrieve a list of **Inventory** objects. Finall the **InventoryForm** object and the list of Inventory objects to the **show** method to display the inventory data in a table in the form.

```
    public static void
jcbFilm_handler(InventoryForm frm) throws
SQLException {
        Object item =
frm.getJCBFilmID().getSelectedItem();
        display_film_data(frm,
Query_Film.get_sql_film_joint() + " WHERE
film_id = ?", item);

        //Filters table
        String sql =
Query_Inventory.get_sql_inventory_joint()
+ " WHERE i.film_id = ? ORDER BY
i.inventory_id";
        ArrayList<Inventory> list =
get_inventory_list(frm, sql,
String.valueOf(item));
        show_table_inventory(frm, list);
    }
```

Step 7

In **InventoryForm**, double click on **jcbFilmID** combobox and define its handler as follows:

```
    private void
jcbFilmIDActionPerformed(java.awt.event.ActionEvent evt) {
        try {
            Inventory_Utils.jcbFilm_handler(this);
        } catch (SQLException ex) {

Logger.getLogger(InventoryForm.class.getName()).log(Level.SEVERE,
null, ex);
        }
    }
```

It is an event handler method for the "action performed" event on the **jcb** When the user selects a film from the **JComboBox**, this method is cal **jcbFilm_handler()** method from **Inventory_Utils** class to display the fi the inventory table for the selected film.

If an SQL exception occurs in **the jcbFilm_handler()** method, the excep
the **Logger** class.

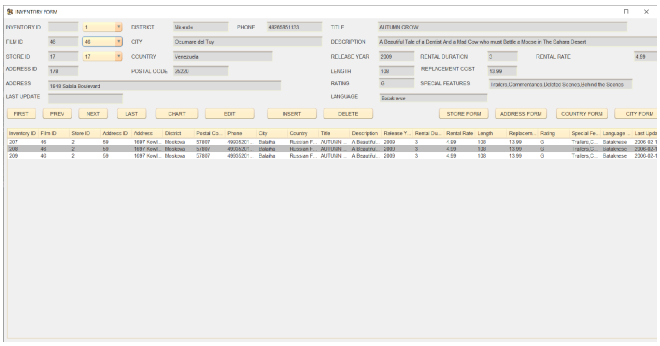


Figure 12.5 Displaying row by row the content of joined **film** and **la**

Step 8

Run the project. Choose one of items in **jcbFilmID** combobox to see row b
joined **film** and **language** tables as shown in Figure 12.5.

Step 9

In **Inventory_Utils** class, define a new method named **display_inv**
method is used to display inventory, store, address, country, city, film,
result row by row. It takes as parameters an instance of the **Inventor**
string, and an item of a generic type T.

Inside the method, a connection to the database is established using th
method. Then a **PreparedStatement** is created using the SQL string pa
and the item passed as a parameter is set as a value for the parameter in
query is executed using **executeQuery()** method and the resulting
obtained.

If no rows are found in the **ResultSet**, the form fields related to store, fil
cleared using the **clear_store_controls()** and **clear_film_controls()** m
update text field is set to empty.

If rows are found in the **ResultSet**, the form fields are populated wit
ResultSet using the **setText()** method of each form field. The item
jcbInventoryID, **jcbStoreID**, and **jcbFilmID** is determi
find_combo_value_selected() method.

Finally, the **ResultSet** object and **PreparedStatement** object ar
SQLException is caught and displayed in a message dialog.

```
1    //Displays inventory, store, address, country, city, film, and l
2  result row by row
3      private static <T> void display_inventory_data(InventoryForm fr
4  sql, T item){
5          try(Connection conn = getConnection()){
```

```
            PreparedStatement ps = conn.prepareStatement(sql);
            ps.setObject(1,item);
            ResultSet rs = ps.executeQuery();

            if (!rs.next()) {
                // no row found, clear the form fields
                clear_store_controls(frm);
                clear_film_controls(frm);
                frm.getJTFLastUpdate().setText("");
                return;
            }

            do{

frm.getJTFInventoryID().setText(String.valueOf(rs.getInt("inventory

frm.getJTFLastUpdate().setText(String.valueOf(rs.getTimestamp("last

                // Determines item selected from jcbInventoryID
                find_combo_value_selected(frm.getJCBInventoryID(),
rs.getInt("inventory_id"));

                // Determines item selected from jcbStoreID
                find_combo_value_selected(frm.getJCBStoreID(),
rs.getInt("store_id"));

                // Determines item selected from jcbFilmID
                find_combo_value_selected(frm.getJCBFilmID(),
rs.getInt("film_id"));

            }while(rs.next());

            rs.close();
            ps.close();
        }catch(SQLException ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }
```

Step 10

In the same class, define another method named **jcbInventory_handl** selection of an item from the **JComboBox** for inventory ID. It gets the se **JComboBox** and passes it as a parameter to the **display_inventory_data(** the SQL query to retrieve the inventory data for the selected **display_inventory_data()** method then executes the SQL query, retrieve displays the inventory, store, address, country, city, film, and language da fields by iterating over the result set row by row. Finally, the **find_coml** method is called to set the selected item in the **JComboBox** for store ID ar

```
    public static void
```

```
jcbInventory_handler(InventoryForm frm) {
        Object item =
frm.getJCBInventoryID().getSelectedItem();
        display_inventory_data(frm,
Query_Inventory.get_sql_inventory_joint()
+ " WHERE inventory_id = ?", item);
    }
```

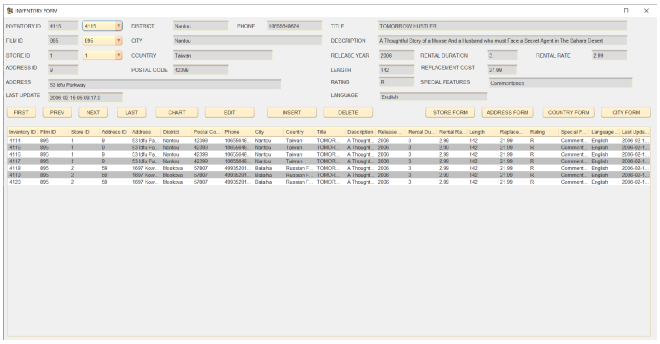


Figure 12.6 Displaying row by row the content of joined **inventory, store city, film,** and **language** tables

Step 11

In **InventoryForm**, double click on **jcbInventoryID** combobox and defi event handler as follows:

```
    private void
jcbInventoryIDActionPerformed(java.awt.event.ActionEvent
evt) {
        Inventory_Utils.jcbInventory_handler(this);
    }
```

Step 12

Run the project. Choose one of items in **jcbInventoryID** combobox to content of joined **inventory, store, address, country, city, film,** and shown in Figure 12.6.

Step 13

Define four navigating methods in **Inventory_Utils** class. These methods between rows in the inventory data displayed on the form.

The **show_first_row()** method is used to display the first row of the i **InventoryForm** form. It sets the current index to the first index and disp first item in the **JComboBox** component for inventory ID.

show_last_row() displays the last row of the inventory data by getting **JComboBox** and calling **display_inventory_data()** with the correspondin

show_prev_row() displays the previous row of the inventory data by decre index and checking if it is within bounds. If it is within bounds, it gets the index in the **JComboBox** and calls **display_inventory_data()** with the query.

show_next_row() displays the next row of the inventory data by incre index and checking if it is within bounds. If it is within bounds, it gets the index in the **JComboBox** and calls **display_inventory_data()** with the query.

```
1   public static void show_first_row(InventoryForm frm){
2       String item =
3   String.valueOf(frm.getJCBInventoryID().getItemAt(FIRST_INDEX));
4       display_inventory_data(frm, SQL_ID, item);
5       currentIndex = FIRST_INDEX;
6   }
7
8   public static void show_last_row(InventoryForm frm){
9       int endIndex = frm.getJCBInventoryID().getItemCount() -
10  1;
11      String item =
12  String.valueOf(frm.getJCBInventoryID().getItemAt(endIndex));
13      display_inventory_data(frm, SQL_ID, item);
14      currentIndex = endIndex;
15  }
16
17  public static void show_prev_row(InventoryForm frm){
18      currentIndex--;
19      if(currentIndex < FIRST_INDEX){
20          currentIndex = FIRST_INDEX;
21          return;
22      }
23      String item =
24  String.valueOf(frm.getJCBInventoryID().getItemAt(currentIndex));
25      display_inventory_data(frm, SQL_ID, item);
26  }
27
28  public static void show_next_row(InventoryForm frm){
29      int endIndex = frm.getJCBInventoryID().getItemCount() -
30  1;
31      currentIndex++;
32      if(currentIndex > endIndex){
33          currentIndex = endIndex;
34          return;
35      }
36      String item =
    String.valueOf(frm.getJCBInventoryID().getItemAt(currentIndex));
        display_inventory_data(frm, SQL_ID, item);
    }
```

Step 14

Then in **InventoryForm**, double click on each navigation buttons to define event handler:

```
    private void jbFirstActionPerformed(java.awt.event.ActionEvent evt) {
        Inventory_Utils.show_first_row(this);
    }

    private void jbPrevActionPerformed(java.awt.event.ActionEvent evt) {
        Inventory_Utils.show_prev_row(this);
    }

    private void jbNextActionPerformed(java.awt.event.ActionEvent evt) {
        Inventory_Utils.show_next_row(this);
    }

    private void jbLastActionPerformed(java.awt.event.ActionEvent evt) {
        Inventory_Utils.show_last_row(this);
    }
```

These are event handler methods for the "First", "Prev", "Next", and "
InventoryForm GUI. When one of these buttons is clicked, the corre
called, which in turn calls a method from the **Inventory_Utils** class to
previous row, next row, or last row of data in the inventory table, re
keyword in the method calls refers to the current **InventoryForm** instanc
a parameter to the **Inventory_Utils** methods.

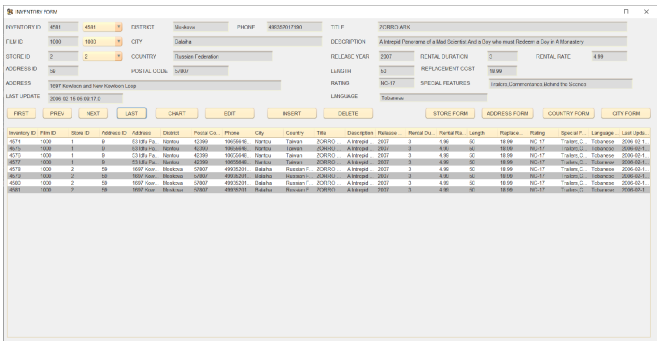


Figure 12.7 User clicks on one or more navigation buttons on inv

Step 15

Run the project. Click on one or more navigation buttons to see the resul
12.7.

Step 16

Define **mouse_pressed_handler()** method in **Inventory_Utils** class. This mouse press event on the **JTable** component in the **InventoryForm**. It fi selected, and if not, it shows a message dialog to inform the user. If a ro the inventory ID from the selected row and calls the **display_inventor** display the inventory data in the form.

The **getConnection()** method is used to establish a connection to the da exception that occurs is caught and logged. If an exception occurs, an err displayed to the user containing the error message and stack trace.

```
1     public static void mouse_pressed_handler(InventoryForm frm) {
2         Objects.requireNonNull(frm, "frm must not be null");
3
4         int selectedIndex = frm.getJTInventory().getSelectedRow();
5         if (selectedIndex == -1) {
6             JOptionPane.showMessageDialog(frm, "Please select a row to view its d
7                 "No row selected", JOptionPane.INFORMATION_MESSAGE)
8             return;
9         }
10
11        try (Connection conn = getConnection()) {
12            String id =
13 String.valueOf(frm.getJTInventory().getModel().getValueAt(selectedI
14 0));
15
16            // Displays inventory data
17            display_inventory_data(frm, SQL_ID, id);
18
19        } catch (SQLException ex) {
20
21 Logger.getLogger(InventoryForm.class.getName()).log(Level.SEVERE, "
22 displaying inventory data", ex);
23            String message = "Error displaying inventory data: " +
24 ex.getMessage();
25            String stackTrace = Arrays.toString(ex.getStackTrace())
26            JOptionPane.showMessageDialog(frm, message + "\n\n" +
27 stackTrace, "ERROR", JOptionPane.ERROR_MESSAGE);
        }
    }
```

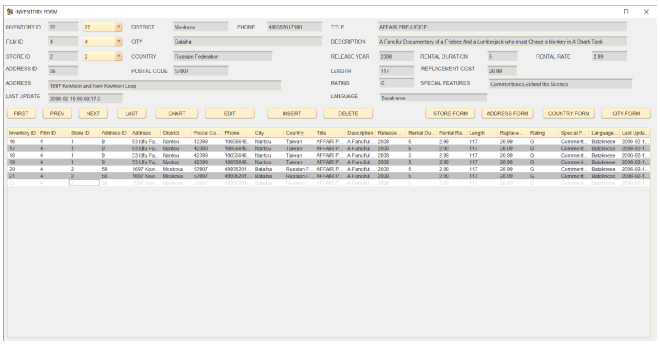


Figure 12.8 User double-clicks on any row in **jtInvento**

Step 17	Right click on **jtInventory**. Then, choose **Events > Mouse > mousePres** handler:
	
	`private void jtInventoryMousePressed(java.awt.event.MouseEvent evt) {`
	`Inventory_Utils.mouse_pressed_handler(this);`
	`}`

Step 18	Run the project. Double click on any row in **jtInventory** table. You will se row of joined inventory, **store**, **address**, **country**, **city**, **film**, and **languag** textfields and comboboxes as shown in Figure 12.8.

UPDATING RECORD

UPDATING RECORD

Step 1	In **Inventory_Utils** class, define a new method named **update_row_by_inventory_id()**. This method updates a row of data in the **inventory** table by the inventory ID. It takes as input parameters an **InventoryForm** object, an integer inventory ID, an integer film ID, and an integer store ID. It first establishes a connection to the database using the **getConnection()** method. It then creates a **ResultSet** object and a query string to search for the inventory ID in the **inventory** table. If the inventory ID is not found, an error message is displayed using a **JOptionPane** dialog.

If the inventory ID is found, it creates an **Inventory** object using the input parameters, and prepares an update query string to update the inventory table with the new film and store IDs for the given inventory ID. It then executes the update query using the **executeUpdate()** method of the **PreparedStatement** object.

If any **SQLException** or **NumberFormatException** is caught during the execution, an error message is displayed using a **JOptionPane** dialog. Finally, all database objects and connections are closed.

```
    //Updates row of data in inventory tabel by inventory_id
    public static void update_row_by_inventory_id(InventoryForm
frm, int inv_id, int film_id, int store_id) throws SQLException{
        Connection conn = getConnection();
        ResultSet rs = null;
        String query_id = "SELECT inventory_id FROM inventory WHERE inventory_id
= ?";
        String update_query = """
            UPDATE inventory SET film_id = ?, store_id = ? WHERE
inventory_id = ?""";
        try(PreparedStatement idPs =
conn.prepareStatement(query_id,

ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_READ_ONLY);
        PreparedStatement updatePS =
conn.prepareStatement(update_query,
                ResultSet.TYPE_SCROLL_SENSITIVE,
ResultSet.CONCUR_UPDATABLE))
        {
            idPs.setInt(1,inv_id);
            if(!idPs.execute()){
                String message = "Can't find inventory_id " +
inv_id;

                JOptionPane.showMessageDialog(frm, message,
                    "ERROR",JOptionPane.ERROR_MESSAGE);
            } else{
                rs = idPs.getResultSet();
                rs.next();

                //Creates a Inventory object using four-params
constructor
                Inventory obj = new Inventory(inv_id, film_id,
store_id, new Timestamp(System.currentTimeMillis()));
                updatePS.setInt(1, obj.getFilmID());
                updatePS.setInt(2, obj.getStoreID());
                updatePS.setInt(3, obj.getInventoryID());

                updatePS.executeUpdate();
                rs.close();

```

```
                updatePS.close();
                idPs.close();
                conn.close();
            }
        }catch(SQLException ex){

Logger.getLogger(InventoryForm.class.getName()).log(Level.SEVERE,
"Error updating inventory data", ex);
            String message = "Error updating inventory data: " +
ex.getMessage();
            String stackTrace =
Arrays.toString(ex.getStackTrace());
            JOptionPane.showMessageDialog(null, message + "\n\n"
+ stackTrace, "ERROR", JOptionPane.ERROR_MESSAGE);
        }catch(java.lang.NumberFormatException ex){

Logger.getLogger(InventoryForm.class.getName()).log(Level.SEVERE,
"Invalid Input", ex);
            String message = "Invalid Input: " + ex.getMessage();
            String stackTrace =
Arrays.toString(ex.getStackTrace());
            JOptionPane.showMessageDialog(null, message + "\n\n"
+ stackTrace, "ERROR", JOptionPane.ERROR_MESSAGE);
        }
    }
```

Step 2

Then in the same class, define a new method **read_inputs()**. This method reads the user inputs from an inventory form and returns a **HashMap** containing the values for the film ID, store ID, and inventory ID.

The method first retrieves the selected values from three **JComboBoxes** representing the film ID, store ID, and inventory ID. It then validates the user inputs by parsing the values into integers and checking that they are greater than zero. If any input is invalid, the method throws an **IllegalArgumentException** and displays an error message using the **JOptionPane** class.

Finally, the method creates a **HashMap** containing the film ID, store ID, and inventory ID values, and returns it to the caller.

```
    private static HashMap<String, String>
read_inputs(InventoryForm frm) {
        HashMap<String, String> input_data = new HashMap<>
();
        String store_id =
String.valueOf(frm.getJCBStoreID().getSelectedItem());

```

```
        String film_id =
String.valueOf(frm.getJCBFilmID().getSelectedItem());
        String inv_id =
String.valueOf(frm.getJCBInventoryID().getSelectedItem());

        // Validate user input
        int inv_id_int = 0;
        try {
            inv_id_int = Integer.parseInt(inv_id);
            if (inv_id_int <= 0) {
                throw new
IllegalArgumentException("Inventory ID cannot be negative
or zero");
            }
        } catch (NumberFormatException ex) {
            JOptionPane.showMessageDialog(frm, "Invalid
Inventory ID: " + inv_id,
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        } catch (IllegalArgumentException ex) {
            JOptionPane.showMessageDialog(frm,
ex.getMessage(),
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        }

        int film_id_int = 0;
        try {
            film_id_int = Integer.parseInt(film_id);
            if (film_id_int <= 0) {
                throw new IllegalArgumentException("Film ID cannot
be negative or zero");
            }
        } catch (NumberFormatException ex) {
            JOptionPane.showMessageDialog(frm, "Invalid
Film ID: " + film_id,
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        } catch (IllegalArgumentException ex) {
            JOptionPane.showMessageDialog(frm,
ex.getMessage(),
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        }

        int store_id_int = 0;
        try {
            store_id_int = Integer.parseInt(store_id);
            if (store_id_int <= 0) {
                throw new IllegalArgumentException("Store
ID cannot be negative or zero");
            }
        } catch (NumberFormatException ex) {

```

```
            JOptionPane.showMessageDialog(frm, "Invalid
Store ID: " + store_id,
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        } catch (IllegalArgumentException ex) {
            JOptionPane.showMessageDialog(frm,
ex.getMessage(),
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        }

        input_data.put("film_id", film_id);
        input_data.put("store_id", store_id);
        input_data.put("inv_id", inv_id);

        return input_data;
    }
```

Step 3

Still in the same class, define another method named **edit_actual()**. It takes an **InventoryForm** object as input, reads the user inputs by calling the read_inputs method, extracts the film ID, inventory ID, and store ID from the user inputs, updates the database by calling the **update_row_by_inventory_id()** method, and then refreshes all objects on the form by calling the refresh_controls method. If an **SQLException** is thrown during the database update, an error message dialog is displayed.

```
    private static void
edit_actual(InventoryForm frm){
        try{
            HashMap<String, String>
input_data = read_inputs(frm);
            int film_id =
Integer.parseInt(input_data.get("film_id"));
            int inv_id =
Integer.parseInt(input_data.get("inv_id"));
            int store_id =
Integer.parseInt(input_data.get("store_id"));

            update_row_by_inventory_id(frm,
inv_id, film_id, store_id);

            //Refreshes all objects on form
            refresh_controls(frm);

        }catch(SQLException ex){

JOptionPane.showMessageDialog(frm,
ex.getMessage(),
```

```
"ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }
```

Step 4	Lastly, define two new methods named **enable_controls()** and **edit_handler()**. The **enable_controls()** method takes a boolean state and an **InventoryForm** object frm. It enables or disables various controls on the form based on the value of the state parameter. If state is true, it enables all the controls, and if it's false, it disables them.

The **edit_handler()** method handles the Edit button on the form. If the text of the button is "EDIT", it changes it to "CONFIRM" and disables all the controls on the form by calling **enable_controls(false, frm)**. If the text is "CONFIRM", it changes it back to "EDIT", calls the edit_actual method to perform the actual editing, and then re-enables all the controls on the form by calling **enable_controls(true, frm)**.

```
1   |     private static void
2   | enable_controls(boolean state, InventoryForm
3   | frm){
4   |         frm.getJBFirst().setEnabled(state);
5   |         frm.getJBPrev().setEnabled(state);
6   |         frm.getJBNext().setEnabled(state);
7   |         frm.getJBLast().setEnabled(state);
8   |         frm.getJBInsert().setEnabled(state);
9   |         frm.getJBDelete().setEnabled(state);
10  |
11  | frm.getJTFAddressID().setEnabled(state);
12  |         frm.getJTFStoreID().setEnabled(state);
13  |         frm.getJTFAddress().setEnabled(state);
14  |
15  | frm.getJTFDistrict().setEnabled(state);
16  |         frm.getJTFPhone().setEnabled(state);
17  |
18  | frm.getJTFPostalCode().setEnabled(state);
19  |         frm.getJTFCity().setEnabled(state);
20  |         frm.getJTFCountry().setEnabled(state);
21  |         frm.getJTFFilmID().setEnabled(state);
22  |         frm.getJTFTitle().setEnabled(state);
23  |
24  | frm.getJTFDescription().setEnabled(state);
25  |
26  | frm.getJTFReleaseYear().setEnabled(state);
27  |
28  | frm.getJTFRentalDuration().setEnabled(state);
29  |
30  | frm.getJTFRentalRate().setEnabled(state);
31  |         frm.getJTFLength().setEnabled(state);
32  |
33  | frm.getJTFReplacementCost().setEnabled(state);
34  |
35  | frm.getJTFSpecialFeatures().setEnabled(state);
36  |
37  | frm.getJTFLastUpdate().setEnabled(state);
38  |         frm.getJTFRating().setEnabled(state);
39  |
40  | frm.getJTFLanguage().setEnabled(state);
41  |     }
```

```
42
43      public static void
44  edit_handler(InventoryForm frm){
45
46   if(frm.getJBEdit().getText().equals("EDIT")){
47
    frm.getJBEdit().setText("CONFIRM");

                // Disables controls
                enable_controls(false, frm);
            }

            else {
                frm.getJBEdit().setText("EDIT");

                // Actual editing
                edit_actual(frm);

                //Enables controls
                enable_controls(true, frm);
            }
        }
```

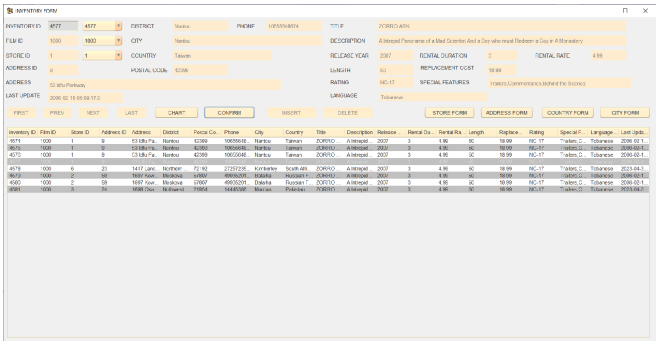
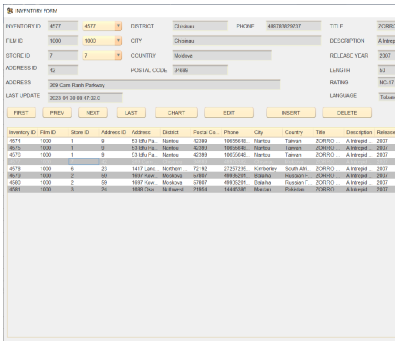


Figure 12.9 The inventory form is in editing state

Step 5	Run the project. Choos using **jcbInventoryID** co using **jcbFilmID** and **jcbStoreID** (in this case, **4577**, **film_id = 1000**, ar Then, click on EDIT but Figure 12.9. Choose **store_id** using this case, store_id = 7) CONFIRM button. The

	been saved into **inventor** in Figure 12.10.  **Figure 12.10** The edite saved into dat

INSERTING NEW RECORD
INSERTING NEW RECORD

Step 1	In **Inventory_Utils** class, define a method named **insert_row()**. This method **insert_row()** is responsible for inserting a new row into the inventory table. It first reads the input data from the form using the **read_inputs()** method, which returns a **HashMap** containing the film ID and store ID. It then parses these values into integers. The method then creates an SQL insert statement with placeholders for the film ID and store ID. It then establishes a connection to the database and prepares the SQL statement using the **prepareStatement()** method of the **Connection** class. A new **Inventory** object is created using the film ID, store ID, and the current timestamp, and the film ID and store ID are set in the prepared statement using the **setInt()** method. The SQL statement is then executed using the **executeUpdate()** method of the **PreparedStatement** class, which inserts a new row into the **inventory** table. If an exception occurs during the execution of the SQL statement, the exception message is logged using the **Logger** class, and an error message is displayed to the user using a **JOptionPane** message dialog. 1 2 `//Inserts new row into inventory table`

```
    private static void insert_row(InventoryForm frm) throws
SQLException{
        HashMap<String, String> input_data = read_inputs(frm);
        int film_id =
Integer.parseInt(input_data.get("film_id"));
        int store_id =
Integer.parseInt(input_data.get("store_id"));

        // SQL insert statement
        String sql = """
            INSERT INTO inventory(film_id, store_id) VALUES(?,
?)""";

        try(Connection conn = getConnection();
         PreparedStatement pstmt = conn.prepareStatement(sql)){

            //Creates a Inventory object three-params constructor
            Inventory obj = new Inventory(film_id, store_id, new
Timestamp(System.currentTimeMillis()));
            pstmt.setInt(1,obj.getFilmID());
            pstmt.setInt(2,obj.getStoreID());

            //Executes the sql insert statement
            pstmt.executeUpdate();
        } catch (SQLException ex) {

Logger.getLogger(InventoryForm.class.getName()).log(Level.SEVERE,
"Database error", ex);
            JOptionPane.showMessageDialog(frm, "Error: Database
error\n" + ex.getMessage());
        }
    }
```

Step 2

Still in **Inventory_Utils.java**, define **insert_actual()** and **insert_handler()** methods. The **insert_handler()** method is a handler function that is called when the user clicks the "INSERT" button on the **InventoryForm**.

If the text of the button is currently "INSERT", it changes the button text to "CONFIRM" and then disables the **jbEdit** button, disables the controls, and enables the "INSERT" button.

If the text of the button is currently "CONFIRM", it changes the button text back to "INSERT", enables the **jbEdit** button, enables the controls, disables the "INSERT" button, and then calls the **insert_actual()** method, which performs the actual insertion of a new row into the inventory table and refreshes the controls on the form.

If any **SQLExceptions** are caught during the insertion process, an error message is displayed on the form.

```
1   private static void
2  insert_actual(InventoryForm frm){
3          try{
4              insert_row(frm);
5
6              //Refreshes table and comboboxes
7              refresh_controls(frm);
8
9          }catch(SQLException ex){
10             JOptionPane.showMessageDialog(frm,
11 ex.getMessage(),
12
13  "ERROR",JOptionPane.ERROR_MESSAGE);
14         }
15     }
16
17     public static void
18 insert_handler(InventoryForm frm){
19
20  if(frm.getJBInsert().getText().equals("INSERT")
21 ){
22
23 frm.getJBInsert().setText("CONFIRM");
24
25             //Disables jbEdit
26             frm.getJBEdit().setEnabled(false);
27
28             // Disables controls
29             enable_controls(false, frm);
30
31 frm.getJCBInventoryID().setEnabled(false);
32
33             // Enables
34             frm.getJBInsert().setEnabled(true);
35         }
36
37         else {
38             frm.getJBInsert().setText("INSERT");
39
40             // Actual insertion
41             insert_actual(frm);
42
               //Enables jbEdit
               frm.getJBEdit().setEnabled(true);

               //Enables controls
               enable_controls(true, frm);

   frm.getJCBInventoryID().setEnabled(true);
           }
```

```
    }
```

Step 3

In **InventoryForm.java**, double click on INSERT button to create its event listener:

```
    private void
jbInsertActionPerformed(java.awt.event.ActionEvent
evt) {
        Inventory_Utils.insert_handler(this);
    }
```

Step 4

Run the project. Click on INSERT button. You will see the state of inventory form when insertion is in progress as shown in Figure 12.11.

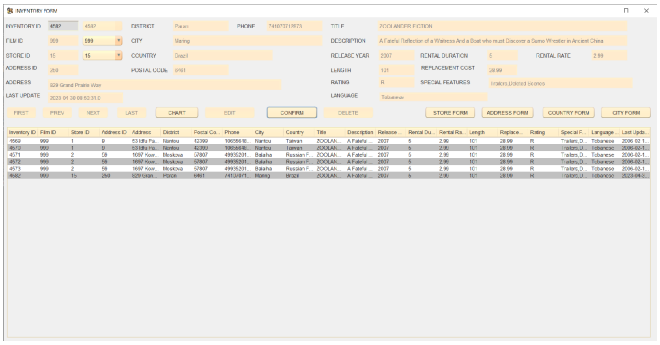


Figure 12.11 When user clicks on INSERT button, the inventory form will be in state of insertion

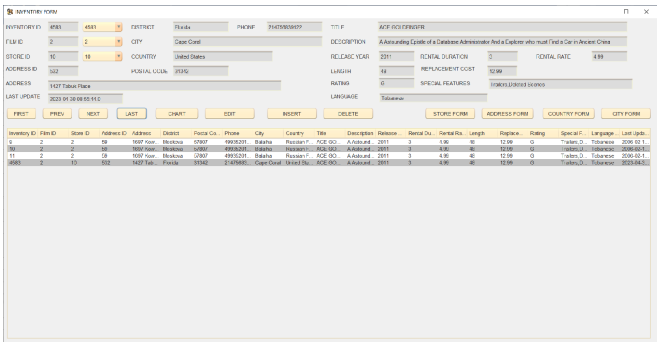


Figure 12.12 The new data had been saved into **inventory** table

Then, choose **film_id** using **jcbFilmID** (in this case, **film_id = 2**) and choose **store_id** using **jcbStoreID** (in this case, **store_id = 10**). Then, click CONFIRM button to save the new record into **inventory** table as shown in Figure 12.12.

DELETING RECORD
DELETING RECORD

Step 1

Then in **Inventory_Utils** class, define **delete_handler()** method. Thi handles the delete operation for a selected row in the **inventory** tab prompts the user with a confirmation dialog to ensure that the user want the selected row. If the user confirms the deletion, the method then ex SQL DELETE statement to delete the corresponding row from the inven using the inventory ID selected in the form's combobox. The method also the table and comboboxes to reflect the updated data in the **inventory** t exception is caught during the SQL operation, an error message dialog is d

```
1   public static void delete_handler(InventoryForm frm){
2       int dialogButton = JOptionPane.YES_NO_OPTION;
3       int inv_id =
4   Integer.parseInt(String.valueOf(frm.getJCBInventoryID().getSelected
5
6       String message = String.format("Are you sure you want to de
7   row Inventory ID: %d)", inv_id);
8       int answer = JOptionPane.showConfirmDialog(frm, message, "D
9   ROW OF DATA", dialogButton);
10
11      if(answer == JOptionPane.YES_OPTION){
12          String query = """
13              DELETE FROM inventory WHERE inventory_id = ?""";
14          try(Connection conn = getConnection();
15              PreparedStatement ps = conn.prepareStatement(query)
16              // Use PreparedStatement to avoid SQL injection att
17              ps.setInt(1, inv_id);
18              ps.executeUpdate();
19
20              // Refresh table and comboboxes
21              refresh_controls(frm);
22
23          } catch (SQLException ex){
24              JOptionPane.showMessageDialog(frm, ex.getMessage(),
25                  "ERROR",JOptionPane.ERROR_MESSAGE);
26          }
27      }
28  }
```

Step 2

In **InventoryForm.java**, double click on DELETE button to generate listener:

```
1   private void
2   jbDeleteActionPerformed(java.awt.event.ActionEvent
3   evt) {
```

```
            Inventory_Utils.delete_handler(this);
        }
```

Step 3	Run the project. Choose **inventory_id** using **jcbInventoryID** combobox. T on DELETE button. The corresponding row of data had been deleted from

PLOTTING CHART
PLOTTING CHART

Step 1	Create a new **JFrame** and save it as **Charts_Inventory.java**.
Step 2	In **Charts_Inventory.java**, add three **JPanels** and set their correspond **Variable Name** as **jPanel1**, **jPanel2**, **jPanel3**, **jPanel4**, **jPanel5**, a **jPanel6**. Then, add getter method for each object as follows:

```
1     //Getter method for jPanel1
2     public JPanel getJPanel1(){
3         return this.jPanel1;
4     }
5
6     //Getter method for jPanel2
7     public JPanel getJPanel2(){
8         return this.jPanel2;
9     }
10
11    //Getter method for jPanel3
12    public JPanel getJPanel3(){
13        return this.jPanel3;
14    }
15
16    //Getter method for jPanel4
17    public JPanel getJPanel4(){
18        return this.jPanel4;
19    }
20
21    //Getter method for jPanel5
22    public JPanel getJPanel5(){
23        return this.jPanel5;
24    }
25
26
```

```
27      //Getter method for jPanel6
28      public JPanel getJPanel6(){
29          return this.jPanel6;
        }
```

Step 3

In **Inventory_Utils** class, define six new methods. The first three metho **draw_pie_chart_inventory_by_year()**, **draw_pie_chart_inventory_by_rating()**, a **draw_pie_chart_inventory_by_language()**, all create pie charts based different categories of inventory data. The **create_pie_dataset()** method called in each of these methods to create a **DefaultPieDataset** object, wh is then passed into the **draw_piechart_with_dataset()** method along w some descriptive strings for the chart title and labels.

The next three methods, **draw_bar_chart_inventory_by_duration** **draw_bar_chart_inventory_by_city()**, a **draw_bar_chart_inventory_by_country()**, all create bar charts based different categories of inventory data. The **create_bar_dataset()** method called in each of these methods to create a **DefaultCategoryDataset** obje which is then passed into the **draw_barchart_with_dataset()** method alo with some descriptive strings for the chart title and labels.

```
1       private static void
2   draw_pie_chart_inventory_by_year(Charts_Inventory frm, JPanel jp){
3           jp.setPreferredSize(new Dimension(jp.getWidth(),
4   jp.getHeight()));
5           DefaultPieDataset dataset =
6   create_pie_dataset(Query_Inventory.get_sql_inventory_year_dist(),
7   "Number", "year");
8
9           //Draws piechart inventory distribution by film release yea
10          draw_piechart_with_dataset(frm, jp, dataset, "TOP 10
11  INVENTORY DISTRIBUTION BY RELEASE YEAR");
12      }
13
14      private static void
15  draw_pie_chart_inventory_by_rating(Charts_Inventory frm, JPanel jp)
16          jp.setPreferredSize(new Dimension(jp.getWidth(),
17  jp.getHeight()));
18          DefaultPieDataset dataset =
19  create_pie_dataset(Query_Inventory.get_sql_inventory_rating_dist(),
20  "Number", "rating");
21
22          //Draws piechart inventory distribution by film rating
23          draw_piechart_with_dataset(frm, jp, dataset, "TOP 10
24  INVENTORY DISTRIBUTION BY FILM RATING");
25      }
26
27
```

```
    private static void
draw_pie_chart_inventory_by_language(Charts_Inventory frm, JPanel j
{
        jp.setPreferredSize(new Dimension(jp.getWidth(),
jp.getHeight()));
        DefaultPieDataset dataset =
create_pie_dataset(Query_Inventory.get_sql_inventory_lang_dist(),
"Number", "l1.name");

        //Draws piechart inventory distribution by film language
        draw_piechart_with_dataset(frm, jp, dataset, "TOP 10
INVENTORY DISTRIBUTION BY FILM LANGUAGE");
    }

    private static void
draw_bar_chart_inventory_by_duration(Charts_Inventory frm, JPanel j
{
        jp.setPreferredSize(new Dimension(jp.getWidth(),
jp.getHeight()));

        DefaultCategoryDataset dataset =
create_bar_dataset(Query_Inventory.get_sql_inventory_duration_dist(
"Number", "duration");

        //Draws barchart store distribution by rental duration
        draw_barchart_with_dataset(frm, jp, dataset, "THE 10
INVENTORY DISTRIBUTION BY RENTAL DURATION", "RENTAL DURATION",
"NUMBER");
    }

    private static void
draw_bar_chart_inventory_by_city(Charts_Inventory frm, JPanel jp){
        jp.setPreferredSize(new Dimension(jp.getWidth(),
jp.getHeight()));

        DefaultCategoryDataset dataset =
create_bar_dataset(Query_Inventory.get_sql_inventory_city_dist(),
"Number", "city");

        //Draws barchart store distribution by city
        draw_barchart_with_dataset(frm, jp, dataset, "THE 10
INVENTORY DISTRIBUTION BY CITY", "CITY", "NUMBER");
    }

    private static void
draw_bar_chart_inventory_by_country(Charts_Inventory frm, JPanel jp
        jp.setPreferredSize(new Dimension(jp.getWidth(),
jp.getHeight()));

        DefaultCategoryDataset dataset =
create_bar_dataset(Query_Inventory.get_sql_inventory_country_dist()
"Number", "country");
```

```
        //Draws barchart store distribution by country
        draw_barchart_with_dataset(frm, jp, dataset, "THE 10
INVENTORY DISTRIBUTION BY COUNTRY", "COUNTRY", "NUMBER");
    }
```

Step 4

In **Inventory_Utils** class, define a new method named **jbchart_handler()**

```
    public static void
jbchart_handler(Charts_Inventory frm){
        //Draws piechart inventory
distribution by film release year

draw_pie_chart_inventory_by_year(frm,
frm.getJPanel1());

        //Draws piechart inventory
distribution by film rating

draw_pie_chart_inventory_by_rating(frm,
frm.getJPanel2());

        //Draws piechart inventory
distribution by film language

draw_pie_chart_inventory_by_language(frm,
frm.getJPanel3());

        //Draws barchart store
distribution by rental duration

draw_bar_chart_inventory_by_duration(frm,
frm.getJPanel4());

        //Draws barchart store
distribution by city

draw_bar_chart_inventory_by_city(frm,
frm.getJPanel5());

        //Draws barchart store
distribution by country

draw_bar_chart_inventory_by_country(frm,
frm.getJPanel6());
    }
```

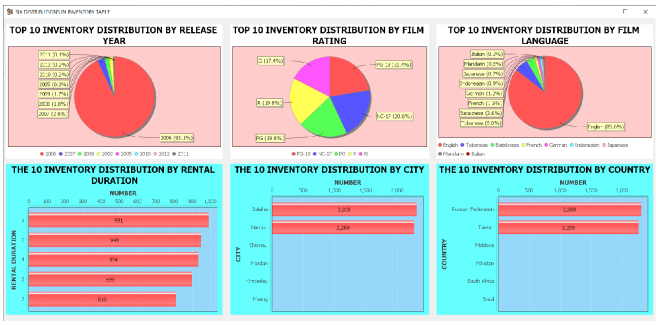


Figure 12.13 The top 10 inventory distribution by release year, the top 1 inventory distribution by film rating, the top 10 inventory distribution by fi language, the top 10 inventory distribution by film rental duration, the top inventory distribution by city, and the top 10 inventory distribution by country

Step 5

In **InventoryForm**, double click on **jbChart** button to define its ev listener:

```
    private void
jbChartActionPerformed(java.awt.event.ActionEvent
evt) {
        Charts_Inventory frm1 = new
Charts_Inventory();
        frm1.setLocationRelativeTo(null);
        frm1.setTitle("SIX DISTRIBUTIONS IN
INVENTORY TABLE");
        frm1.setVisible(true);
        Inventory_Utils.jbchart_handler(frm1);
    }
```

Step 6

Run the project. Click on CHART button on the form. You will see the charts displayed on the panels as shown in Figure 12.13.

This is the full version of **Inventory_Utils.java**:

```
package sakila;
import java.awt.Dimension;
import java.util.logging.Level;
import java.util.logging.Logger;
import java.sql.*;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashMap;
import java.util.Objects;
import javax.swing.JComboBox;
```

```java
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.event.TableModelEvent;
import javax.swing.table.DefaultTableModel;
import javax.swing.table.TableModel;
import org.jfree.data.category.DefaultCategoryDataset;
import org.jfree.data.general.DefaultPieDataset;
import static sakila.Utility.create_bar_dataset;
import static sakila.Utility.draw_barchart_with_dataset;

public class Inventory_Utils extends Utility{
    public static final int FIRST_INDEX = 0;
    public static final int INVALID_INDEX = -1;

    private static int currentIndex = FIRST_INDEX;
    private static final String SQL_ID = Query_Inventory.get_sql_id();

    //Creates inventory table
    public static void create_inventory_table() {
        try (Connection conn = getConnection()) {
            Statement stmt = conn.createStatement();
            stmt.addBatch(Query_Inventory.get_sql_inventory());
            stmt.executeBatch();

            String message = String.format("Successfully creates inventory ta
            JOptionPane.showMessageDialog(null, message,
            "INFORMATION",JOptionPane.INFORMATION_MESSAGE);

        } catch (SQLException ex) {
            JOptionPane.showMessageDialog(null, ex.getMessage(),
                "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    //Populates inventory table with some rows of data
    public static void populate_inventory_table(){
        try(Connection conn = getConnection()){
            String sql = """
                INSERT INTO inventory(inventory_id, film_id, store_id, last_u
                VALUES(?, ?, ?, ?)""";

            //Creates a new Inventory class with default constructor
            PreparedStatement ps1 = conn.prepareStatement(sql);
            Inventory obj1 = new Inventory();
            ps1.setInt(1,obj1.getInventoryID());
            ps1.setInt(2,obj1.getFilmID());
            ps1.setInt(3,obj1.getStoreID());
            ps1.setTimestamp(4,obj1.getLastUpdate());

            // Creates a new Inventory class with four-params constructor
            PreparedStatement ps2 = conn.prepareStatement(sql);
            Inventory obj2 = new Inventory(2, 2, 2, new
Timestamp(System.currentTimeMillis()));
```

```
            ps2.setInt(1,obj2.getInventoryID());
            ps2.setInt(2,obj2.getFilmID());
            ps2.setInt(3,obj2.getStoreID());
            ps2.setTimestamp(4,obj2.getLastUpdate());

            ps1.executeUpdate();
            ps2.executeUpdate();

        }catch(SQLException ex){
            JOptionPane.showMessageDialog(null, ex.getMessage(),
                "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    //Reads the content of joined inventory, film, language, store, address,
country tables
    public static void read_inventory_table(){
        try(Connection conn = getConnection()){
            Statement stmt = conn.createStatement();
            ResultSet rs =
stmt.executeQuery(Query_Inventory.get_sql_inventory_joint());

            while(rs.next()){
                int inv_id = rs.getInt("inventory_id");
                int film_id = rs.getInt("film_id");
                int store_id = rs.getInt("store_id");
                int add_id = rs.getInt("address_id");
                String add = rs.getString("address");
                String dist = rs.getString("district");
                String post = rs.getString("postal_code");
                String ph = rs.getString("phone");
                String city = rs.getString("city");
                String country = rs.getString("country");
                String title = rs.getString("title");
                String description = rs.getString("description");
                int year = rs.getInt("release_year");
                int duration = rs.getInt("rental_duration");
                double rate = rs.getDouble("rental_rate");
                int length = rs.getInt("length");
                double cost = rs.getDouble("replacement_cost");
                String rating = rs.getString("rating");
                String features = rs.getString("special_features");
                String lang_name = rs.getString("language_name");

                Timestamp lu = rs.getTimestamp("last_update");

                //Creates a Inventory object using twenty-one-params constru
                Inventory obj = new Inventory(inv_id, film_id, store_id, add_
                    dist, post, ph, city, country, title, description, year,
rate,
                    length, cost, rating, features, lang_name, lu);
                System.out.println(obj);
            }
```

```
            rs.close();
            stmt.close();

        }catch(SQLException ex){
            JOptionPane.showMessageDialog(null, ex.getMessage(),
                "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    private static ArrayList<Inventory> get_inventory_list(InventoryForm frm,
sql, String item){
        ArrayList<Inventory> list = new ArrayList<>();

        try(Connection conn = getConnection();
            PreparedStatement ps = conn.prepareStatement(sql)){
            if (item.equalsIgnoreCase("none")==false) {
                 ps.setString(1,item);
            }
            ResultSet rs = ps.executeQuery();

            Inventory obj;
            while(rs.next()){
                //Using twenty-one-params constructor
                obj = new Inventory(rs.getInt("inventory_id"),
                    rs.getInt("film_id"),
                    rs.getInt("store_id"),
                    rs.getInt("address_id"),
                    rs.getString("address"),
                    rs.getString("district"),
                    rs.getString("postal_code"),
                    rs.getString("phone"),
                    rs.getString("city"),
                    rs.getString("country"),
                    rs.getString("title"),
                    rs.getString("description"),
                    rs.getInt("release_year"),
                    rs.getInt("rental_duration"),
                    rs.getDouble("rental_rate"),
                    rs.getInt("length"),
                    rs.getDouble("replacement_cost"),
                    rs.getString("rating"),
                    rs.getString("special_features"),
                    rs.getString("language_name"),
                    rs.getTimestamp("last_update"));

                list.add(obj);
            }
        }catch (SQLException ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
                "ERROR",JOptionPane.ERROR_MESSAGE);
        }
        return list;
    }
```

```
    private static void show_table_inventory(InventoryForm frm, ArrayList<Inv
list) throws SQLException{
        DefaultTableModel model = new DefaultTableModel(0,0);

        String header[] = {"Inventory ID", "Film ID", "Store ID", "Address ID
"Address",
            "District", "Postal Code", "Phone", "City", "Country",
            "Title", "Description", "Release Year", "Rental Duration",
            "Rental Rate", "Length", "Replacement Cost", "Rating", "Special F
            "Language Name", "Last Update"};

        model.setColumnIdentifiers(set_column_header(frm.getJTInventory(), he
        frm.getJTInventory().setModel(model);

        Object[] row = new Object[21];

        for(int i=0; i<list.size(); i++){
            row[0] = list.get(i).getInventoryID();
            row[1] = list.get(i).getFilmID();
            row[2] = list.get(i).getStoreID();
            row[3] = list.get(i).getAddressID();
            row[4] = list.get(i).getAddress();
            row[5] = list.get(i).getDistrict();
            row[6] = list.get(i).getPostalCode();
            row[7] = list.get(i).getPhone();
            row[8] = list.get(i).getCity();
            row[9] = list.get(i).getCountry();
            row[10] = list.get(i).getTitle();
            row[11] = list.get(i).getDescription();
            row[12] = list.get(i).getReleaseYear();
            row[13] = list.get(i).getRentalDuration();
            row[14] = list.get(i).getRentalRate();
            row[15] = list.get(i).getLength();
            row[16] = list.get(i).getReplacementCost();
            row[17] = list.get(i).getRating();
            row[18] = list.get(i).getSpecialFeatures();
            row[19] = list.get(i).getLanguageName();
            row[20] = list.get(i).getLastUpdate();

            model.addRow(row);
        }
    }

    public static void refresh_controls(InventoryForm frm){
        frm.setLocationRelativeTo(null);
        frm.setTitle("INVENTORY FORM");

        //Shows the content of inventory table and populates combobox
        try{
            //Makes alternating color for table rows
            table_renderer(frm.getJTInventory());

            //Populates table
```

```
            ArrayList<Inventory> list = get_inventory_list(frm,
Query_Inventory.get_sql_inventory_joint() + " ORDER BY inventory_id", "none");
            show_table_inventory(frm, list);

            //Populates jcbInventoryID
            String sql_inv_id = "SELECT inventory_id FROM inventory ORDER BY
inventory_id";
            populate_combobox(sql_inv_id, frm.getJCBInventoryID(), frm);

            //Populates jcbFilmID
            String sql_film_id = "SELECT film_id FROM film ORDER BY film_id";
            populate_combobox(sql_film_id, frm.getJCBFilmID(), frm);

            //Populates jcbStoreID
            String sql_store_id = "SELECT store_id FROM store ORDER BY store_
            populate_combobox(sql_store_id, frm.getJCBStoreID(), frm);

        }catch (SQLException ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
            "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    private static void clear_store_controls(InventoryForm frm){
        frm.getJTFAddressID().setText("");
        frm.getJTFAddress().setText("");
        frm.getJTFDistrict().setText("");
        frm.getJTFPhone().setText("");
        frm.getJTFPostalCode().setText("");
        frm.getJTFCity().setText("");
        frm.getJTFCountry().setText("");
    }

   //Displays store, address, country, and city data result row by row
    private static <T> void display_store_data(InventoryForm frm, String sql,
        try(Connection conn = getConnection()){
            PreparedStatement ps = conn.prepareStatement(sql);
            ps.setObject(1,item);
            ResultSet rs = ps.executeQuery();

            if (!rs.next()) {
                // no row found, clear the form fields
                clear_store_controls(frm);
                return;
            }

            do{
                frm.getJTFStoreID().setText(String.valueOf(rs.getInt("store_i

frm.getJTFAddressID().setText(String.valueOf(rs.getInt("address_id")));
                frm.getJTFAddress().setText(rs.getString("address"));
                frm.getJTFDistrict().setText(rs.getString("district"));
                frm.getJTFPhone().setText(rs.getString("phone"));
```

```
                frm.getJTFPostalCode().setText(rs.getString("postal_code"));
                frm.getJTFCity().setText(rs.getString("city"));
                frm.getJTFCountry().setText(rs.getString("country"));

                // Determines item selected from jcbStoreID
                find_combo_value_selected(frm.getJCBStoreID(), rs.getInt("sto

            }while(rs.next());

            rs.close();
            ps.close();
        }catch(SQLException ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    public static void jcbStore_handler(InventoryForm frm) {
        Object item = frm.getJCBStoreID().getSelectedItem();
        display_store_data(frm, Query_Store.get_sql_store_joint() + " WHERE s
?", item);
    }

    private static void clear_film_controls(InventoryForm frm){
        frm.getJTFFilmID().setText("");
        frm.getJTFTitle().setText("");
        frm.getJTFDescription().setText("");
        frm.getJTFReleaseYear().setText("");
        frm.getJTFRentalDuration().setText("");
        frm.getJTFRentalRate().setText("");
        frm.getJTFLength().setText("");
        frm.getJTFReplacementCost().setText("");
        frm.getJTFSpecialFeatures().setText("");
        frm.getJTFLastUpdate().setText("");
    }

   //Displays joined film and language data result row by row
    private static <T> void display_film_data(InventoryForm frm, String sql,
        try(Connection conn = getConnection()){
            PreparedStatement ps = conn.prepareStatement(sql);
            ps.setObject(1,item);
            ResultSet rs = ps.executeQuery();

            if (!rs.next()) {
                // no row found, clear the form fields
                clear_film_controls(frm);
                return;
            }

            do{
                frm.getJTFFilmID().setText(String.valueOf(rs.getInt("film_id'
                frm.getJTFTitle().setText(rs.getString("title"));
                frm.getJTFDescription().setText(rs.getString("description"));
```

```
frm.getJTFReleaseYear().setText(String.valueOf(rs.getInt("release_year")));

frm.getJTFRentalDuration().setText(String.valueOf(rs.getInt("rental_duration"

frm.getJTFRentalRate().setText(String.valueOf(rs.getDouble("rental_rate")));
                frm.getJTFLength().setText(String.valueOf(rs.getInt("length")

frm.getJTFReplacementCost().setText(String.valueOf(rs.getDouble("replacement_
                frm.getJTFSpecialFeatures().setText(rs.getString("special_fea
                frm.getJTFLanguage().setText(rs.getString("language_name"));
                frm.getJTFRating().setText(rs.getString("rating"));

                // Determines item selected from jcbFilmID
                find_combo_value_selected(frm.getJCBFilmID(), rs.getInt("filn

            }while(rs.next());

            rs.close();
            ps.close();
        }catch(SQLException ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    public static void jcbFilm_handler(InventoryForm frm) throws SQLExceptior
        Object item = frm.getJCBFilmID().getSelectedItem();
        display_film_data(frm, Query_Film.get_sql_film_joint() + " WHERE filn
item);

        //Filters table
        String sql = Query_Inventory.get_sql_inventory_joint() + " WHERE i.fi
ORDER BY i.inventory_id";
        ArrayList<Inventory> list = get_inventory_list(frm, sql,
String.valueOf(item));
        show_table_inventory(frm, list);
    }

   //Displays inventory, store, address, country, city, film, and language da
row by row
    private static <T> void display_inventory_data(InventoryForm frm, String
item){
        try(Connection conn = getConnection()){
            PreparedStatement ps = conn.prepareStatement(sql);
            ps.setObject(1,item);
            ResultSet rs = ps.executeQuery();

            if (!rs.next()) {
                // no row found, clear the form fields
                clear_store_controls(frm);
                clear_film_controls(frm);
                frm.getJTFLastUpdate().setText("");
```

```
                return;
            }

            do{

frm.getJTFInventoryID().setText(String.valueOf(rs.getInt("inventory_id")));

frm.getJTFLastUpdate().setText(String.valueOf(rs.getTimestamp("last_update"))

                // Determines item selected from jcbInventoryID
                find_combo_value_selected(frm.getJCBInventoryID(),
rs.getInt("inventory_id"));

                // Determines item selected from jcbStoreID
                find_combo_value_selected(frm.getJCBStoreID(), rs.getInt("sto
```

```
                // Determines item selected from jcbFilmID
                find_combo_value_selected(frm.getJCBFilmID(),
rs.getInt("film_id"));

            }while(rs.next());

            rs.close();
            ps.close();
        }catch(SQLException ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    public static void jcbInventory_handler(InventoryForm frm) {
        Object item = frm.getJCBInventoryID().getSelectedItem();
        display_inventory_data(frm,
Query_Inventory.get_sql_inventory_joint() + " WHERE inventory_id = ?",
item);
    }

    public static void show_first_row(InventoryForm frm){
        String item =
String.valueOf(frm.getJCBInventoryID().getItemAt(FIRST_INDEX));
        display_inventory_data(frm, SQL_ID, item);
        currentIndex = FIRST_INDEX;
    }

    public static void show_last_row(InventoryForm frm){
        int endIndex = frm.getJCBInventoryID().getItemCount() - 1;
        String item =
String.valueOf(frm.getJCBInventoryID().getItemAt(endIndex));
        display_inventory_data(frm, SQL_ID, item);
        currentIndex = endIndex;
    }

    public static void show_prev_row(InventoryForm frm){
        currentIndex--;
        if(currentIndex < FIRST_INDEX){
            currentIndex = FIRST_INDEX;
            return;
        }
        String item =
String.valueOf(frm.getJCBInventoryID().getItemAt(currentIndex));
        display_inventory_data(frm, SQL_ID, item);
    }

    public static void show_next_row(InventoryForm frm){
        int endIndex = frm.getJCBInventoryID().getItemCount() - 1;
        currentIndex++;
        if(currentIndex > endIndex){
            currentIndex = endIndex;
```

```
            return;
        }
        String item =
String.valueOf(frm.getJCBInventoryID().getItemAt(currentIndex));
        display_inventory_data(frm, SQL_ID, item);
    }

    public static void mouse_pressed_handler(InventoryForm frm) {
        Objects.requireNonNull(frm, "frm must not be null");

        int selectedIndex = frm.getJTInventory().getSelectedRow();
        if (selectedIndex == -1) {
            JOptionPane.showMessageDialog(frm, "Please select a row to view
its data.",
                "No row selected", JOptionPane.INFORMATION_MESSAGE);
            return;
        }

        try (Connection conn = getConnection()) {
            String id =
String.valueOf(frm.getJTInventory().getModel().getValueAt(selectedIndex,
0));

            // Displays inventory data
            display_inventory_data(frm, SQL_ID, id);

        } catch (SQLException ex) {

Logger.getLogger(InventoryForm.class.getName()).log(Level.SEVERE, "Error
displaying inventory data", ex);
            String message = "Error displaying inventory data: " +
ex.getMessage();
            String stackTrace = Arrays.toString(ex.getStackTrace());
            JOptionPane.showMessageDialog(frm, message + "\n\n" +
stackTrace, "ERROR", JOptionPane.ERROR_MESSAGE);
        }
    }

    //Updates row of data in inventory tabel by inventory_id
    public static void update_row_by_inventory_id(InventoryForm frm, int
inv_id, int film_id, int store_id) throws SQLException{
        Connection conn = getConnection();
        ResultSet rs = null;
        String query_id = "SELECT inventory_id FROM inventory WHERE
inventory_id = ?";
        String update_query = """
            UPDATE inventory SET film_id = ?, store_id = ? WHERE
inventory_id = ?""";
        try(PreparedStatement idPs = conn.prepareStatement(query_id,

ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_READ_ONLY);
        PreparedStatement updatePS = conn.prepareStatement(update_query,
```

```
                ResultSet.TYPE_SCROLL_SENSITIVE,
ResultSet.CONCUR_UPDATABLE))
        {
            idPs.setInt(1,inv_id);
            if(!idPs.execute()){
                String message = "Can't find inventory_id " + inv_id;

                JOptionPane.showMessageDialog(frm, message,
                    "ERROR",JOptionPane.ERROR_MESSAGE);
            } else{
                rs = idPs.getResultSet();
                rs.next();

                //Creates a Inventory object using four-params constructor
                Inventory obj = new Inventory(inv_id, film_id, store_id, new
Timestamp(System.currentTimeMillis()));
                updatePS.setInt(1, obj.getFilmID());
                updatePS.setInt(2, obj.getStoreID());
                updatePS.setInt(3, obj.getInventoryID());

                updatePS.executeUpdate();
                rs.close();
                updatePS.close();
                idPs.close();
                conn.close();
            }
        }catch(SQLException ex){

Logger.getLogger(InventoryForm.class.getName()).log(Level.SEVERE, "Error
updating inventory data", ex);
            String message = "Error updating inventory data: " +
ex.getMessage();
            String stackTrace = Arrays.toString(ex.getStackTrace());
            JOptionPane.showMessageDialog(null, message + "\n\n" +
stackTrace, "ERROR", JOptionPane.ERROR_MESSAGE);
        }catch(java.lang.NumberFormatException ex){

Logger.getLogger(InventoryForm.class.getName()).log(Level.SEVERE, "Invalid
Input", ex);
            String message = "Invalid Input: " + ex.getMessage();
            String stackTrace = Arrays.toString(ex.getStackTrace());
            JOptionPane.showMessageDialog(null, message + "\n\n" +
stackTrace, "ERROR", JOptionPane.ERROR_MESSAGE);
        }
    }

    private static HashMap<String, String> read_inputs(InventoryForm frm) {
        HashMap<String, String> input_data = new HashMap<>();
        String store_id =
String.valueOf(frm.getJCBStoreID().getSelectedItem());
        String film_id =
String.valueOf(frm.getJCBFilmID().getSelectedItem());
        String inv_id =
String.valueOf(frm.getJCBInventoryID().getSelectedItem());
```

```
        // Validate user input
        int inv_id_int = 0;
        try {
            inv_id_int = Integer.parseInt(inv_id);
            if (inv_id_int <= 0) {
                throw new IllegalArgumentException("Inventory ID cannot be
negative or zero");
            }
        } catch (NumberFormatException ex) {
            JOptionPane.showMessageDialog(frm, "Invalid Inventory ID: " +
inv_id,
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        } catch (IllegalArgumentException ex) {
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        }

        int film_id_int = 0;
        try {
            film_id_int = Integer.parseInt(film_id);
            if (film_id_int <= 0) {
                throw new IllegalArgumentException("Film ID cannot be
negative or zero");
            }
        } catch (NumberFormatException ex) {
            JOptionPane.showMessageDialog(frm, "Invalid Film ID: " +
film_id,
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        } catch (IllegalArgumentException ex) {
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        }

        int store_id_int = 0;
        try {
            store_id_int = Integer.parseInt(store_id);
            if (store_id_int <= 0) {
                throw new IllegalArgumentException("Store ID cannot be
negative or zero");
            }
        } catch (NumberFormatException ex) {
            JOptionPane.showMessageDialog(frm, "Invalid Store ID: " +
store_id,
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        } catch (IllegalArgumentException ex) {
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
            "Error", JOptionPane.ERROR_MESSAGE);
```

```
            throw ex;
        }

        input_data.put("film_id", film_id);
        input_data.put("store_id", store_id);
        input_data.put("inv_id", inv_id);

        return input_data;
    }

    private static void edit_actual(InventoryForm frm){
        try{
            HashMap<String, String> input_data = read_inputs(frm);
            int film_id = Integer.parseInt(input_data.get("film_id"));
            int inv_id = Integer.parseInt(input_data.get("inv_id"));
            int store_id = Integer.parseInt(input_data.get("store_id"));

            update_row_by_inventory_id(frm, inv_id, film_id, store_id);

            //Refreshes all objects on form
            refresh_controls(frm);

        }catch(SQLException ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    private static void enable_controls(boolean state, InventoryForm frm){
        frm.getJBFirst().setEnabled(state);
        frm.getJBPrev().setEnabled(state);
        frm.getJBNext().setEnabled(state);
        frm.getJBLast().setEnabled(state);
        frm.getJBInsert().setEnabled(state);
        frm.getJBDelete().setEnabled(state);
        frm.getJTFAddressID().setEnabled(state);
        frm.getJTFStoreID().setEnabled(state);
        frm.getJTFAddress().setEnabled(state);
        frm.getJTFDistrict().setEnabled(state);
        frm.getJTFPhone().setEnabled(state);
        frm.getJTFPostalCode().setEnabled(state);
        frm.getJTFCity().setEnabled(state);
        frm.getJTFCountry().setEnabled(state);
        frm.getJTFFilmID().setEnabled(state);
        frm.getJTFTitle().setEnabled(state);
        frm.getJTFDescription().setEnabled(state);
        frm.getJTFReleaseYear().setEnabled(state);
        frm.getJTFRentalDuration().setEnabled(state);
        frm.getJTFRentalRate().setEnabled(state);
        frm.getJTFLength().setEnabled(state);
        frm.getJTFReplacementCost().setEnabled(state);
        frm.getJTFSpecialFeatures().setEnabled(state);
        frm.getJTFLastUpdate().setEnabled(state);
        frm.getJTFRating().setEnabled(state);
```

```
        frm.getJTFLanguage().setEnabled(state);
    }

    public static void edit_handler(InventoryForm frm){
        if(frm.getJBEdit().getText().equals("EDIT")){
            frm.getJBEdit().setText("CONFIRM");

            // Disables controls
            enable_controls(false, frm);
        }

        else {
            frm.getJBEdit().setText("EDIT");

            // Actual editing
            edit_actual(frm);

            //Enables controls
            enable_controls(true, frm);
        }
    }

    //Inserts new row into inventory table
    private static void insert_row(InventoryForm frm) throws SQLException{
        HashMap<String, String> input_data = read_inputs(frm);
        int film_id = Integer.parseInt(input_data.get("film_id"));
        int store_id = Integer.parseInt(input_data.get("store_id"));

        // SQL insert statement
        String sql = """
            INSERT INTO inventory(film_id, store_id) VALUES(?, ?)""";

        try(Connection conn = getConnection();
         PreparedStatement pstmt = conn.prepareStatement(sql)){

            //Creates a Inventory object three-params constructor
            Inventory obj = new Inventory(film_id, store_id, new
Timestamp(System.currentTimeMillis()));
            pstmt.setInt(1,obj.getFilmID());
            pstmt.setInt(2,obj.getStoreID());

            //Executes the sql insert statement
            pstmt.executeUpdate();
        } catch (SQLException ex) {

Logger.getLogger(InventoryForm.class.getName()).log(Level.SEVERE, "Database
error", ex);
            JOptionPane.showMessageDialog(frm, "Error: Database error\n" +
ex.getMessage());
        }
    }

    private static void insert_actual(InventoryForm frm){
```

```
        try{
            insert_row(frm);

            //Refreshes table and comboboxes
            refresh_controls(frm);

        }catch(SQLException ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    public static void insert_handler(InventoryForm frm){
        if(frm.getJBInsert().getText().equals("INSERT")){
            frm.getJBInsert().setText("CONFIRM");

            //Disables jbEdit
            frm.getJBEdit().setEnabled(false);

            // Disables controls
            enable_controls(false, frm);
            frm.getJCBInventoryID().setEnabled(false);

            // Enables
            frm.getJBInsert().setEnabled(true);
        }

        else {
            frm.getJBInsert().setText("INSERT");

            // Actual insertion
            insert_actual(frm);

            //Enables jbEdit
            frm.getJBEdit().setEnabled(true);

            //Enables controls
            enable_controls(true, frm);
            frm.getJCBInventoryID().setEnabled(true);
        }
    }

    public static void delete_handler(InventoryForm frm){
        int dialogButton = JOptionPane.YES_NO_OPTION;
        int inv_id =
Integer.parseInt(String.valueOf(frm.getJCBInventoryID().getSelectedItem()));

        String message = String.format("Are you sure you want to delete the
row Inventory ID: %d)", inv_id);
        int answer = JOptionPane.showConfirmDialog(frm, message, "DELETING
ROW OF DATA", dialogButton);

        if(answer == JOptionPane.YES_OPTION){
```

```
            String query = """
                DELETE FROM inventory WHERE inventory_id = ?""";
            try(Connection conn = getConnection();
                PreparedStatement ps = conn.prepareStatement(query)){
                // Use PreparedStatement to avoid SQL injection attacks
                ps.setInt(1, inv_id);
                ps.executeUpdate();

                // Refresh table and comboboxes
                refresh_controls(frm);

            } catch (SQLException ex){
                JOptionPane.showMessageDialog(frm, ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
            }
        }
    }

    private static void draw_pie_chart_inventory_by_year(Charts_Inventory
frm, JPanel jp){
        jp.setPreferredSize(new Dimension(jp.getWidth(), jp.getHeight()));
        DefaultPieDataset dataset =
create_pie_dataset(Query_Inventory.get_sql_inventory_year_dist(), "Number",
"year");

        //Draws piechart inventory distribution by film release year
        draw_piechart_with_dataset(frm, jp, dataset, "TOP 10 INVENTORY
DISTRIBUTION BY RELEASE YEAR");
    }

    private static void draw_pie_chart_inventory_by_rating(Charts_Inventory
frm, JPanel jp){
        jp.setPreferredSize(new Dimension(jp.getWidth(), jp.getHeight()));
        DefaultPieDataset dataset =
create_pie_dataset(Query_Inventory.get_sql_inventory_rating_dist(),
"Number", "rating");

        //Draws piechart inventory distribution by film rating
        draw_piechart_with_dataset(frm, jp, dataset, "TOP 10 INVENTORY
DISTRIBUTION BY FILM RATING");
    }

    private static void
draw_pie_chart_inventory_by_language(Charts_Inventory frm, JPanel jp){
        jp.setPreferredSize(new Dimension(jp.getWidth(), jp.getHeight()));
        DefaultPieDataset dataset =
create_pie_dataset(Query_Inventory.get_sql_inventory_lang_dist(), "Number",
"l1.name");

        //Draws piechart inventory distribution by film language
        draw_piechart_with_dataset(frm, jp, dataset, "TOP 10 INVENTORY
DISTRIBUTION BY FILM LANGUAGE");
    }
```

```
    private static void
draw_bar_chart_inventory_by_duration(Charts_Inventory frm, JPanel jp){
        jp.setPreferredSize(new Dimension(jp.getWidth(), jp.getHeight()));

        DefaultCategoryDataset dataset =
create_bar_dataset(Query_Inventory.get_sql_inventory_duration_dist(),
"Number", "duration");

        //Draws barchart store distribution by rental duration
        draw_barchart_with_dataset(frm, jp, dataset, "THE 10 INVENTORY
DISTRIBUTION BY RENTAL DURATION", "RENTAL DURATION", "NUMBER");
    }

    private static void draw_bar_chart_inventory_by_city(Charts_Inventory
frm, JPanel jp){
        jp.setPreferredSize(new Dimension(jp.getWidth(), jp.getHeight()));

        DefaultCategoryDataset dataset =
create_bar_dataset(Query_Inventory.get_sql_inventory_city_dist(), "Number",
"city");

        //Draws barchart store distribution by city
        draw_barchart_with_dataset(frm, jp, dataset, "THE 10 INVENTORY
DISTRIBUTION BY CITY", "CITY", "NUMBER");
    }

    private static void draw_bar_chart_inventory_by_country(Charts_Inventory
frm, JPanel jp){
        jp.setPreferredSize(new Dimension(jp.getWidth(), jp.getHeight()));

        DefaultCategoryDataset dataset =
create_bar_dataset(Query_Inventory.get_sql_inventory_country_dist(),
"Number", "country");

        //Draws barchart store distribution by country
        draw_barchart_with_dataset(frm, jp, dataset, "THE 10 INVENTORY
DISTRIBUTION BY COUNTRY", "COUNTRY", "NUMBER");
    }

    public static void jbchart_handler(Charts_Inventory frm){
        //Draws piechart inventory distribution by film release year
        draw_pie_chart_inventory_by_year(frm, frm.getJPanel1());

        //Draws piechart inventory distribution by film rating
        draw_pie_chart_inventory_by_rating(frm, frm.getJPanel2());

        //Draws piechart inventory distribution by film language
        draw_pie_chart_inventory_by_language(frm, frm.getJPanel3());

        //Draws barchart store distribution by rental duration
        draw_bar_chart_inventory_by_duration(frm, frm.getJPanel4());
```

```
        //Draws barchart store distribution by city
        draw_bar_chart_inventory_by_city(frm, frm.getJPanel5());

        //Draws barchart store distribution by country
        draw_bar_chart_inventory_by_country(frm, frm.getJPanel6());
    }
}
```

This is the full version of **InventoryForm.java**:

```
package sakila;

import java.awt.Toolkit;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.sql.SQLException;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.JButton;
import javax.swing.JComboBox;
import javax.swing.JMenuItem;
import javax.swing.JOptionPane;
import javax.swing.JPopupMenu;
import javax.swing.JTable;
import javax.swing.JTextField;
import javax.swing.event.TableModelEvent;
import javax.swing.event.TableModelListener;

public class InventoryForm extends javax.swing.JFrame {

    public InventoryForm() {
        initComponents();
        Utility.setLookAndFeel(this);
        Inventory_Utils.refresh_controls(this);

 this.setIconImage(Toolkit.getDefaultToolkit().getImage(getClass().getResourc
;
        this.setDefaultCloseOperation(this.HIDE_ON_CLOSE);

    }

    //Getter method for jtfInventoryID
    public JTextField getJTFInventoryID(){
        return this.jtfInventoryID;
    }

    //Getter method for jcbInventoryID
    public JComboBox getJCBInventoryID(){
        return this.jcbInventoryID;
    }
```

```
    //Getter method for jtfFilmID
    public JTextField getJTFFilmID(){
        return this.jtfFilmID;
    }

    //Getter method for jcbFilmID
    public JComboBox getJCBFilmID(){
        return this.jcbFilmID;
    }

    //Getter method for jtfStoreID
    public JTextField getJTFStoreID(){
        return this.jtfStoreID;
    }

    //Getter method for jcbStoreID
    public JComboBox getJCBStoreID(){
        return this.jcbStoreID;
    }

   //Getter method for jtInventory
    public JTable getJTInventory(){
        return this.jtInventory;
    }

    //Getter method for jtfAddressID
    public JTextField getJTFAddressID(){
        return this.jtfAddressID;
    }

    //Getter method for jtfAddress
    public JTextField getJTFAddress(){
        return this.jtfAddress;
    }

    //Getter method for jtfDistrict
    public JTextField getJTFDistrict(){
        return this.jtfDistrict;
    }

    //Getter method for jtfPhone
    public JTextField getJTFPhone(){
        return this.jtfPhone;
    }

    //Getter method for jtfCity
    public JTextField getJTFCity(){
        return this.jtfCity;
    }

    //Getter method for jtfCountry
    public JTextField getJTFCountry(){
        return this.jtfCountry;
```

```
    }

    //Getter method for jtfPostalCode
    public JTextField getJTFPostalCode(){
        return this.jtfPostalCode;
    }

    //Getter method for jtfTitle
    public JTextField getJTFTitle(){
        return this.jtfTitle;
    }

    //Getter method for jtfDescription
    public JTextField getJTFDescription(){
        return this.jtfDescription;
    }

    //Getter method for jtfReleaseYear
    public JTextField getJTFReleaseYear(){
        return this.jtfReleaseYear;
    }

    //Getter method for jtfRentalDuration
    public JTextField getJTFRentalDuration(){
        return this.jtfRentalDuration;
    }

    //Getter method for jtfRentalRate
    public JTextField getJTFRentalRate(){
        return this.jtfRentalRate;
    }

    //Getter method for jtfLength
    public JTextField getJTFLength(){
        return this.jtfLength;
    }

    //Getter method for jtfReplacementCost
    public JTextField getJTFReplacementCost(){
        return this.jtfReplacementCost;
    }

    //Getter method for jtfSpecialFeatures
    public JTextField getJTFSpecialFeatures(){
        return this.jtfSpecialFeatures;
    }

    //Getter method for jtfRating
    public JTextField getJTFRating(){
        return this.jtfRating;
    }
```

```
    //Getter method for jtfLanguage
    public JTextField getJTFLanguage(){
        return this.jtfLanguage;
    }

    //Getter method for jtfLastUpdate
    public JTextField getJTFLastUpdate(){
        return this.jtfLastUpdate;
    }

    //Getter method for jbEdit
    public JButton getJBEdit(){
        return this.jbEdit;
    }

    //Getter method for jbInsert
    public JButton getJBInsert(){
        return this.jbInsert;
    }

    //Getter method for jbDelete
    public JButton getJBDelete(){
        return this.jbDelete;
    }

    //Getter method for jbChart
    public JButton getJBChart(){
        return this.jbChart;
    }

    //Getter method for jbFirst
    public JButton getJBFirst(){
        return this.jbFirst;
    }

    //Getter method for jbPrev
    public JButton getJBPrev(){
        return this.jbPrev;
    }

    //Getter method for jbNext
    public JButton getJBNext(){
        return this.jbNext;
    }

    //Getter method for jbLast
    public JButton getJBLast(){
        return this.jbLast;
    }

        @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">
    private void initComponents() {
```

```
        //...
        pack();
    }// </editor-fold>

    private void jtInventoryMousePressed(java.awt.event.MouseEvent evt) {
        Inventory_Utils.mouse_pressed_handler(this);
    }

    private void jbFirstActionPerformed(java.awt.event.ActionEvent evt) {
        Inventory_Utils.show_first_row(this);
    }

    private void jcbFilmIDActionPerformed(java.awt.event.ActionEvent evt) {
        try {
            Inventory_Utils.jcbFilm_handler(this);
        } catch (SQLException ex) {
            Logger.getLogger(InventoryForm.class.getName()).log(Level.SEVERE,
        }
    }

    private void jbPrevActionPerformed(java.awt.event.ActionEvent evt) {
        Inventory_Utils.show_prev_row(this);
    }

    private void jbNextActionPerformed(java.awt.event.ActionEvent evt) {
        Inventory_Utils.show_next_row(this);
    }

    private void jbLastActionPerformed(java.awt.event.ActionEvent evt) {
        Inventory_Utils.show_last_row(this);
    }

    private void jbChartActionPerformed(java.awt.event.ActionEvent evt) {
        Charts_Inventory frm1 = new Charts_Inventory();
        frm1.setLocationRelativeTo(null);
        frm1.setTitle("SIX DISTRIBUTIONS IN INVENTORY TABLE");
        frm1.setVisible(true);
        Inventory_Utils.jbchart_handler(frm1);
    }

    private void jbEditActionPerformed(java.awt.event.ActionEvent evt) {
        Inventory_Utils.edit_handler(this);
    }

    private void jbInsertActionPerformed(java.awt.event.ActionEvent evt) {
        Inventory_Utils.insert_handler(this);
    }

    private void jbDeleteActionPerformed(java.awt.event.ActionEvent evt) {
        Inventory_Utils.delete_handler(this);
    }
```

```
    private void jbCountryFormActionPerformed(java.awt.event.ActionEvent evt)
        CountryForm ct_form = new CountryForm();
        ct_form.setVisible(true);
    }

    private void jbAddressFormActionPerformed(java.awt.event.ActionEvent evt)
        AddressForm add_form = new AddressForm();
        add_form.setVisible(true);
    }

    private void jbStoreFormActionPerformed(java.awt.event.ActionEvent evt) {
        StoreForm str_form = new StoreForm();
        str_form.setVisible(true);
    }

    private void jbCityFormActionPerformed(java.awt.event.ActionEvent evt) {
        CityForm cty_form = new CityForm();
        cty_form.setVisible(true);
    }

    private void jcbInventoryIDActionPerformed(java.awt.event.ActionEvent evt
        Inventory_Utils.jcbInventory_handler(this);
    }

    private void jcbStoreIDActionPerformed(java.awt.event.ActionEvent evt) {
        Inventory_Utils.jcbStore_handler(this);
    }

    /**
     * @param args the command line arguments
     */
    public static void main(String args[]) {
        try {
            for (javax.swing.UIManager.LookAndFeelInfo info :
javax.swing.UIManager.getInstalledLookAndFeels()) {
                if ("Nimbus".equals(info.getName())) {
                    javax.swing.UIManager.setLookAndFeel(info.getClassName()
                    break;
                }
            }
        } catch (ClassNotFoundException ex) {

java.util.logging.Logger.getLogger(InventoryForm.class.getName()).log(java.ut
null, ex);
        } catch (InstantiationException ex) {

java.util.logging.Logger.getLogger(InventoryForm.class.getName()).log(java.ut
null, ex);
        } catch (IllegalAccessException ex) {

java.util.logging.Logger.getLogger(InventoryForm.class.getName()).log(java.ut
null, ex);
        } catch (javax.swing.UnsupportedLookAndFeelException ex) {
```

```
java.util.logging.Logger.getLogger(InventoryForm.class.getName()).log(java.ut
null, ex);
        }
        /* Create and display the form */
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new InventoryForm().setVisible(true);
            }
        });
    }

    // Variables declaration - do not modify
    private javax.swing.JLabel jLabel10;
    private javax.swing.JLabel jLabel11;
    private javax.swing.JLabel jLabel12;
    private javax.swing.JLabel jLabel13;
    private javax.swing.JLabel jLabel15;
    private javax.swing.JLabel jLabel16;
    private javax.swing.JLabel jLabel17;
    private javax.swing.JLabel jLabel18;
    private javax.swing.JLabel jLabel19;
    private javax.swing.JLabel jLabel20;
    private javax.swing.JLabel jLabel21;
    private javax.swing.JLabel jLabel22;
    private javax.swing.JLabel jLabel23;
    private javax.swing.JLabel jLabel24;
    private javax.swing.JLabel jLabel3;
    private javax.swing.JLabel jLabel4;
    private javax.swing.JLabel jLabel5;
    private javax.swing.JLabel jLabel6;
    private javax.swing.JLabel jLabel7;
    private javax.swing.JLabel jLabel8;
    private javax.swing.JLabel jLabel9;
    private javax.swing.JScrollPane jScrollPane1;
    private javax.swing.JButton jbAddressForm;
    private javax.swing.JButton jbChart;
    private javax.swing.JButton jbCityForm;
    private javax.swing.JButton jbCountryForm;
    private javax.swing.JButton jbDelete;
    private javax.swing.JButton jbEdit;
    private javax.swing.JButton jbFirst;
    private javax.swing.JButton jbInsert;
    private javax.swing.JButton jbLast;
    private javax.swing.JButton jbNext;
    private javax.swing.JButton jbPrev;
    private javax.swing.JButton jbStoreForm;
    private javax.swing.JComboBox<String> jcbFilmID;
    private javax.swing.JComboBox<String> jcbInventoryID;
    private javax.swing.JComboBox<String> jcbStoreID;
    private javax.swing.JTable jtInventory;
    private javax.swing.JTextField jtfAddress;
    private javax.swing.JTextField jtfAddressID;
```

```
    private javax.swing.JTextField jtfCity;
    private javax.swing.JTextField jtfCountry;
    private javax.swing.JTextField jtfDescription;
    private javax.swing.JTextField jtfDistrict;
    private javax.swing.JTextField jtfFilmID;
    private javax.swing.JTextField jtfInventoryID;
    private javax.swing.JTextField jtfLanguage;
    private javax.swing.JTextField jtfLastUpdate;
    private javax.swing.JTextField jtfLength;
    private javax.swing.JTextField jtfPhone;
    private javax.swing.JTextField jtfPostalCode;
    private javax.swing.JTextField jtfRating;
    private javax.swing.JTextField jtfReleaseYear;
    private javax.swing.JTextField jtfRentalDuration;
    private javax.swing.JTextField jtfRentalRate;
    private javax.swing.JTextField jtfReplacementCost;
    private javax.swing.JTextField jtfSpecialFeatures;
    private javax.swing.JTextField jtfStoreID;
    private javax.swing.JTextField jtfTitle;
    // End of variables declaration
}
```

CUSTOMER FORM

CUSTOMER FORM

CREATING AND POPULATING CUSTOMER TABLE

CREATING AND POPULATING CUSTOMER TABLE

Step 1	Create a new class named **Query_Customer**. It contains a set of static final strings that represent SQL queries used to retrieve data from the customer table in the sakila database. Here is an explanation of each query: • **sql_min**: This query selects the minimum value of the **customer_id** column in the **customer** table. • **sql_max**: This query selects the maximum value of the **customer_id** column in the **customer** table. • **sql_id**: This query selects all columns for a single row in the **customer** table, where the **customer_id** column matches a specified value. The query uses a parameterized prepared statement with a placeholder (?) for the customer ID value. • **sql_name**: This query selects all columns for a single row in the customer table, where the first name and last name concatenated together matches a specified value. The query uses a parameterized prepared statement with a placeholder (?) for the full name value. • **sql_customer_district_dist**: This query selects the district and the number of customers in each district, for the top 10 districts with

the most customers. The query joins the **customer** and **address** tables on the **address_id** column and groups the results by the district column in the **address** table.

- **sql_customer_country_dist**: This query selects the country and the number of customers in each country, for the top 10 countries with the most customers. The query joins the **customer**, **address**, **city**, and **country** tables on the appropriate foreign keys and groups the results by the **country** column in the **country** table.
- **sql_customer_city_dist**: This query selects the city and the number of customers in each city, for the top 10 cities with the most customers. The query joins the **customer** and **address** tables on the **address_id** column and groups the results by the **city** column in the **address** table.
- **sql_customer_store_district_dist**: This query selects the district and the number of customers in each store district, for the top 10 districts with the most customers in a particular store. The query joins the **customer**, **store**, **address**, **city**, and **country** tables on the appropriate foreign keys and groups the results by the **district** column in the **address** table.
- **sql_customer_store_country_dist**: This query selects the country and the number of customers in each store country, for the top 10 countries with the most customers in a particular store. The query joins the **customer**, **store**, **address**, **city**, and **country** tables on the appropriate foreign keys and groups the results by the **country** column in the **country** table.

- **sql_customer_store_city_dist**: This query selects the city and the number of customers in each store city, for the top 10 cities with the most customers in a particular store. The query joins the customer, store, address, city, and country tables on the appropriate foreign keys and groups the results by the city column in the address table.
- **sql_customer**: This query creates the **customer** table, including all columns and constraints. The customer table has a foreign key relationship with the address and store tables.
- **sql_customer_joint string**: This query contains a SQL query that joins several tables to retrieve information about customers. Specifically, it joins the customer, store, address, city, and country tables. The resulting query selects the customer_id, store_id, first_name, last_name, email, address_id, active, create_date, and last_update columns from the customer table, as well as several columns from the other joined tables. The address table provides the address, district, postal_code, and phone columns, the city table provides the city column, and the country table provides the country column. The resulting query returns a row for each customer, with information about their store, address, city, and country.

```
1  package sakila;
2
3  public class Query_Customer {
4      private static final String
5  sql_min = "SELECT MIN(customer_id)
6  FROM customer";
7      private static final String
8  sql_max = "SELECT MAX(customer_id)
9  FROM customer";
10
```

```
    private static final String
sql_id = "SELECT * FROM customer
WHERE customer_id  = ?";
    private static final String
sql_name = "SELECT * FROM customer
WHERE CONCAT(first_name,'
',last_name) = ?";

    private static final String
sql_customer_district_dist = """
        SELECT ad.district AS
customer_district, COUNT(*) AS
number_of_customers
        FROM customer cu
        JOIN address ad ON
ad.address_id = cu.address_id
        GROUP BY customer_district
        ORDER BY
number_of_customers DESC
        LIMIT 10;""";

    private static final String
sql_customer_country_dist = """
        SELECT co.country AS
customer_country, COUNT(*) AS
number_of_customers
        FROM customer cu
        JOIN address ad ON
ad.address_id = cu.address_id
        JOIN city ci ON ci.city_id
= ad.city_id
        JOIN country co ON
co.country_id = ci.country_id
        GROUP BY customer_country
        ORDER BY
number_of_customers DESC
        LIMIT 10;""";

    private static final String
sql_customer_city_dist = """
        SELECT ci.city AS
customer_city, COUNT(*) AS
number_of_customers
        FROM customer cu
        JOIN address ad ON
ad.address_id = cu.address_id
        JOIN city ci ON ci.city_id
= ad.city_id
        GROUP BY customer_city
        ORDER BY
number_of_customers DESC
        LIMIT 10;""";


```

```
    private static final String
sql_customer_store_district_dist =
"""
        SELECT ad.district AS
store_district, COUNT(*) AS
number_of_customers
        FROM customer cu
        JOIN store st ON
st.store_id = cu.store_id
        JOIN address ad ON
ad.address_id = st.address_id
        JOIN city ci ON ci.city_id
= ad.city_id
        JOIN country co ON
co.country_id = ci.country_id
        GROUP BY store_district
        ORDER BY
number_of_customers DESC
        LIMIT 10;""";

    private static final String
sql_customer_store_country_dist =
"""
```

```
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
```

```
        SELECT co.country AS
store_country, COUNT(*) AS
number_of_customers
        FROM customer cu
        JOIN store st ON st.store_id =
cu.store_id
        JOIN address ad ON
ad.address_id = st.address_id
        JOIN city ci ON ci.city_id =
ad.city_id
        JOIN country co ON
co.country_id = ci.country_id
        GROUP BY store_country
        ORDER BY number_of_customers
DESC
        LIMIT 10;""";

    private static final String
sql_customer_store_city_dist = """
        SELECT ci.city AS store_city,
COUNT(*) AS number_of_customers
        FROM customer cu
        JOIN store st ON st.store_id =
cu.store_id
        JOIN address ad ON
ad.address_id = st.address_id
        JOIN city ci ON ci.city_id =
ad.city_id
        JOIN country co ON
co.country_id = ci.country_id
        GROUP BY store_city
        ORDER BY number_of_customers
DESC
        LIMIT 10;""";

    private static final String
sql_customer_joint = """
        SELECT cu.customer_id,
cu.store_id, cu.first_name,
            cu.last_name, cu.email,
cu.address_id,
            cu.active, cu.create_date,
cu.last_update,
            ad.address, ad.district,
ad.postal_code,
            ad.phone, ci.city,
co.country
        FROM customer cu
        JOIN store st ON st.store_id =
cu.store_id
        JOIN address ad ON
ad.address_id = cu.address_id
```

```
        JOIN city ci ON ci.city_id =
ad.city_id
        JOIN country co ON
co.country_id = ci.country_id""";

    private static final String
sql_customer = """
        CREATE TABLE customer (
          customer_id SMALLINT UNSIGNED
NOT NULL AUTO_INCREMENT,
          store_id TINYINT UNSIGNED NOT
NULL,
          first_name VARCHAR(45) NOT
NULL,
          last_name VARCHAR(45) NOT
NULL,
          email VARCHAR(50) DEFAULT
NULL,
          address_id SMALLINT UNSIGNED
NOT NULL,
          active BOOLEAN NOT NULL
DEFAULT TRUE,
          create_date DATETIME NOT
NULL,
          last_update TIMESTAMP DEFAULT
CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,
          PRIMARY KEY  (customer_id),
          KEY idx_fk_store_id
(store_id),
          KEY idx_fk_address_id
(address_id),
          KEY idx_last_name
(last_name),
          CONSTRAINT
fk_customer_address FOREIGN KEY
(address_id) REFERENCES address
(address_id) ON DELETE RESTRICT ON
UPDATE CASCADE,
          CONSTRAINT fk_customer_store
FOREIGN KEY (store_id) REFERENCES store
(store_id) ON DELETE RESTRICT ON UPDATE
CASCADE
        ) ENGINE=InnoDB DEFAULT
CHARSET=utf8mb4;""";

    //Getter methods
    public static String get_sql_min()
{
        return sql_min;
    }

    public static String get_sql_max()
{
```

```
        return sql_max;
    }

    public static String get_sql_id() {
        return sql_id;
    }

    public static String get_sql_name()
{
        return sql_name;
    }

    public static String
get_sql_customer() {
        return sql_customer;
    }

    public static String
get_sql_customer_joint() {
        return sql_customer_joint;
    }

    public static String
get_sql_customer_store_country_dist() {
        return
sql_customer_store_country_dist;
    }

    public static String
get_sql_customer_store_city_dist() {
        return
sql_customer_store_city_dist;
    }

    public static String
get_sql_customer_store_district_dist()
{
        return
sql_customer_store_district_dist;
    }

    public static String
get_sql_customer_city_dist() {
        return sql_customer_city_dist;
    }

    public static String
get_sql_customer_country_dist() {
        return
sql_customer_country_dist;
    }
```

```
    public static String
get_sql_customer_district_dist() {
        return
sql_customer_district_dist;
    }
}
```

Step 2

Then, create a public class named **C**
15 instance variables and provides
each variable. The instance variable
store information, such as ID, nan
phone number.

The class provides several cons
numbers of parameters. There is a
initializes some of the instance varia
and constructors with eight, nine, ar
creating instances with specific valu

The class also has input validation i
ensure that the data is valid before s
variables. For example, the
setLastName() methods validate tl
or empty and that it is not long
Similarly, the setEmail method val
not longer than 50 characters.

```
1   package sakila;
2   import java.util.Objects;
3   import java.sql.Timestamp;
4   import java.sql.Date;
5   import java.util.Calendar;
6
7   public class Customer {
8       //15 instance variables
9       private int cust_id;
10      private int store_id;
11      private String fname;
12      private String lname;
13      private String email;
14      private int add_id;
15      private boolean active;
16      private Date create_dat
17      private Timestamp last_
18
19      private String cust_add
20      private String cust_dis
21      private String cust_cit
22
```

```
23      private String cust_cou
24      private String cust_pos
25      private String cust_pho
26
27      //Default constructor
28      Customer(){
29          this(1, 1, "Fname",
30  "xxx@gmail.com", 1, true,
31                  new
32  Date(Calendar.getInstance()
33  new Timestamp(System.curren
34      }
35
36      //Constructor with eigh
37      Customer(int store_id,
38  lname, String email,
39              int add_id, boo
40  cr_date, Timestamp lu){
41          setStoreID(store_id
42          setFirstName(fname)
43          setLastName(lname);
44          setAddressID(add_id
45          setEmail(email);
46          setActive(active);
47          setCreateDate(cr_da
48          setLastUpdate(lu);
49      }
50
51      //Constructor with nine
52      Customer(int cust_id, i
53  fname, String lname,
54          String email, int a
55  Date cr_date, Timestamp lu)
56          this(store_id, fnam
57  add_id, active, cr_date, lu
58          setCustomerID(cust_
59      }
60
61      //Constructor with fift
62      Customer(int cust_id, i
63  fname, String lname,
64          String email, int a
65  Date cr_date, Timestamp lu,
66          String cust_address
67  cust_district, String cust_
68          String cust_country
69  cust_postal_code, String cu
70          this(cust_id, store
71  email, add_id, active, cr_c
72          this.cust_address =
73          this.cust_district
74          this.cust_city = cu
75          this.cust_country =
76
```

```
77          this.cust_postal_cc
78          this.cust_phone = c
79      }
80
81      //Getter methods
82      public int getCustomerI
83      public int getStoreID()
84      public String getFirstN
85      public String getLastNa
86      public String getEmail(
87      public int getAddressID
88      public boolean getActiv
89      public Date getCreateDa
90  this.create_date;}
91      public Timestamp getLas
92  last_update;}
93
94      public String getCustom
95  cust_address;}
96      public String getCustom
97  cust_city;}
98      public String getCustom
99  cust_district;}
100     public String getCustom
101 cust_country;}
102     public String getCustom
103 cust_postal_code;}
104     public String getCustom
105 cust_phone;}
106
107     //Setter methods
108     public void setCustomer
109         if (id <= 0) {
110             throw new
111 IllegalArgumentException("C
112 greater than zero.");
113         }
114         this.cust_id = id;
115     }
116
117     public void setStoreID(
118         if (id <= 0) {
119             throw new
120 IllegalArgumentException("S
121 greater than zero.");
122         }
123         this.store_id = id;
124     }
125
126127    public void setFirstNam
128         if (name == null ||
129 name.trim().isEmpty()) {
130
131
```

```
132             throw new
133     IllegalArgumentException("F
134     null or empty");
135         }
136         if (name.length() >
137             throw new
138     IllegalArgumentException("F
139     longer than 45 characters")
140         }
141         this.fname = name;
142     }
143
144     public void setLastName
145         if (name == null ||
146     name.trim().isEmpty()) {
147             throw new
148     IllegalArgumentException("L
149     null or empty");
150         }
151         if (name.length() >
152             throw new
153     IllegalArgumentException("L
154     longer than 45 characters")
155         }
156         this.lname = name;
157     }
158
159     public void setEmail(St
160         if (email != null &
161     {
162             throw new
163     IllegalArgumentException("E
164     than 50 characters");
165         }
166         this.email = (email
167     ""; // set default value to
168     is null
169     }

    public void setAddressI
        if (id <= 0) {
            throw new
    IllegalArgumentException("A
    greater than zero.");
        }
        this.add_id = id;
    }

    public void setActive(b
        this.active = activ
    }

    public void setCreateDa
```

```
        this.create_date =
    }

    public void setLastUpda
        if (date == null) {
        throw new IllegalAr
cannot be null");
        }
        this.last_update =
    }

    @Override
    public String toString(
        return "\nCustomer
getCustomerID() +
                "\nStore ID
getStoreID() +
                "\nCustomer
getFirstName() + " " + get
                "\nEmail
getEmail() +
                "\nAddress I
getAddressID() +
                "\nActive
getActive() +
                "\nCreate Da
getCreateDate() +
                "\nLast Upda
getLastUpdate();
    }
}
```

Step 3

Create a new public class name extends the **Utility** class and pr related to the **customer** table in dat

The class has a public static final int an INVALID_INDEX field, both with a private static int field **curre** FIRST_INDEX.

The class has a private static fin initialized to the **Query_Customer.get_sql_id()** meth

The class has a public static **create_customer_table()** which table in the database by executi returned by the **Query_Custon**

method call. It also shows a succes
dialog if the table is created succ
error message in a message dialog i

The class has a public static
populate_customer_table() which
data into the customer table in the
Customer objects, one using the
the other using the nine-params co
properties. It then executes a prep
SQL statement to insert the data in
error message in a message dialog i

The class has a public static
read_customer_table() which rea
customer table from the database
objects for each row. It executes a
rows from the table, and uses a **R**
the rows. For each row, it gets th
and creates a new **Customer** objec
constructor. It then prints the **C**
console. It shows an error message i
exception occurs.

The class imports several packa
java.util.logging, java.sql, java.u
org.jfree.data.

```
1  package sakila;
2  import java.awt.Dimension;
3  import java.util.logging.Level
4  import java.util.logging.Logge
5  import java.sql.*;
6  import java.util.ArrayList;
7  import java.util.Arrays;
8  import java.util.Calendar;
9  import java.util.HashMap;
10 import java.util.Objects;
11 import javax.swing.JComboBox;
12 import javax.swing.JOptionPane
13 import javax.swing.JPanel;
14 import javax.swing.event.Table
15 import javax.swing.table.Defau
16 import javax.swing.table.Table
17 import
18 org.jfree.data.category.Defaul
19 import org.jfree.data.general.
20 import static sakila.Utility.c
21
```

```
22  import static
23  sakila.Utility.draw_barchart_v
24
25  public class Customer_Utils ex
26      public static final int FI
27      public static final int IN
28
29      private static int current
30  FIRST_INDEX;
31      private static final Strin
32  Query_Customer.get_sql_id();
33
34      //Creates customer table
35      public static void create_
36          try (Connection conn =
37              Statement stmt =
38  conn.createStatement();
39
40  stmt.addBatch(Query_Customer.g
41              stmt.executeBatch(
42
43              String message =
44  String.format("Successfully cr
45  table");
46              JOptionPane.showMe
47  message,
48
49   "INFORMATION",JOptionPane.INF
50
51          } catch (SQLException
52              JOptionPane.showMe
53  ex.getMessage(),
54
55   "ERROR",JOptionPane.ERROR_MES
56          }
57      }
58
59      //Populates customer table
60  data
61      public static void populat
62          try(Connection conn =
63              String sql = """
64                  INSERT INTO cu
65  store_id, first_name, last_nam
66                      email, add
67  create_date, last_update)
68                  VALUES(?, ?, ?
69  ?)""";
70
71              //Creates a new Cu
72  default constructor
73              PreparedStatement
74  conn.prepareStatement(sql);
75
```

```
            Customer obj1 = ne
            ps1.setInt(1,obj1.
            ps1.setInt(2,obj1.
            ps1.setString(3,ob
            ps1.setString(4,ob
            ps1.setString(5,ob
            ps1.setInt(6,obj1.
            ps1.setBoolean(7,o
            ps1.setDate(8,obj1

ps1.setTimestamp(9,obj1.getLas

            // Creates a new C
nine-params constructor
            PreparedStatement
conn.prepareStatement(sql);
            Customer obj2 = ne
"Vivian", "Siahaan", "vivian@g
false, new
Date(Calendar.getInstance().ge
                    new
Timestamp(System.currentTimeMi
            ps2.setInt(1,obj2.
            ps2.setInt(2,obj2.
            ps2.setString(3,ob
            ps2.setString(4,ob
            ps2.setString(5,ob
            ps2.setInt(6,obj2.
            ps2.setBoolean(7,o
            ps2.setDate(8,obj2

ps2.setTimestamp(9,obj2.getLas

            ps1.executeUpdate(
            ps2.executeUpdate(

        }catch(SQLException ex
            JOptionPane.showMe
ex.getMessage(),

 "ERROR",JOptionPane.ERROR_MES
        }
    }

    //Reads the content of cus
    public static void read_cu
        try(Connection conn =
            Statement stmt =
conn.createStatement();
            ResultSet rs =
stmt.executeQuery("SELECT * FR

            while(rs.next()){
```

```
                int cust_id =
rs.getInt("customer_id");
                int store_id =
rs.getInt("store_id");
                int add_id =
rs.getInt("address_id");
                String fname =
rs.getString("first_name");
                String lname =
rs.getString("last_name");
                String email =
rs.getString("email");
                boolean active
rs.getBoolean("active");
                Date cr_date =
rs.getDate("create_date");
                Timestamp lu =
rs.getTimestamp("last_update")

                //Creates a Cu
nine-params constructor
                Customer obj =
Customer(cust_id, store_id, fr
add_id, active, cr_date, lu);
                System.out.pri
            }
            rs.close();
            stmt.close();

        }catch(SQLException ex
            JOptionPane.showMe
ex.getMessage(),

 "ERROR",JOptionPane.ERROR_MES
        }
    }
}
```

Step 4

In the driver class, **create_customer_table()**, **popu** and **read_customer_table()** as sho

```
1  package sakila;
2
3  public class Sakila {
4      public static void main(Str
5  {
6          //...
7
8  //        Store_Utils.create_st
9
```

```
//
Store_Utils.populate_store_tabl
//        Store_Utils.read_stor
//        StoreForm frm = new S
//        frm.setVisible(true);

//
Inventory_Utils.create_inventor
//
Inventory_Utils.populate_invent
//
Inventory_Utils.read_inventory_
//        InventoryForm frm = n
InventoryForm();
//        frm.setVisible(true);


Customer_Utils.create_customer_

Customer_Utils.populate_custome

Customer_Utils.read_customer_ta
    }
}
```

Run project to see the result in cons

```
Customer ID       : 1
Store ID          : 1
Customer Name     : Fname Lname
Email             : xxx@gmail.com
Address ID        : 1
Active            : true
Create Date       : 2023-04-30
Last Update       : 2023-04-30 16:4

Customer ID       : 2
Store ID          : 2
Customer Name     : Vivian Siahaan
Email             : vivian@gmail.co
Address ID        : 2
Active            : false
Create Date       : 2023-04-30
Last Update       : 2023-04-30 16:4
```

DESIGNING GUI

DESIGNING GUI

Step 1	In the project, create a new **JFrame Form** and name it as **CustomerForm.java**. In the Design tab, add twenty-one **JLabels** to the form and set their corresponding **text** properties as CUSTOMER ID, FULL NAME, FIRST NAME, LAST NAME, ADDRESS ID, ADDRESS, EMAIL, DISTRICT, POSTAL CODE, CITY, COUNTRY, PHONE, ACTIVE, CREATE DATE, STORE ID, STORE DISTRICT, STORE COUNTRY, STORE CITY, STORE ADDRESS, STORE PHONE, and LAST UPDATE.
Step 2	Then, add nineteen **JTextField** to the form and set their corresponding **Variable Name** as **jtfCustomerID**, **jtfFirstName**, **jtfLastName**, **jtfAddressID**, **jtfAddress**, **jtfEmail**, **jtfPostalCode**, **jtfPhone**, **jtfDistrict**, **jtfCity**, **jtfCountry**, **jtfCreateDate**, **jtfStoreID**, **jtfStoreDistrict**, **jtfStoreCountry**, **jtfStoreCity**, **jtfStoreAddress**, **jtfStorePhone**, and **jtfLastUpdate**.
Step 3	Then, add twelve **JButton** to the form and set their corresponding **Variable Name** as **jbFirst**, **jbPrev**, **jbNext**, **jbLast**, **jbEdit**, **jbInsert**, **jbDelete**, **jbAddress**, **jbCountryForm**, **jbCityForm**, **jbAddressForm**, **jbStoreForm**, and **jbChart**. Set their corresponding **text** properties as FIRST, PREV, NEXT, LAST, EDIT, INSERT, DELETE, ADDRESS FORM, COUNTRY FORM, CITY FORM, STORE FORM, and CHART.
Step 4	Then, add five **JComboBoxes** to the form and set their corresponding **Variable Name** as **jcbCustomerID**, **jcbCustomerName**, **jcbStoreID**, **jcbAddressID**, and **jcbActive**.
Step 5	Lastly, add a new **JTable** to the form set set its **Variable Name** as **jtCustomer**. Then, right-click on it, then choose **Table Contents...** and set the number of columns to 15 and the number of rows to 50.

Step 6

In the driver class, **Sakila.java**, create a new object of **CustomerForm** class using its default constructor as shown in 16 – 17:

```
1  package sakila;
2
3  public class Sakila {
4      public static void main(String[]
5  args) {
6      //...
7
8  //
9  Address_Utils.create_address_table();
10 //
11 Address_Utils.populate_address_table();
12 //
13 Address_Utils.read_address_table();
14 //         AddressForm frm = new
15 AddressForm();
16 //         frm.setVisible(true);
17
18 //
19 Customer_Utils.create_customer_table();
   //
   Customer_Utils.populate_customer_table();
   //
   Customer_Utils.read_customer_table();
           CustomerForm frm = new
   CustomerForm();
           frm.setVisible(true);
       }
   }
```

Step 7

In **CustomerForm**'s constructor, invoke **setLookAndFeel()** to set the look and feel of the form as shown in line 17.

```
1  package sakila;
2
3  import java.awt.Toolkit;
4  import java.awt.event.ActionEvent;
5  import
6  java.awt.event.ActionListener;
7  import javax.swing.JButton;
8  import javax.swing.JComboBox;
9  import javax.swing.JMenuItem;
10 import javax.swing.JOptionPane;
11 import javax.swing.JPopupMenu;
12 import javax.swing.JTable;
```

```
13 import javax.swing.JTextField;
14
15 public class CustomerForm extends
16 javax.swing.JFrame {
17     public CustomerForm() {
18         initComponents();
19
20 Utility.setLookAndFeel(this);
      }
      //...
}
```

Run the project to see the customer form as shown in Figure 13.1.

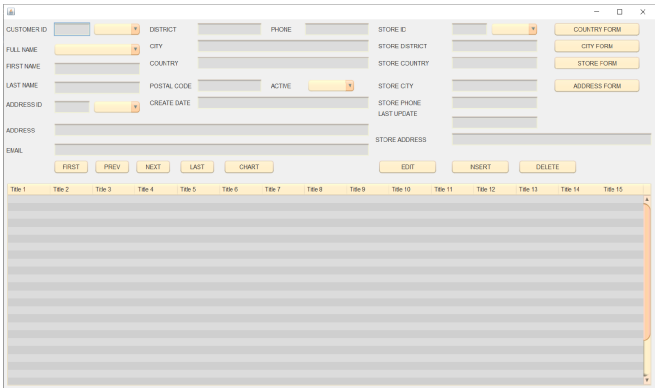


Figure 13.1 The layout of customer form

Step 9

In **CustomerForm.java**, define **getter()** method for every object in the form and place them outside and beneath its default constructor:

```
1     //Getter method for jtCustomer
2     public JTable getJTCustomer(){
3         return this.jtCustomer;
4     }
5
6     //Getter method for
7 jtfCustomerID
8     public JTextField
9 getJTFCustomerID(){
10        return this.jtfCustomerID;
11    }
12
13    //Getter method for
14 jcbCustomerID
15    public JComboBox
16 getJCBCustomerID(){
17
```

```
        return this.jcbCustomerID;
    }

    //Getter method for
jcbCustomerName
    public JComboBox
getJCBCustomerName(){
        return
this.jcbCustomerName;
    }

    //Getter method for
jtfFirstName
    public JTextField
getJTFFirstName(){
        return this.jtfFirstName;
    }

    //Getter method for jtfLastName
    public JTextField
getJTFLastName(){
        return this.jtfLastName;
    }

    //Getter method for
jtfAddressID
    public JTextField
getJTFAddressID(){
        return this.jtfAddressID;
    }

    //Getter method for
jcbAddressID
    public JComboBox
getJCBAddressID(){
        return this.jcbAddressID;
    }

    //Getter method for jtfAddress
    public JTextField
getJTFAddress(){
        return this.jtfAddress;
    }

    //Getter method for jtfEmail
    public JTextField getJTFEmail()
{
        return this.jtfEmail;
    }

    //Getter method for jtfDistrict
    public JTextField
getJTFDistrict(){

```

```
72          return this.jtfDistrict;
73      }
74
75      //Getter method for jtfCity
76      public JTextField getJTFCity(){
77          return this.jtfCity;
78      }
79
80      //Getter method for jtfCountry
81      public JTextField
82  getJTFCountry(){
83          return this.jtfCountry;
84      }
85
86      //Getter method for
87  jtfPostalCode
88      public JTextField
89  getJTFPostalCode(){
90          return this.jtfPostalCode;
91      }
92
93      //Getter method for jtfPhone
94      public JTextField getJTFPhone()
95  {
96          return this.jtfPhone;
97      }
98
99      //Getter method for jcbActive
100     public JComboBox getJCBActive()
101 {
102         return this.jcbActive;
103     }
104
105     //Getter method for jcbStoreID
106     public JComboBox
107 getJCBStoreID(){
108         return this.jcbStoreID;
109     }
110
111     //Getter method for jtfStoreID
112     public JTextField
113 getJTFStoreID(){
114         return this.jtfStoreID;
115     }
116
117     //Getter method for
118 jtfLastUpdate
119     public JTextField
120 getJTFLastUpdate(){
121         return this.jtfLastUpdate;
122     }
123
124
125
```

```
126     //Getter method for
127 jtfCreateDate
128     public JTextField
129 getJTFCreateDate(){
130         return this.jtfCreateDate;
131     }
132 
133     //Getter method for
134 jtfStoreDistrict
135     public JTextField
136 getJTFStoreDistrict(){
137         return
138 this.jtfStoreDistrict;
139     }
140 
141     //Getter method for
142 jtfStoreCity
143     public JTextField
144 getJTFStoreCity(){
145         return this.jtfStoreCity;
146     }
147 
148     //Getter method for
149 jtfStoreCountry
150     public JTextField
151 getJTFStoreCountry(){
152         return
153 this.jtfStoreCountry;
154     }
155 
156     //Getter method for
157 jtfStorePhone
158     public JTextField
159 getJTFStorePhone(){
160         return this.jtfStorePhone;
161     }
162 
163     //Getter method for
164 jtfStoreAddress
165     public JTextField
166 getJTFStoreAddress(){
167         return
168 this.jtfStoreAddress;
169     }
170 
171     //Getter method for jbEdit
172     public JButton getJBEdit(){
173         return this.jbEdit;
174     }
175 
176     //Getter method for jbInsert
177     public JButton getJBInsert(){
178         return this.jbInsert;
179 
```

```
180     }
181
182     //Getter method for jbDelete
183     public JButton getJBDelete(){
184         return this.jbDelete;
        }

        //Getter method for jbChart
        public JButton getJBChart(){
            return this.jbChart;
        }

        //Getter method for jbFirst
        public JButton getJBFirst(){
            return this.jbFirst;
        }

        //Getter method for jbPrev
        public JButton getJBPrev(){
            return this.jbPrev;
        }

        //Getter method for jbNext
        public JButton getJBNext(){
            return this.jbNext;
        }

        //Getter method for jbLast
        public JButton getJBLast(){
            return this.jbLast;
        }

        //Getter method for
    jbAddressForm
        public JButton
    getJBAddressForm(){
            return this.jbAddressForm;
        }

        //Getter method for
    jbCountryForm
        public JButton
    getJBCountryForm(){
            return this.jbCountryForm;
        }

        //Getter method for jbCityForm
        public JButton getJBCityForm(){
            return this.jbCityForm;
        }

        //Getter method for jbStoreForm
```

```
    public JButton getJBStoreForm()
{
        return this.jbStoreForm;
    }
```

POPULATING TABLE AND COMBOBOXES

POPULATING TABLE AND COMBOBOXES

Step 1

In **Customer_Utils.java**, add two new methods: **get_customer_list()** a
method, **get_customer_list()**, takes in three parameters: a **CustomerF**
string item. It returns an **ArrayList** of **Customer** objects based on the res

The SQL query is passed as a parameter to the method and may contain a
with the item parameter if it is not equal to "none". The method establis
the **getConnection()** method and creates a prepared statement object us
to "none", the prepared statement object sets the first parameter to item.

The method then executes the SQL query and retrieves a **ResultSet** obje
method creates a Customer object using the fifteen-params constructo
SQLException is caught, the method displays an error message using the

The second method, **show_table_customer()**, takes in two paramete
ArrayList of **Customer** objects. The method sets up a **DefaultTableMo**
headers and sets the model of the **JTable** in the **CustomerForm** object to

The method then iterates through the **Customer** objects in the **ArrayLis**
appropriate data using the **setValueAt()** method of the table model.

```
1     private static ArrayList<Customer>
2  get_customer_list(CustomerForm frm, String sql, String item){
3         ArrayList<Customer> list = new ArrayList<>();
4
5         try(Connection conn = getConnection();
6             PreparedStatement ps = conn.prepareStatement(sql)){
7             if (item.equalsIgnoreCase("none")==false) {
8                  ps.setString(1,item);
9             }
10            ResultSet rs = ps.executeQuery();
11            Customer obj;
12            while(rs.next()){
13                //Using fifteen-params constructor
14                obj = new Customer(rs.getInt("customer_id"),
15                        rs.getInt("store_id"),
16                        rs.getString("first_name"),
17                        rs.getString("last_name"),
```

```
                        rs.getString("email"),
                        rs.getInt("address_id"),
                        rs.getBoolean("active"),
                        rs.getDate("create_date"),
                        rs.getTimestamp("last_update"),
                        rs.getString("address"),
                        rs.getString("district"),
                        rs.getString("city"),
                        rs.getString("country"),
                        rs.getString("postal_code"),
                        rs.getString("phone"));

                list.add(obj);
            }
        }catch (SQLException ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
                "ERROR",JOptionPane.ERROR_MESSAGE);
        }
        return list;
    }

    private static void show_table_customer(CustomerForm frm,
ArrayList<Customer> list) throws SQLException{
        DefaultTableModel model = new DefaultTableModel(0,0);

        String header[] = {"Customer ID", "Store ID", "First
Name", "Last Name", "Email", "Address ID", "Active", "Create
Date", "Address", "District",
            "City", "Country", "Postal Code", "Phone", "Last
Update"};


model.setColumnIdentifiers(set_column_header(frm.getJTCustomer(),
header));
        frm.getJTCustomer().setModel(model);

        Object[] row = new Object[15];

        for(int i=0; i<list.size(); i++){
            row[0] = list.get(i).getCustomerID();
            row[1] = list.get(i).getStoreID();
            row[2] = list.get(i).getFirstName();
            row[3] = list.get(i).getLastName();
            row[4] = list.get(i).getEmail();
            row[5] = list.get(i).getAddressID();
            row[6] = list.get(i).getActive();
            row[7] = list.get(i).getCreateDate();
            row[8] = list.get(i).getCustomerAddress();
            row[9] = list.get(i).getCustomerDistrict();
            row[10] = list.get(i).getCustomerCity();
            row[11] = list.get(i).getCustomerCountry();
            row[12] = list.get(i).getCustomerPostalCode();
            row[13] = list.get(i).getCustomerPhone();
            row[14] = list.get(i).getLastUpdate();
```

```
72                 model.addRow(row);
            }
        }
```

Step 2

In **Customer_Utils.java**, define **refresh_controls()** method. It refresh object. Here's what it does:

1. Sets the location and title of the **CustomerForm**.
2. Populates a table (**frm.getJTCustomer()**) with data from the cus The data is retrieved by calling the **get_customer_list()** method, query to retrieve the data, and a string representing the value of this case).
3. Populates four combo boxes (**frm.getJCBCustomerI frm.getJCBStoreID()**, and **frm.getJCBActive()**) with data fr retrieved by calling the **populate_combobox()** method, passing ir combo box to populate, and the **CustomerForm**.
4. Populates another combo box (**frm.getJCBAddressID()**) with da retrieved by calling the **populate_combobox()** method, passing ir combo box to populate, and the **CustomerForm**.

If an **SQLException** is thrown at any point during this process, an erro error message.

```
1     public static void
2  refresh_controls(CustomerForm frm){
3
4  frm.setLocationRelativeTo(null);
5          frm.setTitle("CUSTOMER FORM");
6
7          //Shows the content of customer
8  table and populates combobox
9          try{
10             //Makes alternating color
11 for table rows
12
13 table_renderer(frm.getJTCustomer());
14
15             //Populates table
16             ArrayList<Customer> list =
17 get_customer_list(frm,
18 Query_Customer.get_sql_customer_joint()
19 + " ORDER BY customer_id", "none");
20             show_table_customer(frm,
21 list);
22
23             //Populates jcbCustomerID
24             String sql_cust_id =
25  "SELECT customer_id FROM customer
```

```
ORDER BY customer_id";

populate_combobox(sql_cust_id,
frm.getJCBCustomerID(), frm);

            //Populates jcbCustomerName
            String sql_cust_name =
 "SELECT DISTINCT CONCAT(first_name,'
', last_name) FROM customer ORDER BY
CONCAT(first_name,' ', last_name)";

populate_combobox(sql_cust_name,
frm.getJCBCustomerName(), frm);

            //Populates jcbStoreID
            String sql_store_id =
 "SELECT store_id FROM store ORDER BY
store_id";

populate_combobox(sql_store_id,
frm.getJCBStoreID(), frm);

            //Populates jcbActive
            String sql_act = "SELECT
DISTINCT active FROM customer ORDER BY
active";
            populate_combobox(sql_act,
frm.getJCBActive(), frm);

            //Populates jcbAddressID
            String sql_add_id = "SELECT
address_id FROM address ORDER BY
address_id";

populate_combobox(sql_add_id,
frm.getJCBAddressID(), frm);

        }catch (SQLException ex){

JOptionPane.showMessageDialog(frm,
ex.getMessage(),

 "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }
```

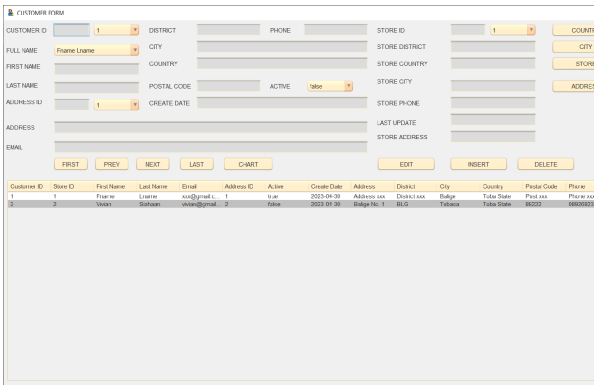


Figure 13.2 The content of joined **customer** table disp

Step 3

In **CustomerForm**'s default constructor, the **Customer_Utils.refresh_** populates the controls in the **CustomerForm** with data from a database from the **Customer_Utils** class.

The **this.setIconImage()** method sets the icon of the **Cu** **this.setDefaultCloseOperation(this.HIDE_ON_CLOSE)** method se **CustomerForm** to hide the form instead of exiting the application when th

```
1  package sakila;
2  import java.awt.Toolkit;
3  import java.awt.event.ActionEvent;
4  import java.awt.event.ActionListener;
5  import java.io.IOException;
6  import java.text.ParseException;
7  import java.util.logging.Level;
8  import java.util.logging.Logger;
9  import javax.swing.JButton;
10 import javax.swing.JComboBox;
11 import javax.swing.JLabel;
12 import javax.swing.JTable;
13 import javax.swing.JTextField;
14
15 public class CustomerForm extends javax.swing.JFrame {
16     public CustomerForm() {
17         initComponents();
18         Utility.setLookAndFeel(this);
19         Customer_Utils.refresh_controls(this);
20
21  this.setIconImage(Toolkit.getDefaultToolkit().getImage(getClass().
22 ;
23         this.setDefaultCloseOperation(this.HIDE_ON_CLOSE);
24     }
25     //...
26 }
```

Step 4	Run the project to see the content of joined **customer** table displayed in **jt** If you use the data from **Sakila** MySQL database available in the interne **customer** table displayed in **jtCustomer** as shown in Figure 13.3.

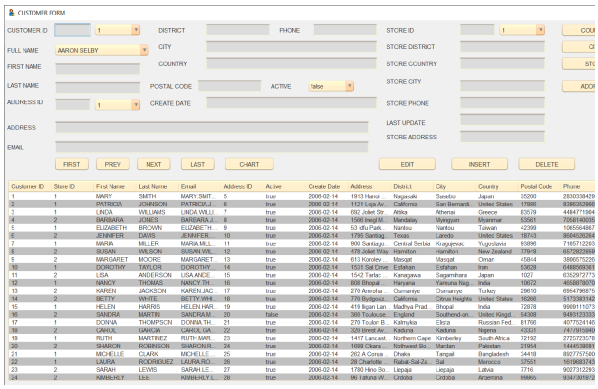


Figure 13.3 The the content of joined **customer** table in original **Saki** displayed in **jtCustomer**

DISPLAYING AND NAVIGATING DATA ROW BY ROW

DISPLAYING AND NAVIGATING DATA ROW BY ROW

Step 1	In **Customer_Utils**, define two new methods named **clear_store_co** **display_store_data()**. The first method, **clear_store_controls(Custome** takes a **CustomerForm** object as input and sets the text of various text form to empty strings. The second method, **display_store_data(CustomerForm frm, String s** takes a **CustomerForm** object, an SQL query string, and an object of typ It executes the SQL query and displays the resulting data row by row in t of the form. If the query returns no rows, the **clear_store_controls()** me to clear the form fields. If the query returns one or more rows, the value are set to the corresponding values from the first row of the res **find_combo_value_selected()** method is also called to select the appro the **jcbStoreID** JComboBox based on the value of **store_id** returned from

```
1     private static void clear_store_controls(CustomerForm frm){
2         frm.getJTFStoreID().setText("");
3         frm.getJTFStoreAddress().setText("");
4         frm.getJTFStoreDistrict().setText("");
5         frm.getJTFStorePhone().setText("");
6         frm.getJTFStoreCity().setText("");
7         frm.getJTFStoreCountry().setText("");
8     }
```

```
9
10     //Displays store data result row by row
11      private static <T> void display_store_data(CustomerForm frm,
12 String sql, T item){
13          try(Connection conn = getConnection()){
14              PreparedStatement ps = conn.prepareStatement(sql);
15              ps.setObject(1,item);
16              ResultSet rs = ps.executeQuery();
17
18              if (!rs.next()) {
19                  // no row found, clear the form fields
20                  clear_store_controls(frm);
21                  return;
22              }
23
24              do{
25
26 frm.getJTFStoreID().setText(String.valueOf(rs.getInt("store_id")));
27
28 frm.getJTFStoreAddress().setText(rs.getString("address"));
29
30 frm.getJTFStoreDistrict().setText(rs.getString("district"));
31
32 frm.getJTFStorePhone().setText(rs.getString("phone"));
33
34 frm.getJTFStoreCity().setText(rs.getString("city"));
35
36 frm.getJTFStoreCountry().setText(rs.getString("country"));
37
38                  // Determines item selected from jcbStoreID
39                  find_combo_value_selected(frm.getJCBStoreID(),
40 rs.getInt("store_id"));
41
42              }while(rs.next());
43
44              rs.close();
45              ps.close();
          }catch(SQLException ex){
              JOptionPane.showMessageDialog(frm, ex.getMessage(),
                      "ERROR",JOptionPane.ERROR_MESSAGE);
          }
      }
```

Step 2	Still in the same class, define another method named **jcbStore_handler()** handles the event when the user selects an item from the **jcbStoreID** com **CustomerForm**. It gets the selected item from the combo box a **display_store_data()** method, passing in the **CustomerForm** object query with a placeholder for the **store_id** parameter. The selected item i parameter value for the query to retrieve the store data that matches the s ID. The **display_store_data()** method then displays the store data on the f

```
    public static void
jcbStore_handler(CustomerForm frm) {
        Object item =
frm.getJCBStoreID().getSelectedItem();
        display_store_data(frm,
Query_Store.get_sql_store_joint() + "
WHERE store_id = ?", item);
    }
```

Step 3

In **CustomerForm**, double click on **jcbStoreID** combobox and define its c event handler as follows:

```
    private void
jcbStoreIDActionPerformed(java.awt.event.ActionEvent
evt) {
        Customer_Utils.jcbStore_handler(this);
    }
```

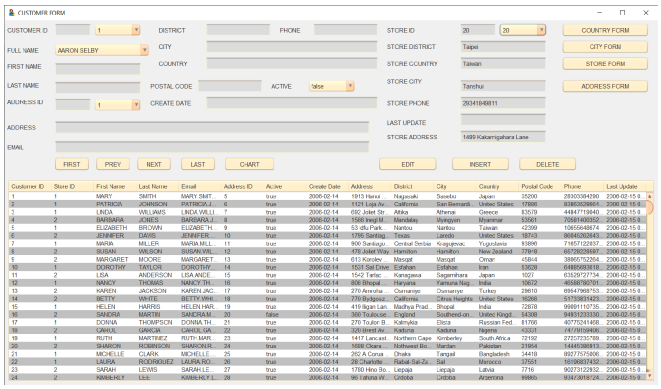


Figure 13.4 Displaying row by row the content of **store** table

Step 4

Run the project. Choose one of items in **jcbStoreID** combobox to see ro content of **store** table as shown in Figure 13.4.

Step 5

In **Customer_Utils**, define two new methods named **clear_address_co display_address_data()**. The **clear_address_controls()** method clears t for the address information on the customer form when it is called. **CustomerForm** object as a parameter and sets the text for each of the fields to an empty string.

The **display_address_data()** method displays the address data for a give customer form. It takes in a **CustomerForm** object, a SQL string to query for the relevant data, and an item object. The method retrieves the d database and sets the text for the relevant text fields on the form to displa also selects the item in the **JComboBox** for the address ID. If no data is

given item, the method calls the **clear_address_controls()** method to c fields.

```
    private static void clear_address_controls(CustomerForm frm){
        frm.getJTFAddressID().setText("");
        frm.getJTFAddress().setText("");
        frm.getJTFDistrict().setText("");
        frm.getJTFPhone().setText("");
        frm.getJTFPostalCode().setText("");
        frm.getJTFCity().setText("");
        frm.getJTFCountry().setText("");
    }

   //Displays address data result row by row
    private static <T> void display_address_data(CustomerForm frm,
String sql, T item){
        try(Connection conn = getConnection()){
            PreparedStatement ps = conn.prepareStatement(sql);
            ps.setObject(1,item);
            ResultSet rs = ps.executeQuery();

            if (!rs.next()) {
                // no row found, clear the form fields
                clear_address_controls(frm);
                return;
            }

            do{

frm.getJTFAddressID().setText(String.valueOf(rs.getInt("address_id"
                frm.getJTFAddress().setText(rs.getString("address")
                frm.getJTFDistrict().setText(rs.getString("district
                frm.getJTFPhone().setText(rs.getString("phone"));

frm.getJTFPostalCode().setText(rs.getString("postal_code"));
                frm.getJTFCity().setText(rs.getString("city"));
                frm.getJTFCountry().setText(rs.getString("country")

                // Determines item selected from jcbAddressID
                find_combo_value_selected(frm.getJCBAddressID(),
rs.getInt("address_id"));

            }while(rs.next());

            rs.close();
            ps.close();
        }catch(SQLException ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }
```

Step 6

Still in the same class, define another method named **jcbAddress_handle** method that handles the selection of an address from the address ID dro the customer form. It gets the selected item from the dropdown list, and t to the **display_address_data()** method along with a SQL query to retriev data associated with that ID.

The **display_address_data()** method then executes the SQL query, sets form fields to the retrieved data, and also sets the selected item in the dro match the retrieved ID. If no rows are found in the result set, it clears the

```
    public static void
jcbAddress_handler(CustomerForm frm) {
        Object item =
frm.getJCBAddressID().getSelectedItem();
        display_address_data(frm,
Query_Address.get_sql_address_joint() +
" WHERE address_id = ?", item);
    }
```

Step 7

In **CustomerForm**, double click on **jcbAddressID** combobox and corresponding event handler as follows:

```
    private void
jcbAddressIDActionPerformed(java.awt.event.ActionEvent
evt) {
        Customer_Utils.jcbStore_handler(this);
    }
```

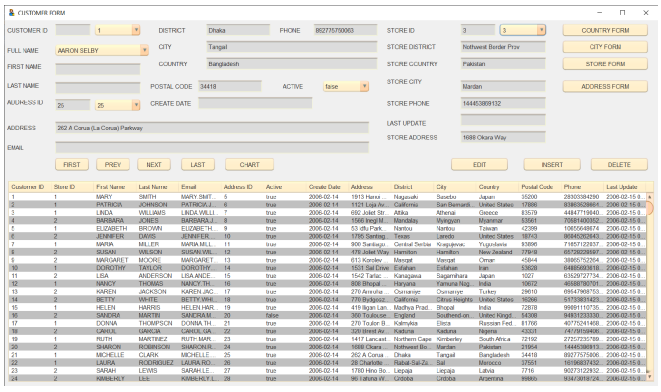


Figure 13.5 Displaying row by row the content of **address** tab

Step 8

Run the project. Choose one of items in **jcbAddressID** combobox to see ro content of **address** table as shown in Figure 13.5.

Step 9

In **Customer_Utils**, define two new methods named **clear_customer_cc** **display_customer_data()**.

- **clear_customer_controls(CustomerForm frm)**: This method cl fields in the **CustomerForm** object, which is a form used to disp data.
- **display_customer_data(CustomerForm frm, String sql, T** method executes a SQL query using the sql string parameter parameter as a placeholder value for a prepared statement. It r from the result set and displays it on the **CustomerForm** object b text of various text fields to the values retrieved from the databas the selected values of various combo boxes on the form based on data.

```
    private static void clear_customer_controls(CustomerForm frm){
        frm.getJTFCustomerID().setText("");
        frm.getJTFFirstName().setText("");
        frm.getJTFLastName().setText("");
        frm.getJTFEmail().setText("");
        frm.getJTFCreateDate().setText("");
        frm.getJTFLastUpdate().setText("");
    }

   //Displays customer data result row by row
    private static <T> void display_customer_data(CustomerForm frm,
T item){
        try(Connection conn = getConnection()){
            PreparedStatement ps = conn.prepareStatement(sql);
            ps.setObject(1,item);
            ResultSet rs = ps.executeQuery();

            if (!rs.next()) {
                // no row found, clear the form fields
                clear_address_controls(frm);
                return;
            }

            do{

frm.getJTFCustomerID().setText(String.valueOf(rs.getInt("customer_i
                frm.getJTFEmail().setText(rs.getString("email"));
                frm.getJTFFirstName().setText(rs.getString("first_n
                frm.getJTFLastName().setText(rs.getString("last_nam

frm.getJTFCreateDate().setText(String.valueOf(rs.getDate("create_da

frm.getJTFLastUpdate().setText(String.valueOf(rs.getTimestamp("last

                // Determines item selected from jcbCustomerID
                find_combo_value_selected(frm.getJCBCustomerID(),
rs.getInt("customer_id"));

```

```
39
40                  // Determines item selected from jcbCustomerName
41                  String full_name = rs.getString("first_name") + " "
42 rs.getString("last_name");
43                  find_combo_value_selected(frm.getJCBCustomerName(),
44
45                  // Determines item selected from jcbAddressID
46                  find_combo_value_selected(frm.getJCBAddressID(),
47 rs.getInt("address_id"));
49
50                  // Determines item selected from jcbStoreID
51                  find_combo_value_selected(frm.getJCBStoreID(),
52 rs.getInt("store_id"));
53
54                  // Determines item selected from jcbActive
55                  find_combo_value_selected(frm.getJCBActive(),
56 rs.getBoolean("active"));
57              }while(rs.next());
58
59              rs.close();
60              ps.close();
61          }catch(SQLException ex){
62              JOptionPane.showMessageDialog(frm, ex.getMessage(),
63                      "ERROR",JOptionPane.ERROR_MESSAGE);
64          }
      }
```

Step 10

Still in the same class, define another method named **jcbCustomer_ha** method is a handler for when the user selects an option from the **JCo** contains customer information. It takes in two parameters: a refe **CustomerForm** object and a reference to the **JComboBox** that was intera

The method first gets the selected item from the **JComboBox** and initia variable sql. Then, it checks which **JComboBox** was interacted with by c passed **JComboBox** object with the **JComboBox** objects in the **Customer** If it matches the **jcbCustomerID** object, then it sets the sql variable to t string that retrieves customer data by ID using **Query_Customer.get_** matches the **jcbCustomerName** object, then it sets the sql variable to t string that retrieves customer data by name using **Query_Customer.get_**

Finally, it calls the **display_customer_data()** method passing in the **Cu** object, the sql query string, and the selected item from the **JComboBox**. retrieves customer data from the database using the sql query string and item as a parameter, and populates the appropriate fields in the **Custome** with the retrieved data.

```
1       public static void
2   jcbCustomer_handler(CustomerForm frm,
```

```
JComboBox<String> jcb1) {
        Object item1 =
jcb1.getSelectedItem();
        String sql = "";
        if
(jcb1.equals(frm.getJCBCustomerID())) {
            sql =
Query_Customer.get_sql_id();
        } else if
(jcb1.equals(frm.getJCBCustomerName()))
{
            sql =
Query_Customer.get_sql_name();
        }

        display_customer_data(frm, sql,
item1);
    }
```

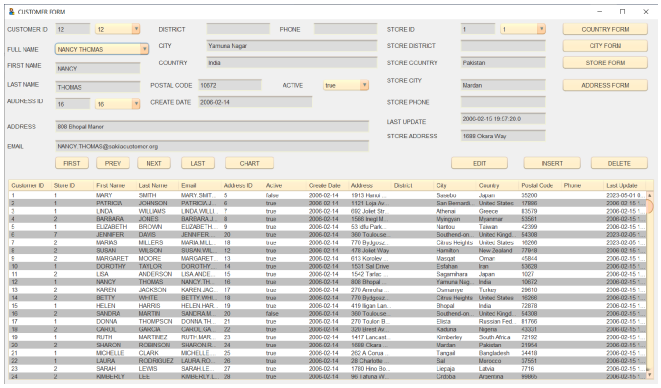


Figure 13.6 Displaying row by row the content of **customer** ta

Step 11

In **CustomerForm**, double click on **jcbCustomerID** and **jcbCus** comboboxes and define their corresponding event handlers as follows:

```
    private void
jcbCustomerIDActionPerformed(java.awt.event.ActionEvent
evt) {
        Customer_Utils.jcbCustomer_handler(this,
this.jcbCustomerID);
    }

    private void
jcbCustomerNameActionPerformed(java.awt.event.ActionEvent
evt) {
        Customer_Utils.jcbCustomer_handler(this,
this.jcbCustomerName);
    }
```

Step 12	Run the project. Choose one of items in **jcbCustomerID** and/or **jcbCus** combobox to see row by row the content of **customer** table as shown in Fi
Step 13	Define four navigating methods in **Customer_Utils** class. These are meth the user to navigate through the customer data displayed in the form. • **show_first_row(CustomerForm frm)**: displays the first row of data. It gets the first item of the **jcbCustomerID** combo box, w the customer IDs, and passes it to the **display_customer_data** display the customer data for that ID. • **show_last_row(CustomerForm frm)**: displays the last row of data. It gets the last item of the **jcbCustomerID** combo box, which customer IDs, and passes it to the **display_customer_data()** meth the customer data for that ID. • **show_prev_row(CustomerForm frm)**: displays the previous customer data. It decrements the **currentIndex** variable, which k the current row being displayed. If **currentIndex** is less than the I constant, which is the index of the first item in the **jcbCustomerI** then it resets **currentIndex** to FIRST_INDEX and returns. Othe the item at the **currentIndex** position in the **jcbCustomerID** co passes it to the **display_customer_data()** method to display the c for that ID. • **show_next_row(CustomerForm frm)**: displays the next row of data. It increments the currentIndex variable. If **currentIndex** is the index of the last item in the **jcbCustomerID** combo b **currentIndex** to the index of the last item and returns. Otherwis item at the **currentIndex** position in the **jcbCustomerID** cor passes it to the **display_customer_data()** method to display the c for that ID.

```
1     public static void show_first_row(CustomerForm frm){
2         String item =
3 String.valueOf(frm.getJCBCustomerID().getItemAt(FIRST_INDEX));
4         display_customer_data(frm, SQL_ID, item);
5         currentIndex = FIRST_INDEX;
6     }
7
8     public static void show_last_row(CustomerForm frm){
9         int endIndex = frm.getJCBCustomerID().getItemCount() -
10 1;
11        String item =
12 String.valueOf(frm.getJCBCustomerID().getItemAt(endIndex));
13        display_customer_data(frm, SQL_ID, item);
14        currentIndex = endIndex;
15    }
16
17    public static void show_prev_row(CustomerForm frm){
```

```
18          currentIndex--;
19          if(currentIndex < FIRST_INDEX){
20              currentIndex = FIRST_INDEX;
21              return;
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
```

```
        }
        String item =
String.valueOf(frm.getJCBCustomerID().getItemAt(currentIndex));
        display_customer_data(frm, SQL_ID, item);
    }

    public static void show_next_row(CustomerForm frm){
        int endIndex = frm.getJCBCustomerID().getItemCount() -
1;
        currentIndex++;
        if(currentIndex > endIndex){
            currentIndex = endIndex;
            return;
        }
        String item =
String.valueOf(frm.getJCBCustomerID().getItemAt(currentIndex));
        display_customer_data(frm, SQL_ID, item);
    }
```

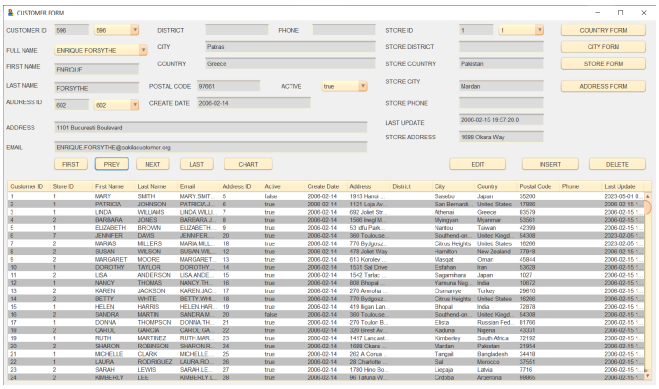


Figure 13.7 User clicks on one or more navigation buttons on customer form

Step 14

Then in **Cu**
their corres

```
    pri
jbFirst
evt) {

    }

    pri
jbPrevA
evt) {

    }
```

```
15      pri
jbNextA
evt) {

    }

    pri
jbLastA
evt) {

    }
```

These are
buttons in t
correspond
Customer_
row of data
calls refers
parameter t

Step 15

Run the pr
shown in Fi

Step 16

Define **mou**
single argu
"frm" argu
which throv

The metho
the "frm" c
(i.e., the in
dialog box a

If a row is
getConnec
selected ro
named **disp**
constant as

If a **SQLEx**
the error
JOptionPa

```
1      pub
2
3
4
```

	5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26
	`data.",`
	`String.`
	`0));`
	`Logger.`
	`display`
	`ex.getM`
	`stackTr`
	`}`

Step 17	Right click
	Define its e
	1 2 3
	`priv`
	`jtCustom`
	`evt) {`
	`Custome`
	`}`

Step 18	Run the pro correspond as shown in

F

UPDATING RECORD
UPDATING RECORD

Step 1	In **Customer_Utils** class, define a new method named **update_row_by_customer_id()**. The method takes as parameters a **CustomerForm** object **frm**, which is a reference to the form that called this method, an integer **cust_id**, which represents the **customer_id** of the row to be updated, an integer **store_id**, a string **fname** for the first name, a string **lname** for the last name, a string **email** for the email address, an integer **add_id** for the **address_id**, a boolean **active** to indicate whether the customer is active or not, and a Date object **cr_date** for the **create_date**. The method begins by obtaining a database connection using the **getConnection()** method, and then creates a **ResultSet** object rs and two prepared statement objects, **idPs** and **updatePS**. The first prepared statement **idPs** executes a SELECT statement to retrieve the row with the specified **customer_id** from the **customer** table. If the SELECT statement returns a result set, the method creates a new **Customer** object using the provided parameters and sets the values of the prepared statement object **updatePS** using the setter methods of the **Customer** object. Finally, the method executes the UPDATE statement using **updatePS.executeUpdate()** to update the row in the customer table. If the SELECT statement does not return a result set, the method displays an error message to the user using a JOptionPane.

If an **SQLException** or a **java.lang.NumberFormatException** occurs, the method logs the error and displays an error message to the user using a **JOptionPane**.

Finally, the method closes the **ResultSet** object, the prepared statement objects, and the database connection using their respective **close()** methods.

```
    //Updates row of data in customer tabel by customer_id
    public static void update_row_by_customer_id(CustomerForm
frm, int cust_id, int store_id, String fname, String lname,
        String email, int add_id, boolean active, Date cr_date)
throws SQLException{
        Connection conn = getConnection();
        ResultSet rs = null;
        String query_id = "SELECT customer_id FROM customer WHERE customer_id =
?";
        String update_query = """
            UPDATE customer SET store_id = ?, first_name = ?,
last_name = ?,
            email = ?, address_id = ?, active = ?, create_date =
? WHERE customer_id = ?""";
        try(PreparedStatement idPs =
conn.prepareStatement(query_id,

ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_READ_ONLY);
        PreparedStatement updatePS =
conn.prepareStatement(update_query,
                ResultSet.TYPE_SCROLL_SENSITIVE,
ResultSet.CONCUR_UPDATABLE))
        {
            idPs.setInt(1,cust_id);
            if(!idPs.execute()){
                String message = "Can't find customer_id " +
cust_id;

                JOptionPane.showMessageDialog(frm, message,
                    "ERROR",JOptionPane.ERROR_MESSAGE);
            } else{
                rs = idPs.getResultSet();
                rs.next();

                //Creates a Customer object using nine-params
constructor
                Customer obj = new Customer(cust_id, store_id,
fname, lname,
                    email, add_id, active, cr_date, new
Timestamp(System.currentTimeMillis()));
                updatePS.setInt(1, obj.getStoreID());
                updatePS.setString(2, obj.getFirstName());
                updatePS.setString(3, obj.getLastName());
                updatePS.setString(4, obj.getEmail());
```

```
                updatePS.setInt(5, obj.getAddressID());
                updatePS.setBoolean(6, obj.getActive());
                updatePS.setDate(7, obj.getCreateDate());
                updatePS.setInt(8, obj.getCustomerID());

                updatePS.executeUpdate();
                rs.close();
                updatePS.close();
                idPs.close();
                conn.close();
            }
        }catch(SQLException ex){

Logger.getLogger(CustomerForm.class.getName()).log(Level.SEVERE,
"Error updating customer data", ex);
            String message = "Error updating customer data: " +
ex.getMessage();
            String stackTrace =
Arrays.toString(ex.getStackTrace());
            JOptionPane.showMessageDialog(null, message + "\n\n"
+ stackTrace, "ERROR", JOptionPane.ERROR_MESSAGE);
        }catch(java.lang.NumberFormatException ex){

Logger.getLogger(CustomerForm.class.getName()).log(Level.SEVERE,
"Invalid Input", ex);
            String message = "Invalid Input: " +
ex.getMessage();
            String stackTrace =
Arrays.toString(ex.getStackTrace());
            JOptionPane.showMessageDialog(null, message + "\n\n"
+ stackTrace, "ERROR", JOptionPane.ERROR_MESSAGE);
        }
    }
```

Step 2

Then in the same class, define a new method **read_inputs()**. It reads user inputs from a form and validates them. It takes a **CustomerForm** object as an argument and returns a **HashMap** object with the validated inputs.

The form has several fields, such as customer ID, store ID, address ID, first name, last name, email, active status, and creation date. The method extracts the values from these fields and stores them in variables.

Then it validates the address ID, store ID, customer ID, first name, and last name fields to ensure they meet certain requirements. For example, the ID fields must be positive integers, while the name fields cannot be empty.

If any of the fields fail validation, the method throws an **IllegalArgumentException** with an appropriate error message.

Finally, the method creates a **HashMap** object and adds the validated inputs to it using string keys. It then returns this **HashMap** object.

```
    private static HashMap<String, String>
read_inputs(CustomerForm frm) {
        HashMap<String, String> input_data = new
HashMap<>();
        String cust_id =
String.valueOf(frm.getJCBCustomerID().getSelectedItem());
        String store_id =
String.valueOf(frm.getJCBStoreID().getSelectedItem());
        String add_id =
String.valueOf(frm.getJCBAddressID().getSelectedItem());
        String fname = frm.getJTFFirstName().getText();
        String lname = frm.getJTFLastName().getText();
        String email = frm.getJTFEmail().getText();
        String active =
String.valueOf(frm.getJCBActive().getSelectedItem());
        String cr_date =
frm.getJTFCreateDate().getText();

        // Validate user input
        int add_id_int = 0;
        try {
            add_id_int = Integer.parseInt(add_id);
            if (add_id_int <= 0) {
                throw new
IllegalArgumentException("Address ID cannot be negative
or zero");
            }
        } catch (NumberFormatException ex) {
            JOptionPane.showMessageDialog(frm, "Invalid
Address ID: " + add_id,
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        } catch (IllegalArgumentException ex) {
            JOptionPane.showMessageDialog(frm,
ex.getMessage(),
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        }

        int store_id_int = 0;
        try {
            store_id_int = Integer.parseInt(store_id);
            if (store_id_int <= 0) {
                throw new IllegalArgumentException("Store
ID cannot be negative or zero");
            }
```

```
        } catch (NumberFormatException ex) {
            JOptionPane.showMessageDialog(frm, "Invalid Store ID: " + store_id,
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        } catch (IllegalArgumentException ex) {
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        }

        int cust_id_int = 0;
        try {
            cust_id_int = Integer.parseInt(cust_id);
            if (cust_id_int <= 0) {
                throw new IllegalArgumentException("Customer ID cannot be negative or zero");
            }
        } catch (NumberFormatException ex) {
            JOptionPane.showMessageDialog(frm, "Invalid Manager Customer ID: " + cust_id,
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        } catch (IllegalArgumentException ex) {
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        }

        if (fname == null || fname.isEmpty()) {
            JOptionPane.showMessageDialog(frm, "First name cannot be empty",
            "Error", JOptionPane.ERROR_MESSAGE);
            throw new IllegalArgumentException("First name cannot be empty");
        }

        if (lname == null || lname.isEmpty()) {
            JOptionPane.showMessageDialog(frm, "Last name cannot be empty",
            "Error", JOptionPane.ERROR_MESSAGE);
            throw new IllegalArgumentException("Last name cannot be empty");
        }

        input_data.put("cust_id", cust_id);
        input_data.put("store_id", store_id);
        input_data.put("fname", fname);
        input_data.put("lname", lname);
        input_data.put("email", email);
```

```
        input_data.put("add_id", add_id);
        input_data.put("active", active);
        input_data.put("cr_date", cr_date);

        return input_data;
    }
```

Step 3

Still in the same class, define another method named **edit_actual()**. This method is used to handle the actual editing of the customer data in the database. It starts by calling the **read_inputs()** method to get the user's input data and validate it. Then, it parses the input data and converts it into the appropriate data types required by the **update_row_by_customer_id()** method.

Next, it creates a **java.util.Date** object by parsing the string value of the create date. This **java.util.Date** object is then converted into a **java.sql.Date** object which is needed by the **update_row_by_customer_id**() method to update the create date of the customer.

Finally, the **update_row_by_customer_id()** method is called to update the row in the database with the new customer data. If there is any SQL exception during this process, an error message is displayed in a dialog box.

After the update is successful, the **refresh_controls()** method is called to refresh all the objects on the form.

```
1   private static void
2  edit_actual(CustomerForm frm) throws
3  ParseException{
4          try{
5              HashMap<String, String>
6  input_data = read_inputs(frm);
7              int add_id =
8  Integer.parseInt(input_data.get("add_id"));
9              int store_id =
10 Integer.parseInt(input_data.get("store_id"));
11             int cust_id =
12 Integer.parseInt(input_data.get("cust_id"));
13             String fname =
14 input_data.get("fname");
15             String lname =
16 input_data.get("lname");
17             String email =
18 input_data.get("email");
19             String active =
20 input_data.get("active");
```

```
            String cr_date =
input_data.get("cr_date");

            boolean bool_active = false;

 if(active.equalsIgnoreCase("true")){
                bool_active = true;
            }

            SimpleDateFormat sdf = new
SimpleDateFormat("yyyy-MM-dd");
            java.util.Date date =
sdf.parse(cr_date);
            java.sql.Date sqlDate_create_date
= new java.sql.Date(date.getTime());

            update_row_by_customer_id(frm,
cust_id, store_id, fname, lname, email,
add_id, bool_active, sqlDate_create_date);

            //Refreshes all objects on form
            refresh_controls(frm);

        }catch(SQLException ex){

JOptionPane.showMessageDialog(frm,
ex.getMessage(),

 "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }
```

Step 4

Lastly, define two new methods named **enable_controls()** and **edit_handler()**. The **edit_handler()** method is called when the user clicks on the "EDIT" button on a customer form. The purpose of this method is to toggle the form between an edit mode and a view mode.

When the user clicks on the "EDIT" button for the first time, the button text is changed to "CONFIRM" and the controls on the form are disabled by calling the **enable_controls()** method with the state parameter set to false. This prevents the user from editing any of the form fields while in edit mode.
When the user clicks on the "CONFIRM" button, the **edit_actual()** method is called to perform the actual editing of the customer data in the database. Once the editing is complete, the button text is changed back to "EDIT" and the **enable_controls()** method is called again with the state parameter set to true to re-enable the controls on the form.

If an exception occurs while calling the **edit_actual()** method, a **JOptionPane** error message is displayed with the error message as its content.

```
    private static void enable_controls(boolean state,
CustomerForm frm){
        frm.getJBFirst().setEnabled(state);
        frm.getJBPrev().setEnabled(state);
        frm.getJBNext().setEnabled(state);
        frm.getJBLast().setEnabled(state);
        frm.getJBInsert().setEnabled(state);
        frm.getJBDelete().setEnabled(state);
        frm.getJTFAddressID().setEnabled(state);
        frm.getJTFStoreID().setEnabled(state);
        frm.getJTFAddress().setEnabled(state);
        frm.getJTFDistrict().setEnabled(state);
        frm.getJTFPhone().setEnabled(state);
        frm.getJTFPostalCode().setEnabled(state);
        frm.getJTFCity().setEnabled(state);
        frm.getJTFCountry().setEnabled(state);
        frm.getJTFStoreAddress().setEnabled(state);
        frm.getJTFStoreDistrict().setEnabled(state);
        frm.getJTFStorePhone().setEnabled(state);
        frm.getJTFStoreCity().setEnabled(state);
        frm.getJTFStoreCountry().setEnabled(state);
    }

    public static void edit_handler(CustomerForm frm){
        if(frm.getJBEdit().getText().equals("EDIT")){
            frm.getJBEdit().setText("CONFIRM");

            // Disables controls
            enable_controls(false, frm);
        }

        else {
            try {
                frm.getJBEdit().setText("EDIT");

                // Actual editing
                edit_actual(frm);

                //Enables controls
                enable_controls(true, frm);
            } catch (ParseException ex) {

Logger.getLogger(Customer_Utils.class.getName()).log(Level.SEVERE,
null, ex);
            }
        }
    }
```

Step 5	Run the project. Choose **customer_id** using **jcbCustomerID** or **jcbCustomerName** combobox. Or, you can choose one of rows in **jtCustomer** (in this case, **customer_id = 5**). Then, click on EDIT button as shown in Figure 13.9. Edit any field you want. Then, click on CONFIRM button. The edited row had been saved into **customer** table as shown in Figure 13.10.

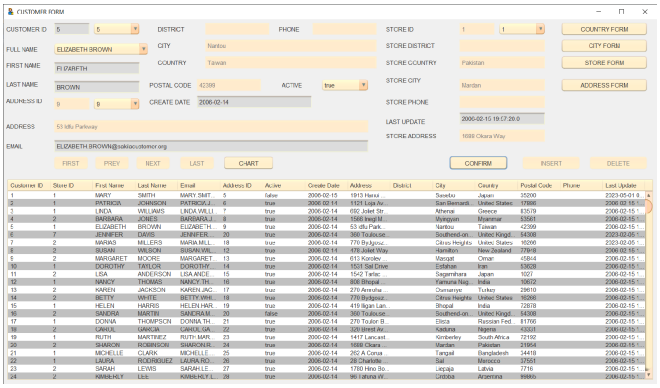


Figure 13.9 The customer form is in editing state

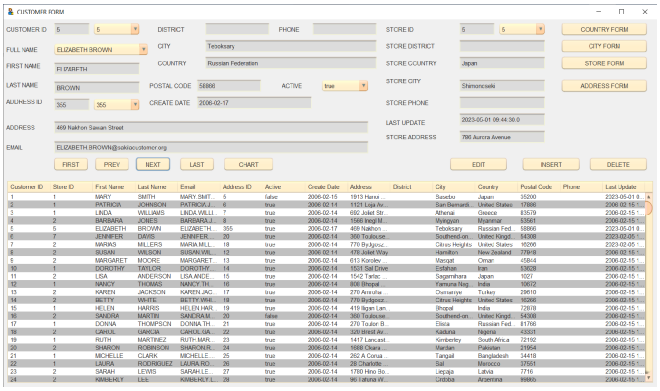


Figure 13.10 The edited row had been saved into database

UPDATING RECORD DIRECTLY ON JTABLE

UPDATING RECORD DIRECTLY ON JTABLE

Step 1	In **Customer_Utility** class, define a new metho **edit_database_from_jtable()**. This method is used to handle updates **JTable** in the customer form. It is triggered when the user updates a cell i

The method first checks if the type of the **TableModelEvent** is UPDATE. I the row index of the updated cell and retrieves the values of the updated r table model. These values are used to update the corresponding row in t using the **update_row_by_customer_id()** method.

If an SQL exception occurs during the update, a message dialog is shown with information about the error. Otherwise, the **refresh_controls()** met to update all the objects on the form with the new data from the database.

```
    public static void edit_database_from_jtable(TableModelEvent
e, CustomerForm frm) throws ParseException{
        if (e.getType() == TableModelEvent.UPDATE) {
            int row = e.getFirstRow();
            TableModel model = (TableModel)e.getSource();
            int cust_id =
Integer.parseInt(String.valueOf(model.getValueAt(row, 0)));
            int store_id =
Integer.parseInt(String.valueOf(model.getValueAt(row, 1)));
            int add_id =
Integer.parseInt(String.valueOf(model.getValueAt(row, 5)));
            String fname = String.valueOf(model.getValueAt(row,
2));
            String lname = String.valueOf(model.getValueAt(row,
3));
            String email = String.valueOf(model.getValueAt(row,
4));
            String active = String.valueOf(model.getValueAt(row,
6));
            String cr_date =
String.valueOf(model.getValueAt(row, 7));

            boolean bool_active = false;
            if(active.equalsIgnoreCase("true")){
                bool_active = true;
            }

            SimpleDateFormat sdf = new SimpleDateFormat("yyyy-
MM-dd");
            java.util.Date date = sdf.parse(cr_date);
            java.sql.Date sqlDate_create_date = new
java.sql.Date(date.getTime());

            try{
                update_row_by_customer_id(frm, cust_id,
store_id, fname, lname, email, add_id, bool_active,
sqlDate_create_date);

                //Refreshes all objects on form
                refresh_controls(frm);

            } catch (SQLIntegrityConstraintViolationException
ex) {
```

```
44
45 Logger.getLogger(CustomerForm.class.getName()).log(Level.SEVERE,
46 "Duplicate entry", ex);
47                 JOptionPane.showMessageDialog(frm, "Error:
48 Duplicate entry\n" + ex.getMessage());
49             } catch (SQLSyntaxErrorException ex) {
50
51 Logger.getLogger(CustomerForm.class.getName()).log(Level.SEVERE,
   "Invalid SQL syntax", ex);
                JOptionPane.showMessageDialog(frm, "Error:
Invalid SQL syntax\n" + ex.getMessage());
            } catch (SQLException ex) {

Logger.getLogger(CustomerForm.class.getName()).log(Level.SEVERE,
"Database error", ex);
                JOptionPane.showMessageDialog(frm, "Error:
Database error\n" + ex.getMessage());
            }
        }
    }
```

Step 2

Create a new public class named **CustomerTableModelListener**. It imp
TableModelListener interface. It listens for changes to the data in a
updates the database accordingly using the **edit_database_from_jtab**
from the **Customer_Utils** class.

The constructor takes a **JTable** and a **CustomerForm** object as parameter
used to call the **edit_database_from_jtable()** method.

The **tableChanged()** method is called whenever a change is made to th
catches any **ParseException** thrown by the **edit_database_from_jtab**
and logs it. If the table cell editor is not null, it stops cell editing.

```
1  package sakila;
2  import java.text.ParseException;
3  import java.util.logging.Level;
4  import java.util.logging.Logger;
5  import javax.swing.event.TableModelEvent;
6  import javax.swing.event.TableModelListener;
7  import javax.swing.JTable;
8
9  public class CustomerTableModelListener implements TableModelListen
10     private final JTable jt;
11     private final CustomerForm frm;
12
13     public CustomerTableModelListener(JTable jt, CustomerForm frm)
14         this.jt = jt;
15         this.frm = frm;
16     }
```

```
17
18      @Override
19      public void tableChanged(TableModelEvent e) {
20          try {
21              Customer_Utils.edit_database_from_jtable(e, frm);
22          } catch (ParseException ex) {
23
24  Logger.getLogger(CustomerTableModelListener.class.getName()).log(Le
25  null, ex);
26          }
27
28          if (jt.getCellEditor() != null) {
29              jt.getCellEditor().stopCellEditing();
30          }
31      }
32  }
```

Step 3

Right click on **jtCustomer**. Then, choose **Events > Mouse > mouseClic** its event handler:

```
1      private void jtCustomerMouseClicked(java.awt.event.MouseEvent ev
2          // instantiate CustomerTableModelListener and add it as a li
3  to the table model
4          CustomerTableModelListener tableModelListener = new
5  CustomerTableModelListener(this.getJTCustomer(), this);
6
7   this.getJTCustomer().getModel().addTableModelListener(tableModelLis
    ;
      }
```

It is is an event handler that is triggered when the user clicks on a cell i **jtCustomer**. It creates a new instance of the **CustomerTableModelLi** and adds it as a listener to the table model. This **CustomerTableModelListener** to receive events when the user edits table, which it can then use to update the database t **edit_database_from_jtable()** method in the **Customer_Utils** class.

Step 4

Run the project. Click on any cell in second column to seventh column i **jtCustomer** that you want to edit. Then, change it. Then, click any where corresponding cell. The edited data had been saved into database.

INSERTING NEW RECORD
INSERTING NEW RECORD

Step 1

In **Customer_Utils** class, define a method named **insert_row()**. It inserts a new row into the **customer** table in the database with the data from the input fields of the **CustomerForm** form.

First, the method reads the inputs from the form and parses the relevant data. Then, it creates a new **Customer** object using the parsed data and the current date and time.

Next, it creates an SQL insert statement with placeholders for the data. It then sets the values of the placeholders using the data from the **Customer** object.

Finally, it executes the SQL insert statement using a **PreparedStatement**, which is a type of Statement that allows for parameterized queries, which helps prevent SQL injection attacks. If there is a SQL exception, it logs the error and displays a message dialog with the error message.

```
1   //Inserts new row into customer table
2   private static void insert_row(CustomerForm frm) throws
3   SQLException, ParseException{
4           HashMap<String, String> input_data = read_inputs(frm);
5           int add_id = Integer.parseInt(input_data.get("add_id"));
6           int store_id =
7   Integer.parseInt(input_data.get("store_id"));
8           String fname = input_data.get("fname");
9           String lname = input_data.get("lname");
10          String email = input_data.get("email");
11          String active = input_data.get("active");
12          String cr_date = input_data.get("cr_date");
13
14          boolean bool_active = false;
15          if(active.equalsIgnoreCase("true")){
16              bool_active = true;
17          }
18
19          // SQL insert statement
20          String sql = """
21              INSERT INTO customer(store_id, first_name,
22  last_name,
23             email, address_id, active, create_date) VALUES(?, ?,
24  ?, ?, ?, ?, ?)""";
25
26          try(Connection conn = getConnection();
27           PreparedStatement pstmt = conn.prepareStatement(sql)){
28
29              //Creates a Customer object eight-params constructor
30              Customer obj = new Customer(store_id, fname, lname,
31  email, add_id, bool_active, new
32  Date(Calendar.getInstance().getTime().getTime()), new
33  Timestamp(System.currentTimeMillis()));
```

```
            pstmt.setInt(1,obj.getStoreID());
            pstmt.setString(2,obj.getFirstName());
            pstmt.setString(3,obj.getLastName());
            pstmt.setString(4,obj.getEmail());
            pstmt.setInt(5, obj.getAddressID());
            pstmt.setBoolean(6, obj.getActive());
            pstmt.setDate(7, obj.getCreateDate());

            //Executes the sql insert statement
            pstmt.executeUpdate();
        } catch (SQLException ex) {

Logger.getLogger(CustomerForm.class.getName()).log(Level.SEVERE,
"Database error", ex);
            JOptionPane.showMessageDialog(frm, "Error: Database
error\n" + ex.getMessage());
        }
    }
```

Step 2

Still in **Customer_Utils.java**, define **insert_actual()** and **insert_handler()** methods. The **insert_handler()** method is called when the user clicks on the "INSERT" button in the user interface.

If the text of the button is "INSERT", the method changes the button text to "CONFIRM" and disables the "jbEdit" button and other controls. It also clears the input controls.

If the button text is "CONFIRM", the method calls the **insert_actual()** method to insert a new customer record into the database. If the insertion is successful, the method enables the "jbEdit" button and input controls, and sets the button text back to "INSERT". If there is an exception thrown during insertion, it displays an error message.

```
    private static void
insert_actual(CustomerForm frm) throws
ParseException{
        try{
            insert_row(frm);

            //Refreshes table and comboboxes
            refresh_controls(frm);

        }catch(SQLException ex){
            JOptionPane.showMessageDialog(frm,
ex.getMessage(),

 "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }
```

```
17
18      public static void
19  insert_handler(CustomerForm frm){
20
21   if(frm.getJBInsert().getText().equals("INSERT")
22  ){
23
24  frm.getJBInsert().setText("CONFIRM");
25
26              //Disables jbEdit
27              frm.getJBEdit().setEnabled(false);
28
29              // Disables controls
30              enable_controls(false, frm);
31
32  frm.getJCBCustomerID().setEnabled(false);
33
34  frm.getJCBCustomerName().setEnabled(false);
35
36  frm.getJTFCreateDate().setEnabled(false);
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
```

```
            // Clears controls
            clear_customer_controls(frm);

            // Enables
            frm.getJBInsert().setEnabled(true);
        }

        else {
            frm.getJBInsert().setText("INSERT");

            try {
                // Actual insertion
                insert_actual(frm);
            } catch (ParseException ex) {

Logger.getLogger(Customer_Utils.class.getName()).log(Level.SEVERE,
null, ex);
            }

            //Enables jbEdit
            frm.getJBEdit().setEnabled(true);

            //Enables controls
            enable_controls(true, frm);
            frm.getJCBCustomerID().setEnabled(true);
            frm.getJCBCustomerName().setEnabled(true);
            frm.getJTFCreateDate().setEnabled(true);
        }
    }
```

Step 3	In **Custo** to create
	
	`1 p`
	`2 jbIns`
	`3 evt)`
	`}`

Step 4	Run the the state as show

Figure
C

Figu

Then, fi
save the
Figure 1

DELETING RECORD
DELETING RECORD

Step 1	Then in **Customer_Utils** class, define **delete_handler()** method. This r responsible for handling the deletion of a row in the **customer** table. It prompting the user with a confirmation message asking whether they re to delete the row. If the user clicks the "Yes" button, the method proceeds the row using a SQL DELETE statement with a WHERE clause that ma customer ID of the selected customer in the customer ID combobox. The method uses a prepared statement to avoid SQL injection attacks, good practice. After deleting the row, the method calls the **refresh_c**

method to update the table and comboboxes with the latest data.

If an SQL exception occurs during the deletion process, the method di error message dialog to the user with the error message obtained exception.

```
    public static void delete_handler(CustomerForm frm){
        int dialogButton = JOptionPane.YES_NO_OPTION;
        int cust_id =
Integer.parseInt(String.valueOf(frm.getJCBCustomerID().getSelectedI

        String message = String.format("Are you sure you want to de
row Customer ID: %d)", cust_id);
        int answer = JOptionPane.showConfirmDialog(frm, message, "D
ROW OF DATA", dialogButton);

        if(answer == JOptionPane.YES_OPTION){
            String query = """
                DELETE FROM customer WHERE customer_id = ?""";
            try(Connection conn = getConnection();
                PreparedStatement ps = conn.prepareStatement(query)
                // Use PreparedStatement to avoid SQL injection att
                ps.setInt(1, cust_id);
                ps.executeUpdate();

                // Refresh table and comboboxes
                refresh_controls(frm);

            } catch (SQLException ex){
                JOptionPane.showMessageDialog(frm, ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
            }
        }
    }
```

Step 2

In **CustomerForm.java**, double click on DELETE button to generate listener:

```
    private void
jbDeleteActionPerformed(java.awt.event.ActionEvent
evt) {
        Customer_Utils.delete_handler(this);
    }
```

Step 3

Run the project. Choose **customer_id** using **jcbCustome** **jcbCustomerName** combobox. Then, Click on DELETE button. The corre row of data had been deleted from database.

PLOTTING CHART
PLOTTING CHART

Step 1	Create a new **JFrame** and save it as **Charts_Customer.java**.
Step 2	In **Charts_Customer.java**, add six **JPanels** and set their corresponding **Va** **Name** as **jPanel1**, **jPanel2**, **jPanel3**, **jPanel4**, **jPanel5**, and **jPanel6**. Th getter method for each object as follows:
Step 3	In **Customer_Utils** class, define six new methods. These are six static m that draw pie charts to represent the distribution of customers in a reta

```
1       //Getter method for jPanel1
2       public JPanel getJPanel1(){
3           return this.jPanel1;
4       }
5
6       //Getter method for jPanel2
7       public JPanel getJPanel2(){
8           return this.jPanel2;
9       }
10
11      //Getter method for jPanel3
12      public JPanel getJPanel3(){
13          return this.jPanel3;
14      }
15
16      //Getter method for jPanel4
17      public JPanel getJPanel4(){
18          return this.jPanel4;
19      }
20
21      //Getter method for jPanel5
22      public JPanel getJPanel5(){
23          return this.jPanel5;
24      }
25
26      //Getter method for jPanel6
27      public JPanel getJPanel6(){
28          return this.jPanel6;
29      }
```

chain based on their location. Each method takes two parameters: an inst
the **Charts_Customer** class and a **JPanel** component that will hold the pie

The first three methods draw pie charts based on the customer's location
country, city, and district level. They use the **create_pie_dataset()** me
create a **DefaultPieDataset** object that contains the data for the pie
obtained by executing SQL queries to retrieve the number of custom
country, city, and district. The pie chart is then drawn usir
draw_piechart_with_dataset() method.

The second set of three methods draws pie charts based on the location
store where the customer made their purchases, at the country, city, and
level. These methods follow the same approach as the first set, but they u
queries that retrieve the number of customers by store country, city, and d

The **setPreferredSize()** method is called on the **JPanel** component to
that the pie chart is displayed with the desired size. The title of each pie
passed as a parameter to the **draw_piechart_with_dataset()** method.

```
1     private static void
2  draw_pie_chart_customer_by_country(Charts_Customer frm, JPanel jp){
3          jp.setPreferredSize(new Dimension(jp.getWidth(),
4  jp.getHeight()));
5          DefaultPieDataset dataset =
6  create_pie_dataset(Query_Customer.get_sql_customer_country_dist(),
7  "number_of_customers", "customer_country");
8
9          //Draws piechart customer distribution by country
10         draw_piechart_with_dataset(frm, jp, dataset, "TOP 10 CUSTOM
11 DISTRIBUTION BY COUNTRY");
12     }
13
14     private static void draw_pie_chart_customer_by_city(Charts_Cust
15 frm, JPanel jp){
16         jp.setPreferredSize(new Dimension(jp.getWidth(),
17 jp.getHeight()));
18         DefaultPieDataset dataset =
19 create_pie_dataset(Query_Customer.get_sql_customer_city_dist(),
20 "number_of_customers", "customer_city");
21
22         //Draws piechart customer distribution by city
23         draw_piechart_with_dataset(frm, jp, dataset, "TOP 10 CUSTOM
24 DISTRIBUTION BY CITY");
25     }
26
27     private static void
28 draw_pie_chart_customer_by_district(Charts_Customer frm, JPanel jp)
29         jp.setPreferredSize(new Dimension(jp.getWidth(),
30 jp.getHeight()));
31
32
```

```
        DefaultPieDataset dataset =
create_pie_dataset(Query_Customer.get_sql_customer_district_dist(),
"number_of_customers", "customer_district");

        //Draws piechart customer distribution by district
        draw_piechart_with_dataset(frm, jp, dataset, "TOP 10 CUSTOM
DISTRIBUTION BY DISTRICT");
    }

    private static void
draw_pie_chart_customer_store_by_country(Charts_Customer frm, JPane
        jp.setPreferredSize(new Dimension(jp.getWidth(),
jp.getHeight()));
        DefaultPieDataset dataset =
create_pie_dataset(Query_Customer.get_sql_customer_store_country_di
"number_of_customers", "store_country");

        //Draws piechart customer distribution by store country
        draw_piechart_with_dataset(frm, jp, dataset, "TOP 10 CUSTOM
DISTRIBUTION BY STORE COUNTRY");
    }

    private static void
draw_pie_chart_customer_store_by_city(Charts_Customer frm, JPanel j
        jp.setPreferredSize(new Dimension(jp.getWidth(),
jp.getHeight()));
        DefaultPieDataset dataset =
create_pie_dataset(Query_Customer.get_sql_customer_store_city_dist(
"number_of_customers", "store_city");

        //Draws piechart customer distribution by store city
        draw_piechart_with_dataset(frm, jp, dataset, "TOP 10 CUSTOM
DISTRIBUTION BY STORE CITY");
    }

    private static void
draw_pie_chart_customer_store_by_district(Charts_Customer frm, JPan
{
        jp.setPreferredSize(new Dimension(jp.getWidth(),
jp.getHeight()));
        DefaultPieDataset dataset =
create_pie_dataset(Query_Customer.get_sql_customer_store_district_d
"number_of_customers", "store_district");

        //Draws piechart customer distribution by store district
        draw_piechart_with_dataset(frm, jp, dataset, "TOP 10 CUSTOM
DISTRIBUTION BY STORE DISTRICT");
    }
```

Step 4

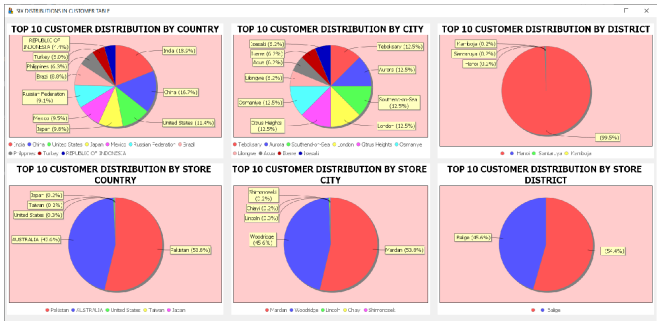
In **Customer_Utils** class, define a new method named **jbchart_handler()**

```
    public static void
jbchart_handler(Charts_Customer frm){
        //Draws piechart customer distribution
by country

draw_pie_chart_customer_by_country(frm,
frm.getJPanel1());

        //Draws piechart customer distribution
by city
        draw_pie_chart_customer_by_city(frm,
frm.getJPanel2());

        //Draws piechart customer distribution
by district

draw_pie_chart_customer_by_district(frm,
frm.getJPanel3());

        //Draws piechart customer distribution
by store country

draw_pie_chart_customer_store_by_country(frm,
frm.getJPanel4());

        //Draws piechart customer distribution
by store city

draw_pie_chart_customer_store_by_city(frm,
frm.getJPanel5());

        //Draws piechart customer distribution
by store district

draw_pie_chart_customer_store_by_district(frm,
frm.getJPanel6());
    }
```

Step 5

In **CustomerForm**, double click on **jbChart** button to define its event liste

```
    private void
jbChartActionPerformed(java.awt.event.ActionEvent
evt) {
        Charts_Customer frm1 = new
Charts_Customer();
        frm1.setLocationRelativeTo(null);
        frm1.setTitle("SIX DISTRIBUTIONS IN
CUSTOMER TABLE");
        frm1.setVisible(true);
        Customer_Utils.jbchart_handler(frm1);
    }
```

Step 6	Run the project. Click on CHART button on the form. You will see the six displayed on the panels as shown in Figure 13.13.  **Figure 13.13** The top 10 customer distribution by country, the top 10 cus distribution by city, the top 10 customer distribution by district, the top customer distribution by store country, the top 10 customer distribution b city, and the top 10 customer distribution by store district

This is the full version of **Customer_Utils.java**:

```
package sakila;
import java.awt.Dimension;
import java.util.logging.Level;
import java.util.logging.Logger;
import java.sql.*;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Calendar;
import java.util.HashMap;
import java.util.Objects;
import javax.swing.JComboBox;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.event.TableModelEvent;
import javax.swing.table.DefaultTableModel;
import javax.swing.table.TableModel;
import org.jfree.data.category.DefaultCategoryDataset;
import org.jfree.data.general.DefaultPieDataset;
import static sakila.Utility.create_bar_dataset;
import static sakila.Utility.draw_barchart_with_dataset;

public class Customer_Utils extends Utility{
    public static final int FIRST_INDEX = 0;
    public static final int INVALID_INDEX = -1;
```

```
    private static int currentIndex = FIRST_INDEX;
    private static final String SQL_ID = Query_Customer.get_sql_id();

    //Creates customer table
    public static void create_customer_table() {
        try (Connection conn = getConnection()) {
            Statement stmt = conn.createStatement();
            stmt.addBatch(Query_Customer.get_sql_customer());
            stmt.executeBatch();

            String message = String.format("Successfully creates customer
table");
            JOptionPane.showMessageDialog(null, message,
            "INFORMATION",JOptionPane.INFORMATION_MESSAGE);

        } catch (SQLException ex) {
            JOptionPane.showMessageDialog(null, ex.getMessage(),
                "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    //Populates customer table with some rows of data
    public static void populate_customer_table(){
        try(Connection conn = getConnection()){
            String sql = """
                INSERT INTO customer(customer_id, store_id, first_name,
last_name,
                    email, address_id, active, create_date, last_update)
                VALUES(?, ?, ?, ?, ?, ?, ?, ?, ?)""";

            //Creates a new Customer class with default constructor
            PreparedStatement ps1 = conn.prepareStatement(sql);
            Customer obj1 = new Customer();
            ps1.setInt(1,obj1.getCustomerID());
            ps1.setInt(2,obj1.getStoreID());
            ps1.setString(3,obj1.getFirstName());
            ps1.setString(4,obj1.getLastName());
            ps1.setString(5,obj1.getEmail());
            ps1.setInt(6,obj1.getAddressID());
            ps1.setBoolean(7,obj1.getActive());
            ps1.setDate(8,obj1.getCreateDate());
            ps1.setTimestamp(9,obj1.getLastUpdate());

            // Creates a new Customer class with nine-params constructor
            PreparedStatement ps2 = conn.prepareStatement(sql);
            Customer obj2 = new Customer(2, 2, "Vivian", "Siahaan",
"vivian@gmail.com",
                    2, false, new
Date(Calendar.getInstance().getTime().getTime()),
                    new Timestamp(System.currentTimeMillis()));
            ps2.setInt(1,obj2.getCustomerID());
            ps2.setInt(2,obj2.getStoreID());
```

```
            ps2.setString(3,obj2.getFirstName());
            ps2.setString(4,obj2.getLastName());
            ps2.setString(5,obj2.getEmail());
            ps2.setInt(6,obj2.getAddressID());
            ps2.setBoolean(7,obj2.getActive());
            ps2.setDate(8,obj2.getCreateDate());
            ps2.setTimestamp(9,obj2.getLastUpdate());

            ps1.executeUpdate();
            ps2.executeUpdate();

        }catch(SQLException ex){
            JOptionPane.showMessageDialog(null, ex.getMessage(),
                "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    //Reads the content of customer table
    public static void read_customer_table(){
        try(Connection conn = getConnection()){
            Statement stmt = conn.createStatement();
            ResultSet rs = stmt.executeQuery("SELECT * FROM customer");

            while(rs.next()){
                int cust_id = rs.getInt("customer_id");
                int store_id = rs.getInt("store_id");
                int add_id = rs.getInt("address_id");
                String fname = rs.getString("first_name");
                String lname = rs.getString("last_name");
                String email = rs.getString("email");
                boolean active = rs.getBoolean("active");
                Date cr_date = rs.getDate("create_date");
                Timestamp lu = rs.getTimestamp("last_update");

                //Creates a Customer object using nine-params constructor
                Customer obj = new  Customer(cust_id, store_id, fname, lname,
email, add_id, active, cr_date, lu);
                System.out.println(obj);
            }
            rs.close();
            stmt.close();

        }catch(SQLException ex){
            JOptionPane.showMessageDialog(null, ex.getMessage(),
                "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    private static ArrayList<Customer> get_customer_list(CustomerForm frm,
String sql, String item){
        ArrayList<Customer> list = new ArrayList<>();

        try(Connection conn = getConnection();
```

```
            PreparedStatement ps = conn.prepareStatement(sql)){
            if (item.equalsIgnoreCase("none")==false) {
                 ps.setString(1,item);
            }
            ResultSet rs = ps.executeQuery();
            Customer obj;
            while(rs.next()){
                //Using fifteen-params constructor
                obj = new Customer(rs.getInt("customer_id"),
                        rs.getInt("store_id"),
                        rs.getString("first_name"),
                        rs.getString("last_name"),
                        rs.getString("email"),
                        rs.getInt("address_id"),
                        rs.getBoolean("active"),
                        rs.getDate("create_date"),
                        rs.getTimestamp("last_update"),
                        rs.getString("address"),
                        rs.getString("district"),
                        rs.getString("city"),
                        rs.getString("country"),
                        rs.getString("postal_code"),
                        rs.getString("phone"));

                list.add(obj);
            }
        }catch (SQLException ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
                "ERROR",JOptionPane.ERROR_MESSAGE);
        }
        return list;
    }

    private static void show_table_customer(CustomerForm frm,
ArrayList<Customer> list) throws SQLException{
        DefaultTableModel model = new DefaultTableModel(0,0);

        String header[] = {"Customer ID", "Store ID", "First Name", "Last
Name", "Email",
            "Address ID", "Active", "Create Date", "Address", "District",
            "City", "Country", "Postal Code", "Phone", "Last Update"};

        model.setColumnIdentifiers(set_column_header(frm.getJTCustomer(),
header));
        frm.getJTCustomer().setModel(model);

        Object[] row = new Object[15];

        for(int i=0; i<list.size(); i++){
            row[0] = list.get(i).getCustomerID();
            row[1] = list.get(i).getStoreID();
            row[2] = list.get(i).getFirstName();
            row[3] = list.get(i).getLastName();
            row[4] = list.get(i).getEmail();
```

```
            row[5] = list.get(i).getAddressID();
            row[6] = list.get(i).getActive();
            row[7] = list.get(i).getCreateDate();
            row[8] = list.get(i).getCustomerAddress();
            row[9] = list.get(i).getCustomerDistrict();
            row[10] = list.get(i).getCustomerCity();
            row[11] = list.get(i).getCustomerCountry();
            row[12] = list.get(i).getCustomerPostalCode();
            row[13] = list.get(i).getCustomerPhone();
            row[14] = list.get(i).getLastUpdate();

            model.addRow(row);
        }
    }

    public static void refresh_controls(CustomerForm frm){
        frm.setLocationRelativeTo(null);
        frm.setTitle("CUSTOMER FORM");

        //Shows the content of customer table and populates combobox
        try{
            //Makes alternating color for table rows
            table_renderer(frm.getJTCustomer());

            //Populates table
            ArrayList<Customer> list = get_customer_list(frm,
Query_Customer.get_sql_customer_joint() + " ORDER BY customer_id", "none");
            show_table_customer(frm, list);

            //Populates jcbCustomerID
            String sql_cust_id = "SELECT customer_id FROM customer ORDER BY
customer_id";
            populate_combobox(sql_cust_id, frm.getJCBCustomerID(), frm);

            //Populates jcbCustomerName
            String sql_cust_name = "SELECT DISTINCT CONCAT(first_name,' ',
last_name) FROM customer ORDER BY CONCAT(first_name,' ', last_name)";
            populate_combobox(sql_cust_name, frm.getJCBCustomerName(), frm);

            //Populates jcbStoreID
            String sql_store_id = "SELECT store_id FROM store ORDER BY
store_id";
            populate_combobox(sql_store_id, frm.getJCBStoreID(), frm);

            //Populates jcbActive
            String sql_act = "SELECT DISTINCT active FROM customer ORDER BY
active";
            populate_combobox(sql_act, frm.getJCBActive(), frm);

            //Populates jcbAddressID
            String sql_add_id = "SELECT address_id FROM address ORDER BY
address_id";
            populate_combobox(sql_add_id, frm.getJCBAddressID(), frm);
```

```
        }catch (SQLException ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
            "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    private static void clear_store_controls(CustomerForm frm){
        frm.getJTFStoreID().setText("");
        frm.getJTFStoreAddress().setText("");
        frm.getJTFStoreDistrict().setText("");
        frm.getJTFStorePhone().setText("");
        frm.getJTFStoreCity().setText("");
        frm.getJTFStoreCountry().setText("");
    }

   //Displays store data result row by row
    private static <T> void display_store_data(CustomerForm frm, String sql,
item){
        try(Connection conn = getConnection()){
            PreparedStatement ps = conn.prepareStatement(sql);
            ps.setObject(1,item);
            ResultSet rs = ps.executeQuery();

            if (!rs.next()) {
                // no row found, clear the form fields
                clear_store_controls(frm);
                return;
            }

            do{

frm.getJTFStoreID().setText(String.valueOf(rs.getInt("store_id")));
                frm.getJTFStoreAddress().setText(rs.getString("address"));
                frm.getJTFStoreDistrict().setText(rs.getString("district"));
                frm.getJTFStorePhone().setText(rs.getString("phone"));
                frm.getJTFStoreCity().setText(rs.getString("city"));
                frm.getJTFStoreCountry().setText(rs.getString("country"));

                // Determines item selected from jcbStoreID
                find_combo_value_selected(frm.getJCBStoreID(),
rs.getInt("store_id"));

            }while(rs.next());

            rs.close();
            ps.close();
        }catch(SQLException ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }
```

```
    public static void jcbStore_handler(CustomerForm frm) {
        Object item = frm.getJCBStoreID().getSelectedItem();
        display_store_data(frm, Query_Store.get_sql_store_joint() + " WHERE
store_id = ?", item);
    }

    private static void clear_address_controls(CustomerForm frm){
        frm.getJTFAddressID().setText("");
        frm.getJTFAddress().setText("");
        frm.getJTFDistrict().setText("");
        frm.getJTFPhone().setText("");
        frm.getJTFPostalCode().setText("");
        frm.getJTFCity().setText("");
        frm.getJTFCountry().setText("");
    }

   //Displays address data result row by row
    private static <T> void display_address_data(CustomerForm frm, String sql
T item){
        try(Connection conn = getConnection()){
            PreparedStatement ps = conn.prepareStatement(sql);
            ps.setObject(1,item);
            ResultSet rs = ps.executeQuery();

            if (!rs.next()) {
                // no row found, clear the form fields
                clear_address_controls(frm);
                return;
            }

            do{

frm.getJTFAddressID().setText(String.valueOf(rs.getInt("address_id")));
                frm.getJTFAddress().setText(rs.getString("address"));
                frm.getJTFDistrict().setText(rs.getString("district"));
                frm.getJTFPhone().setText(rs.getString("phone"));
                frm.getJTFPostalCode().setText(rs.getString("postal_code"));
                frm.getJTFCity().setText(rs.getString("city"));
                frm.getJTFCountry().setText(rs.getString("country"));

                // Determines item selected from jcbAddressID
                find_combo_value_selected(frm.getJCBAddressID(),
rs.getInt("address_id"));

            }while(rs.next());

            rs.close();
            ps.close();
        }catch(SQLException ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }
```

```
    public static void jcbAddress_handler(CustomerForm frm) {
        Object item = frm.getJCBAddressID().getSelectedItem();
        display_address_data(frm, Query_Address.get_sql_address_joint() + "
WHERE address_id = ?", item);
    }

    private static void clear_customer_controls(CustomerForm frm){
        frm.getJTFCustomerID().setText("");
        frm.getJTFFirstName().setText("");
        frm.getJTFLastName().setText("");
        frm.getJTFEmail().setText("");
        frm.getJTFCreateDate().setText("");
        frm.getJTFLastUpdate().setText("");
    }

   //Displays customer data result row by row
    private static <T> void display_customer_data(CustomerForm frm, String sc
T item){
        try(Connection conn = getConnection()){
            PreparedStatement ps = conn.prepareStatement(sql);
            ps.setObject(1,item);
            ResultSet rs = ps.executeQuery();

            if (!rs.next()) {
                // no row found, clear the form fields
                clear_address_controls(frm);
                return;
            }

            do{

frm.getJTFCustomerID().setText(String.valueOf(rs.getInt("customer_id")));
                frm.getJTFEmail().setText(rs.getString("email"));
                frm.getJTFFirstName().setText(rs.getString("first_name"));
                frm.getJTFLastName().setText(rs.getString("last_name"));

frm.getJTFCreateDate().setText(String.valueOf(rs.getDate("create_date")));

frm.getJTFLastUpdate().setText(String.valueOf(rs.getTimestamp("last_update"))

                // Determines item selected from jcbCustomerID
                find_combo_value_selected(frm.getJCBCustomerID(),
rs.getInt("customer_id"));

                // Determines item selected from jcbCustomerName
                String full_name = rs.getString("first_name") + " " +
rs.getString("last_name");
                find_combo_value_selected(frm.getJCBCustomerName(), full_name

                // Determines item selected from jcbAddressID
                find_combo_value_selected(frm.getJCBAddressID(),
rs.getInt("address_id"));
```

```
                // Determines item selected from jcbStoreID
                find_combo_value_selected(frm.getJCBStoreID(),
rs.getInt("store_id"));

                // Determines item selected from jcbActive
                find_combo_value_selected(frm.getJCBActive(),
rs.getBoolean("active"));
            }while(rs.next());

            rs.close();
            ps.close();
        }catch(SQLException ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    public static void jcbCustomer_handler(CustomerForm frm, JComboBox<String
jcb1) {
        Object item1 = jcb1.getSelectedItem();
        String sql = "";
        if (jcb1.equals(frm.getJCBCustomerID())) {
            sql = Query_Customer.get_sql_id();
        } else if (jcb1.equals(frm.getJCBCustomerName())) {
            sql = Query_Customer.get_sql_name();
        }

        display_customer_data(frm, sql, item1);
    }

    public static void show_first_row(CustomerForm frm){
        String item =
String.valueOf(frm.getJCBCustomerID().getItemAt(FIRST_INDEX));
        display_customer_data(frm, SQL_ID, item);
        currentIndex = FIRST_INDEX;
    }

    public static void show_last_row(CustomerForm frm){
        int endIndex = frm.getJCBCustomerID().getItemCount() - 1;
        String item =
String.valueOf(frm.getJCBCustomerID().getItemAt(endIndex));
        display_customer_data(frm, SQL_ID, item);
        currentIndex = endIndex;
    }

    public static void show_prev_row(CustomerForm frm){
        currentIndex--;
        if(currentIndex < FIRST_INDEX){
            currentIndex = FIRST_INDEX;
            return;
        }
```

```
        String item =
String.valueOf(frm.getJCBCustomerID().getItemAt(currentIndex));
        display_customer_data(frm, SQL_ID, item);
    }

    public static void show_next_row(CustomerForm frm){
        int endIndex = frm.getJCBCustomerID().getItemCount() - 1;
        currentIndex++;
        if(currentIndex > endIndex){
            currentIndex = endIndex;
            return;
        }
        String item =
String.valueOf(frm.getJCBCustomerID().getItemAt(currentIndex));
        display_customer_data(frm, SQL_ID, item);
    }

    public static void mouse_pressed_handler(CustomerForm frm) {
        Objects.requireNonNull(frm, "frm must not be null");

        int selectedIndex = frm.getJTCustomer().getSelectedRow();
        if (selectedIndex == -1) {
            JOptionPane.showMessageDialog(frm, "Please select a row to view 
data.",
                "No row selected", JOptionPane.INFORMATION_MESSAGE);
            return;
        }

        try (Connection conn = getConnection()) {
            String id =
String.valueOf(frm.getJTCustomer().getModel().getValueAt(selectedIndex, 0));

            // Displays customer data
            display_customer_data(frm, SQL_ID, id);

        } catch (SQLException ex) {
            Logger.getLogger(CustomerForm.class.getName()).log(Level.SEVERE,
"Error displaying customer data", ex);
            String message = "Error displaying store data: " + ex.getMessage(
            String stackTrace = Arrays.toString(ex.getStackTrace());
            JOptionPane.showMessageDialog(frm, message + "\n\n" + stackTrace,
"ERROR", JOptionPane.ERROR_MESSAGE);
        }
    }

    //Updates row of data in customer tabel by customer_id
    public static void update_row_by_customer_id(CustomerForm frm, int cust_i
int store_id, String fname, String lname,
        String email, int add_id, boolean active, Date cr_date) throws
SQLException{
        Connection conn = getConnection();
        ResultSet rs = null;
```

```
        String query_id = "SELECT customer_id FROM customer WHERE customer_id
?";
        String update_query = """
            UPDATE customer SET store_id = ?, first_name = ?, last_name = ?,
            email = ?, address_id = ?, active = ?, create_date = ? WHERE
customer_id = ?""";
        try(PreparedStatement idPs = conn.prepareStatement(query_id,

ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_READ_ONLY);
        PreparedStatement updatePS = conn.prepareStatement(update_query,
                ResultSet.TYPE_SCROLL_SENSITIVE, ResultSet.CONCUR_UPDATABLE))
        {
            idPs.setInt(1,cust_id);
            if(!idPs.execute()){
                String message = "Can't find customer_id " + cust_id;

                JOptionPane.showMessageDialog(frm, message,
                    "ERROR",JOptionPane.ERROR_MESSAGE);
            } else{
                rs = idPs.getResultSet();
                rs.next();

                //Creates a Customer object using nine-params constructor
                Customer obj = new Customer(cust_id, store_id, fname, lname,
                    email, add_id, active, cr_date, new
Timestamp(System.currentTimeMillis()));
                updatePS.setInt(1, obj.getStoreID());
                updatePS.setString(2, obj.getFirstName());
                updatePS.setString(3, obj.getLastName());
                updatePS.setString(4, obj.getEmail());
                updatePS.setInt(5, obj.getAddressID());
                updatePS.setBoolean(6, obj.getActive());
                updatePS.setDate(7, obj.getCreateDate());
                updatePS.setInt(8, obj.getCustomerID());

                updatePS.executeUpdate();
                rs.close();
                updatePS.close();
                idPs.close();
                conn.close();
            }
        }catch(SQLException ex){
            Logger.getLogger(CustomerForm.class.getName()).log(Level.SEVERE,
"Error updating customer data", ex);
            String message = "Error updating customer data: " +
ex.getMessage();
            String stackTrace = Arrays.toString(ex.getStackTrace());
            JOptionPane.showMessageDialog(null, message + "\n\n" + stackTrace
"ERROR", JOptionPane.ERROR_MESSAGE);
        }catch(java.lang.NumberFormatException ex){
            Logger.getLogger(CustomerForm.class.getName()).log(Level.SEVERE,
"Invalid Input", ex);
            String message = "Invalid Input: " + ex.getMessage();
            String stackTrace = Arrays.toString(ex.getStackTrace());
```

```
            JOptionPane.showMessageDialog(null, message + "\n\n" + stackTrace
"ERROR", JOptionPane.ERROR_MESSAGE);
        }
    }

    private static HashMap<String, String> read_inputs(CustomerForm frm) {
        HashMap<String, String> input_data = new HashMap<>();
        String cust_id =
String.valueOf(frm.getJCBCustomerID().getSelectedItem());
        String store_id =
String.valueOf(frm.getJCBStoreID().getSelectedItem());
        String add_id =
String.valueOf(frm.getJCBAddressID().getSelectedItem());
        String fname = frm.getJTFFirstName().getText();
        String lname = frm.getJTFLastName().getText();
        String email = frm.getJTFEmail().getText();
        String active = String.valueOf(frm.getJCBActive().getSelectedItem());
        String cr_date = frm.getJTFCreateDate().getText();

        // Validate user input
        int add_id_int = 0;
        try {
            add_id_int = Integer.parseInt(add_id);
            if (add_id_int <= 0) {
                throw new IllegalArgumentException("Address ID cannot be
negative or zero");
            }
        } catch (NumberFormatException ex) {
            JOptionPane.showMessageDialog(frm, "Invalid Address ID: " + add_i
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        } catch (IllegalArgumentException ex) {
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        }

        int store_id_int = 0;
        try {
            store_id_int = Integer.parseInt(store_id);
            if (store_id_int <= 0) {
                throw new IllegalArgumentException("Store ID cannot be negati
or zero");
            }
        } catch (NumberFormatException ex) {
            JOptionPane.showMessageDialog(frm, "Invalid Store ID: " + store_i
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        } catch (IllegalArgumentException ex) {
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        }
```

```
        int cust_id_int = 0;
        try {
            cust_id_int = Integer.parseInt(cust_id);
            if (cust_id_int <= 0) {
                throw new IllegalArgumentException("Customer ID cannot be
negative or zero");
            }
        } catch (NumberFormatException ex) {
            JOptionPane.showMessageDialog(frm, "Invalid Manager Customer ID:
+ cust_id,
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        } catch (IllegalArgumentException ex) {
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        }

        if (fname == null || fname.isEmpty()) {
            JOptionPane.showMessageDialog(frm, "First name cannot be empty",
            "Error", JOptionPane.ERROR_MESSAGE);
            throw new IllegalArgumentException("First name cannot be empty");
        }

        if (lname == null || lname.isEmpty()) {
            JOptionPane.showMessageDialog(frm, "Last name cannot be empty",
            "Error", JOptionPane.ERROR_MESSAGE);
            throw new IllegalArgumentException("Last name cannot be empty");
        }

        input_data.put("cust_id", cust_id);
        input_data.put("store_id", store_id);
        input_data.put("fname", fname);
        input_data.put("lname", lname);
        input_data.put("email", email);
        input_data.put("add_id", add_id);
        input_data.put("active", active);
        input_data.put("cr_date", cr_date);

        return input_data;
    }

    private static void edit_actual(CustomerForm frm) throws ParseException{
        try{
            HashMap<String, String> input_data = read_inputs(frm);
            int add_id = Integer.parseInt(input_data.get("add_id"));
            int store_id = Integer.parseInt(input_data.get("store_id"));
            int cust_id = Integer.parseInt(input_data.get("cust_id"));
            String fname = input_data.get("fname");
            String lname = input_data.get("lname");
            String email = input_data.get("email");
            String active = input_data.get("active");
```

```
            String cr_date = input_data.get("cr_date");

            boolean bool_active = false;
            if(active.equalsIgnoreCase("true")){
                bool_active = true;
            }

            SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");
            java.util.Date date = sdf.parse(cr_date);
            java.sql.Date sqlDate_create_date = new
java.sql.Date(date.getTime());

            update_row_by_customer_id(frm, cust_id, store_id, fname, lname,
email, add_id, bool_active, sqlDate_create_date);

            //Refreshes all objects on form
            refresh_controls(frm);

        }catch(SQLException ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
        }
```

```
    }

    private static void enable_controls(boolean state, CustomerForm frm){
        frm.getJBFirst().setEnabled(state);
        frm.getJBPrev().setEnabled(state);
        frm.getJBNext().setEnabled(state);
        frm.getJBLast().setEnabled(state);
        frm.getJBInsert().setEnabled(state);
        frm.getJBDelete().setEnabled(state);
        frm.getJTFAddressID().setEnabled(state);
        frm.getJTFStoreID().setEnabled(state);
        frm.getJTFAddress().setEnabled(state);
        frm.getJTFDistrict().setEnabled(state);
        frm.getJTFPhone().setEnabled(state);
        frm.getJTFPostalCode().setEnabled(state);
        frm.getJTFCity().setEnabled(state);
        frm.getJTFCountry().setEnabled(state);
        frm.getJTFStoreAddress().setEnabled(state);
        frm.getJTFStoreDistrict().setEnabled(state);
        frm.getJTFStorePhone().setEnabled(state);
        frm.getJTFStoreCity().setEnabled(state);
        frm.getJTFStoreCountry().setEnabled(state);
        frm.getJTFLastUpdate().setEnabled(state);
    }

    public static void edit_handler(CustomerForm frm){
        if(frm.getJBEdit().getText().equals("EDIT")){
            frm.getJBEdit().setText("CONFIRM");

            // Disables controls
            enable_controls(false, frm);
        }

        else {
            try {
                frm.getJBEdit().setText("EDIT");

                // Actual editing
                edit_actual(frm);

                //Enables controls
                enable_controls(true, frm);
            } catch (ParseException ex) {

Logger.getLogger(Customer_Utils.class.getName()).log(Level.SEVERE, null,
ex);
            }
        }
    }

    public static void edit_database_from_jtable(TableModelEvent e,
CustomerForm frm) throws ParseException{
        if (e.getType() == TableModelEvent.UPDATE) {
```

```
            int row = e.getFirstRow();
            TableModel model = (TableModel)e.getSource();
            int cust_id =
Integer.parseInt(String.valueOf(model.getValueAt(row, 0)));
            int store_id =
Integer.parseInt(String.valueOf(model.getValueAt(row, 1)));
            int add_id =
Integer.parseInt(String.valueOf(model.getValueAt(row, 5)));
            String fname = String.valueOf(model.getValueAt(row, 2));
            String lname = String.valueOf(model.getValueAt(row, 3));
            String email = String.valueOf(model.getValueAt(row, 4));
            String active = String.valueOf(model.getValueAt(row, 6));
            String cr_date = String.valueOf(model.getValueAt(row, 7));

            boolean bool_active = false;
            if(active.equalsIgnoreCase("true")){
                bool_active = true;
            }

            SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");
            java.util.Date date = sdf.parse(cr_date);
            java.sql.Date sqlDate_create_date = new
java.sql.Date(date.getTime());

            try{
                update_row_by_customer_id(frm, cust_id, store_id, fname,
lname, email, add_id, bool_active, sqlDate_create_date);

                //Refreshes all objects on form
                refresh_controls(frm);

            } catch (SQLIntegrityConstraintViolationException ex) {

Logger.getLogger(CustomerForm.class.getName()).log(Level.SEVERE, "Duplicate
entry", ex);
                JOptionPane.showMessageDialog(frm, "Error: Duplicate
entry\n" + ex.getMessage());
            } catch (SQLSyntaxErrorException ex) {

Logger.getLogger(CustomerForm.class.getName()).log(Level.SEVERE, "Invalid
SQL syntax", ex);
                JOptionPane.showMessageDialog(frm, "Error: Invalid SQL
syntax\n" + ex.getMessage());
            } catch (SQLException ex) {

Logger.getLogger(CustomerForm.class.getName()).log(Level.SEVERE, "Database
error", ex);
                JOptionPane.showMessageDialog(frm, "Error: Database
error\n" + ex.getMessage());
            }
        }
    }
```

```
    //Inserts new row into customer table
    private static void insert_row(CustomerForm frm) throws SQLException,
ParseException{
        HashMap<String, String> input_data = read_inputs(frm);
        int add_id = Integer.parseInt(input_data.get("add_id"));
        int store_id = Integer.parseInt(input_data.get("store_id"));
        String fname = input_data.get("fname");
        String lname = input_data.get("lname");
        String email = input_data.get("email");
        String active = input_data.get("active");
        String cr_date = input_data.get("cr_date");

        boolean bool_active = false;
        if(active.equalsIgnoreCase("true")){
            bool_active = true;
        }

        // SQL insert statement
        String sql = """
            INSERT INTO customer(store_id, first_name, last_name,
                email, address_id, active, create_date) VALUES(?, ?, ?, ?,
?, ?, ?)""";

        try(Connection conn = getConnection();
         PreparedStatement pstmt = conn.prepareStatement(sql)){

            //Creates a Customer object eight-params constructor
            Customer obj = new Customer(store_id, fname, lname, email,
add_id, bool_active,
                    new Date(Calendar.getInstance().getTime().getTime()),
new Timestamp(System.currentTimeMillis()));
            pstmt.setInt(1,obj.getStoreID());
            pstmt.setString(2,obj.getFirstName());
            pstmt.setString(3,obj.getLastName());
            pstmt.setString(4,obj.getEmail());
            pstmt.setInt(5, obj.getAddressID());
            pstmt.setBoolean(6, obj.getActive());
            pstmt.setDate(7, obj.getCreateDate());

            //Executes the sql insert statement
            pstmt.executeUpdate();
        } catch (SQLException ex) {

Logger.getLogger(CustomerForm.class.getName()).log(Level.SEVERE, "Database
error", ex);
            JOptionPane.showMessageDialog(frm, "Error: Database error\n" +
ex.getMessage());
        }
    }

    private static void insert_actual(CustomerForm frm) throws
ParseException{
        try{
            insert_row(frm);
```

```
            //Refreshes table and comboboxes
            refresh_controls(frm);

        }catch(SQLException ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    public static void insert_handler(CustomerForm frm){
        if(frm.getJBInsert().getText().equals("INSERT")){
            frm.getJBInsert().setText("CONFIRM");

            //Disables jbEdit
            frm.getJBEdit().setEnabled(false);

            // Disables controls
            enable_controls(false, frm);
            frm.getJCBCustomerID().setEnabled(false);
            frm.getJCBCustomerName().setEnabled(false);
            frm.getJTFCreateDate().setEnabled(false);

            // Clears controls
            clear_customer_controls(frm);

            // Enables
            frm.getJBInsert().setEnabled(true);
        }

        else {
            frm.getJBInsert().setText("INSERT");

            try {
                // Actual insertion
                insert_actual(frm);
            } catch (ParseException ex) {

Logger.getLogger(Customer_Utils.class.getName()).log(Level.SEVERE, null,
ex);
            }

            //Enables jbEdit
            frm.getJBEdit().setEnabled(true);

            //Enables controls
            enable_controls(true, frm);
            frm.getJCBCustomerID().setEnabled(true);
            frm.getJCBCustomerName().setEnabled(true);
            frm.getJTFCreateDate().setEnabled(true);
        }
    }
```

```
    public static void delete_handler(CustomerForm frm){
        int dialogButton = JOptionPane.YES_NO_OPTION;
        int cust_id =
Integer.parseInt(String.valueOf(frm.getJCBCustomerID().getSelectedItem()));

        String message = String.format("Are you sure you want to delete the
row Customer ID: %d)", cust_id);
        int answer = JOptionPane.showConfirmDialog(frm, message, "DELETING
ROW OF DATA", dialogButton);

        if(answer == JOptionPane.YES_OPTION){
            String query = """
                DELETE FROM customer WHERE customer_id = ?""";
            try(Connection conn = getConnection();
                PreparedStatement ps = conn.prepareStatement(query)){
                // Use PreparedStatement to avoid SQL injection attacks
                ps.setInt(1, cust_id);
                ps.executeUpdate();

                // Refresh table and comboboxes
                refresh_controls(frm);

            } catch (SQLException ex){
                JOptionPane.showMessageDialog(frm, ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
            }
        }
    }

    private static void draw_pie_chart_customer_by_country(Charts_Customer
frm, JPanel jp){
        jp.setPreferredSize(new Dimension(jp.getWidth(), jp.getHeight()));
        DefaultPieDataset dataset =
create_pie_dataset(Query_Customer.get_sql_customer_country_dist(),
"number_of_customers", "customer_country");

        //Draws piechart customer distribution by country
        draw_piechart_with_dataset(frm, jp, dataset, "TOP 10 CUSTOMER
DISTRIBUTION BY COUNTRY");
    }

    private static void draw_pie_chart_customer_by_city(Charts_Customer
frm, JPanel jp){
        jp.setPreferredSize(new Dimension(jp.getWidth(), jp.getHeight()));
        DefaultPieDataset dataset =
create_pie_dataset(Query_Customer.get_sql_customer_city_dist(),
"number_of_customers", "customer_city");

        //Draws piechart customer distribution by city
        draw_piechart_with_dataset(frm, jp, dataset, "TOP 10 CUSTOMER
DISTRIBUTION BY CITY");
    }
```

```
    private static void draw_pie_chart_customer_by_district(Charts_Customer
frm, JPanel jp){
        jp.setPreferredSize(new Dimension(jp.getWidth(), jp.getHeight()));
        DefaultPieDataset dataset =
create_pie_dataset(Query_Customer.get_sql_customer_district_dist(),
"number_of_customers", "customer_district");

        //Draws piechart customer distribution by district
        draw_piechart_with_dataset(frm, jp, dataset, "TOP 10 CUSTOMER
DISTRIBUTION BY DISTRICT");
    }

    private static void
draw_pie_chart_customer_store_by_country(Charts_Customer frm, JPanel jp){
        jp.setPreferredSize(new Dimension(jp.getWidth(), jp.getHeight()));
        DefaultPieDataset dataset =
create_pie_dataset(Query_Customer.get_sql_customer_store_country_dist(),
"number_of_customers", "store_country");

        //Draws piechart customer distribution by store country
        draw_piechart_with_dataset(frm, jp, dataset, "TOP 10 CUSTOMER
DISTRIBUTION BY STORE COUNTRY");
    }

    private static void
draw_pie_chart_customer_store_by_city(Charts_Customer frm, JPanel jp){
        jp.setPreferredSize(new Dimension(jp.getWidth(), jp.getHeight()));
        DefaultPieDataset dataset =
create_pie_dataset(Query_Customer.get_sql_customer_store_city_dist(),
"number_of_customers", "store_city");

        //Draws piechart customer distribution by store city
        draw_piechart_with_dataset(frm, jp, dataset, "TOP 10 CUSTOMER
DISTRIBUTION BY STORE CITY");
    }

    private static void
draw_pie_chart_customer_store_by_district(Charts_Customer frm, JPanel jp){
        jp.setPreferredSize(new Dimension(jp.getWidth(), jp.getHeight()));
        DefaultPieDataset dataset =
create_pie_dataset(Query_Customer.get_sql_customer_store_district_dist(),
"number_of_customers", "store_district");

        //Draws piechart customer distribution by store district
        draw_piechart_with_dataset(frm, jp, dataset, "TOP 10 CUSTOMER
DISTRIBUTION BY STORE DISTRICT");
    }

    public static void jbchart_handler(Charts_Customer frm){
        //Draws piechart customer distribution by country
        draw_pie_chart_customer_by_country(frm, frm.getJPanel1());

        //Draws piechart customer distribution by city
```

```
        draw_pie_chart_customer_by_city(frm, frm.getJPanel2());

        //Draws piechart customer distribution by district
        draw_pie_chart_customer_by_district(frm, frm.getJPanel3());

        //Draws piechart customer distribution by store country
        draw_pie_chart_customer_store_by_country(frm, frm.getJPanel4());

        //Draws piechart customer distribution by store city
        draw_pie_chart_customer_store_by_city(frm, frm.getJPanel5());

        //Draws piechart customer distribution by store district
        draw_pie_chart_customer_store_by_district(frm, frm.getJPanel6());
    }
}
```

This is the full version of **CustomerForm.java**:

```
package sakila;
import java.awt.Toolkit;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JComboBox;
import javax.swing.JMenuItem;
import javax.swing.JOptionPane;
import javax.swing.JPopupMenu;
import javax.swing.JTable;
import javax.swing.JTextField;

public class CustomerForm extends javax.swing.JFrame {
    public CustomerForm() {
        initComponents();
        Utility.setLookAndFeel(this);
        Customer_Utils.refresh_controls(this);

 this.setIconImage(Toolkit.getDefaultToolkit().getImage(getClass().getResour
;
        this.setDefaultCloseOperation(this.HIDE_ON_CLOSE);
    }

    //Getter method for jtCustomer
    public JTable getJTCustomer(){
        return this.jtCustomer;
    }

    //Getter method for jtfCustomerID
    public JTextField getJTFCustomerID(){
        return this.jtfCustomerID;
    }
```

```
    //Getter method for jcbCustomerID
    public JComboBox getJCBCustomerID(){
        return this.jcbCustomerID;
    }

    //Getter method for jcbCustomerName
    public JComboBox getJCBCustomerName(){
        return this.jcbCustomerName;
    }

    //Getter method for jtfFirstName
    public JTextField getJTFFirstName(){
        return this.jtfFirstName;
    }

    //Getter method for jtfLastName
    public JTextField getJTFLastName(){
        return this.jtfLastName;
    }

    //Getter method for jtfAddressID
    public JTextField getJTFAddressID(){
        return this.jtfAddressID;
    }

    //Getter method for jcbAddressID
    public JComboBox getJCBAddressID(){
        return this.jcbAddressID;
    }

    //Getter method for jtfAddress
    public JTextField getJTFAddress(){
        return this.jtfAddress;
    }

    //Getter method for jtfEmail
    public JTextField getJTFEmail(){
        return this.jtfEmail;
    }

    //Getter method for jtfDistrict
    public JTextField getJTFDistrict(){
        return this.jtfDistrict;
    }

    //Getter method for jtfCity
    public JTextField getJTFCity(){
        return this.jtfCity;
    }

    //Getter method for jtfCountry
    public JTextField getJTFCountry(){
        return this.jtfCountry;
```

```
    }

    //Getter method for jtfPostalCode
    public JTextField getJTFPostalCode(){
        return this.jtfPostalCode;
    }

    //Getter method for jtfPhone
    public JTextField getJTFPhone(){
        return this.jtfPhone;
    }

    //Getter method for jcbActive
    public JComboBox getJCBActive(){
        return this.jcbActive;
    }

    //Getter method for jcbStoreID
    public JComboBox getJCBStoreID(){
        return this.jcbStoreID;
    }

    //Getter method for jtfStoreID
    public JTextField getJTFStoreID(){
        return this.jtfStoreID;
    }

    //Getter method for jtfLastUpdate
    public JTextField getJTFLastUpdate(){
        return this.jtfLastUpdate;
    }

    //Getter method for jtfCreateDate
    public JTextField getJTFCreateDate(){
        return this.jtfCreateDate;
    }

    //Getter method for jtfStoreDistrict
    public JTextField getJTFStoreDistrict(){
        return this.jtfStoreDistrict;
    }

    //Getter method for jtfStoreCity
    public JTextField getJTFStoreCity(){
        return this.jtfStoreCity;
    }

    //Getter method for jtfStoreCountry
    public JTextField getJTFStoreCountry(){
        return this.jtfStoreCountry;
    }

    //Getter method for jtfStorePhone
```

```
    public JTextField getJTFStorePhone(){
        return this.jtfStorePhone;
    }

    //Getter method for jtfStoreAddress
    public JTextField getJTFStoreAddress(){
        return this.jtfStoreAddress;
    }

    //Getter method for jbEdit
    public JButton getJBEdit(){
        return this.jbEdit;
    }

    //Getter method for jbInsert
    public JButton getJBInsert(){
        return this.jbInsert;
    }

    //Getter method for jbDelete
    public JButton getJBDelete(){
        return this.jbDelete;
    }

    //Getter method for jbChart
    public JButton getJBChart(){
        return this.jbChart;
    }

    //Getter method for jbFirst
    public JButton getJBFirst(){
        return this.jbFirst;
    }

    //Getter method for jbPrev
    public JButton getJBPrev(){
        return this.jbPrev;
    }

    //Getter method for jbNext
    public JButton getJBNext(){
        return this.jbNext;
    }

    //Getter method for jbLast
    public JButton getJBLast(){
        return this.jbLast;
    }

    //Getter method for jbAddressForm
    public JButton getJBAddressForm(){
        return this.jbAddressForm;
    }
```

```
    //Getter method for jbCountryForm
    public JButton getJBCountryForm(){
        return this.jbCountryForm;
    }

    //Getter method for jbCityForm
    public JButton getJBCityForm(){
        return this.jbCityForm;
    }

    //Getter method for jbStoreForm
    public JButton getJBStoreForm(){
        return this.jbStoreForm;
    }

    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">
    private void initComponents() {
        //...
        pack();
    }// </editor-fold>

    private void jtCustomerMousePressed(java.awt.event.MouseEvent evt) {
        Customer_Utils.mouse_pressed_handler(this);
    }

    private void jbFirstActionPerformed(java.awt.event.ActionEvent evt) {
        Customer_Utils.show_first_row(this);
    }

    private void jbPrevActionPerformed(java.awt.event.ActionEvent evt) {
        Customer_Utils.show_prev_row(this);
    }

    private void jbNextActionPerformed(java.awt.event.ActionEvent evt) {
        Customer_Utils.show_next_row(this);
    }

    private void jbLastActionPerformed(java.awt.event.ActionEvent evt) {
        Customer_Utils.show_last_row(this);
    }

    private void jbChartActionPerformed(java.awt.event.ActionEvent evt) {
        Charts_Customer frm1 = new Charts_Customer();
        frm1.setLocationRelativeTo(null);
        frm1.setTitle("SIX DISTRIBUTIONS IN CUSTOMER TABLE");
        frm1.setVisible(true);
        Customer_Utils.jbchart_handler(frm1);
    }

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

```
        Customer_Utils.edit_handler(this);
    }

    private void jbInsertActionPerformed(java.awt.event.ActionEvent evt) {
        Customer_Utils.insert_handler(this);
    }

    private void jbDeleteActionPerformed(java.awt.event.ActionEvent evt) {
        Customer_Utils.delete_handler(this);
    }

    private void jbCountryFormActionPerformed(java.awt.event.ActionEvent evt)
        CountryForm ct_form = new CountryForm();
        ct_form.setVisible(true);
    }

    private void jbAddressFormActionPerformed(java.awt.event.ActionEvent evt)
        AddressForm add_form = new AddressForm();
        add_form.setVisible(true);
    }

    private void jbStoreFormActionPerformed(java.awt.event.ActionEvent evt) {
        StoreForm str_form = new StoreForm();
        str_form.setVisible(true);
    }

    private void jbCityFormActionPerformed(java.awt.event.ActionEvent evt) {
        CityForm cty_form = new CityForm();
        cty_form.setVisible(true);
    }

    private void jcbCustomerIDActionPerformed(java.awt.event.ActionEvent evt)
        Customer_Utils.jcbCustomer_handler(this, this.jcbCustomerID);
    }

    private void jcbAddressIDActionPerformed(java.awt.event.ActionEvent evt)
        Customer_Utils.jcbAddress_handler(this);
    }

    private void jcbStoreIDActionPerformed(java.awt.event.ActionEvent evt) {
        Customer_Utils.jcbStore_handler(this);
    }

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

    private void jcbCustomerNameActionPerformed(java.awt.event.ActionEvent ev
        Customer_Utils.jcbCustomer_handler(this, this.jcbCustomerName);
    }

    private void jtCustomerMouseClicked(java.awt.event.MouseEvent evt) {
```

```
        // instantiate CustomerTableModelListener and add it as a listener to
        CustomerTableModelListener tableModelListener = new
CustomerTableModelListener(this.getJTCustomer(), this);
        this.getJTCustomer().getModel().addTableModelListener(tableModelListe
    }

    public static void main(String args[]) {
        try {
            for (javax.swing.UIManager.LookAndFeelInfo info :
javax.swing.UIManager.getInstalledLookAndFeels()) {
                if ("Nimbus".equals(info.getName())) {
                    javax.swing.UIManager.setLookAndFeel(info.getClassName())
                    break;
                }
            }
        } catch (ClassNotFoundException ex) {

java.util.logging.Logger.getLogger(CustomerForm.class.getName()).log(java.uti
null, ex);
        } catch (InstantiationException ex) {

java.util.logging.Logger.getLogger(CustomerForm.class.getName()).log(java.uti
null, ex);
        } catch (IllegalAccessException ex) {

java.util.logging.Logger.getLogger(CustomerForm.class.getName()).log(java.uti
null, ex);
        } catch (javax.swing.UnsupportedLookAndFeelException ex) {

java.util.logging.Logger.getLogger(CustomerForm.class.getName()).log(java.uti
null, ex);
        }

        /* Create and display the form */
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new CustomerForm().setVisible(true);
            }
        });
    }

    // Variables declaration - do not modify
    private javax.swing.JLabel jLabel10;
    private javax.swing.JLabel jLabel11;
    private javax.swing.JLabel jLabel12;
    private javax.swing.JLabel jLabel13;
    private javax.swing.JLabel jLabel14;
    private javax.swing.JLabel jLabel15;
    private javax.swing.JLabel jLabel16;
    private javax.swing.JLabel jLabel17;
    private javax.swing.JLabel jLabel18;
    private javax.swing.JLabel jLabel19;
    private javax.swing.JLabel jLabel20;
```

```
    private javax.swing.JLabel jLabel21;
    private javax.swing.JLabel jLabel22;
    private javax.swing.JLabel jLabel23;
    private javax.swing.JLabel jLabel3;
    private javax.swing.JLabel jLabel4;
    private javax.swing.JLabel jLabel5;
    private javax.swing.JLabel jLabel6;
    private javax.swing.JLabel jLabel7;
    private javax.swing.JLabel jLabel8;
    private javax.swing.JLabel jLabel9;
    private javax.swing.JScrollPane jScrollPane1;
    private javax.swing.JButton jbAddressForm;
    private javax.swing.JButton jbChart;
    private javax.swing.JButton jbCityForm;
    private javax.swing.JButton jbCountryForm;
    private javax.swing.JButton jbDelete;
    private javax.swing.JButton jbEdit;
    private javax.swing.JButton jbFirst;
    private javax.swing.JButton jbInsert;
    private javax.swing.JButton jbLast;
    private javax.swing.JButton jbNext;
    private javax.swing.JButton jbPrev;
    private javax.swing.JButton jbStoreForm;
    private javax.swing.JComboBox<String> jcbActive;
    private javax.swing.JComboBox<String> jcbAddressID;
    private javax.swing.JComboBox<String> jcbCustomerID;
    private javax.swing.JComboBox<String> jcbCustomerName;
    private javax.swing.JComboBox<String> jcbStoreID;
    private javax.swing.JTable jtCustomer;
    private javax.swing.JTextField jtfAddress;
    private javax.swing.JTextField jtfAddressID;
    private javax.swing.JTextField jtfCity;
    private javax.swing.JTextField jtfCountry;
    private javax.swing.JTextField jtfCreateDate;
    private javax.swing.JTextField jtfCustomerID;
    private javax.swing.JTextField jtfDistrict;
    private javax.swing.JTextField jtfEmail;
    private javax.swing.JTextField jtfFirstName;
    private javax.swing.JTextField jtfLastName;
    private javax.swing.JTextField jtfLastUpdate;
    private javax.swing.JTextField jtfPhone;
    private javax.swing.JTextField jtfPostalCode;
    private javax.swing.JTextField jtfStoreAddress;
    private javax.swing.JTextField jtfStoreCity;
    private javax.swing.JTextField jtfStoreCountry;
    private javax.swing.JTextField jtfStoreDistrict;
    private javax.swing.JTextField jtfStoreID;
    private javax.swing.JTextField jtfStorePhone;
    // End of variables declaration
}
```

STAFF FORM

STAFF FORM

CREATING AND POPULATING STAFF TABLE

CREATING AND POPULATING STAFF TABLE

Step 1	Create a new class named **Query_Staff**. It contains several SQL queries related to the staff table in the sakila database. Here's a brief explanation of each query: 1. **sql_min**: Selects the minimum value of the **staff_id** column from the **staff** table. 2. **sql_max**: Selects the maximum value of the **staff_id** column from the **staff** table. 3. **sql_id**: Selects all columns from the staff table where the **staff_id** column matches a given input value. 4. **sql_name**: Selects all columns from the **staff** table where the concatenated **first_name** and **last_name** columns match a given input value. 5. **sql_staff_country_dist**: Selects the number of staffs in each country where a staff member works, sorted in descending order by the number of customers. Only the top 10 countries are returned. 6. **sql_staff_city_dist**: Selects the number of staffs in each city where a staff member works, sorted in descending order by the number of customers. Only the top 10 cities are returned. 7. **sql_staff_joint**: Selects all columns from the staff, store, address, city, and country tables, where the foreign key relationships are used to join the tables together.

8. **sql_staff**: Creates the **staff** table in the database with the specified column names, data types, and constraints.

The class also contains several getter methods that return the SQL queries as strings. These methods can be used by other classes to access and execute the queries.

```
1  package sakila;
2
3  public class Query_Staff {
4      private static final String
5  sql_min = "SELECT MIN(staff_id)
6  FROM staff";
7      private static final String
8  sql_max = "SELECT MAX(staff_id)
9  FROM staff";
10     private static final String
11 sql_id = "SELECT * FROM staff WHERE
12 staff_id  = ?";
13     private static final String
14 sql_name = "SELECT * FROM staff
15 WHERE CONCAT(first_name,'
16 ',last_name) = ?";
17
18     private static final String
19 sql_staff_country_dist = """
20         SELECT co.country AS
21 staff_country, COUNT(*) AS
22 number_of_staffs
23         FROM staff sf
24         JOIN address ad ON
25 ad.address_id = sf.address_id
26         JOIN city ci ON ci.city_id
27 = ad.city_id
28         JOIN country co ON
29 co.country_id = ci.country_id
30         GROUP BY staff_country
31         ORDER BY number_of_staffs
32 DESC
33         LIMIT 10;""";
34
35     private static final String
36 sql_staff_city_dist = """
37         SELECT ci.city AS
38 staff_city, COUNT(*) AS
39 number_of_staffs
40         FROM staff sf
41
```

```
42          JOIN address ad ON
43  ad.address_id = sf.address_id
44          JOIN city ci ON ci.city_id
45  = ad.city_id
46          GROUP BY staff_city
47          ORDER BY number_of_staffs
48  DESC
49          LIMIT 10;""";
50
51      private static final String
52  sql_staff_joint = """
53          SELECT sf.staff_id,
54  sf.store_id, sf.first_name,
55              sf.last_name, sf.email,
56  sf.address_id,
57              sf.active, sf.picture,
58  sf.username,
59              sf.password,
60  sf.last_update,
61              ad.address,
62  ad.district, ad.postal_code,
63              ad.phone, ci.city,
64  co.country
65          FROM staff sf
66          JOIN store st ON
67  st.store_id = sf.store_id
68          JOIN address ad ON
69  ad.address_id = sf.address_id
70          JOIN city ci ON ci.city_id
71  = ad.city_id
72          JOIN country co ON
73  co.country_id = ci.country_id""";
74
75      private static final String
76  sql_staff = """
77          CREATE TABLE staff (
78            staff_id TINYINT UNSIGNED
79  NOT NULL AUTO_INCREMENT,
80            first_name VARCHAR(45)
81  NOT NULL,
82            last_name VARCHAR(45) NOT
83  NULL,
84            address_id SMALLINT
85  UNSIGNED NOT NULL,
86            picture BLOB DEFAULT
87  NULL,
88            email VARCHAR(50) DEFAULT
89  NULL,
90            store_id TINYINT UNSIGNED
91  NOT NULL,
92            active BOOLEAN NOT NULL
93  DEFAULT TRUE,
94
95
```

```
96          username VARCHAR(16) NOT
97  NULL,
98          password VARCHAR(40)
99  CHARACTER SET utf8mb4 COLLATE
    utf8mb4_bin DEFAULT NULL,
          last_update TIMESTAMP NOT
    NULL DEFAULT CURRENT_TIMESTAMP ON
    UPDATE CURRENT_TIMESTAMP,
          PRIMARY KEY  (staff_id),
          KEY idx_fk_store_id
    (store_id),
          KEY idx_fk_address_id
    (address_id),
          CONSTRAINT fk_staff_store
    FOREIGN KEY (store_id) REFERENCES
    store (store_id) ON DELETE RESTRICT
    ON UPDATE CASCADE,
          CONSTRAINT
    fk_staff_address FOREIGN KEY
    (address_id) REFERENCES address
    (address_id) ON DELETE RESTRICT ON
    UPDATE CASCADE
        ) ENGINE=InnoDB DEFAULT
    CHARSET=utf8mb4;""";

    //Getter methods
    public static String
    get_sql_min() {
        return sql_min;
    }

    public static String
    get_sql_max() {
        return sql_max;
    }

    public static String
    get_sql_id() {
        return sql_id;
    }

    public static String
    get_sql_name() {
        return sql_name;
    }

    public static String
    get_sql_staff() {
        return sql_staff;
    }

    public static String
    get_sql_staff_joint() {
```

```
        return sql_staff_joint;
    }

    public static String
get_sql_staff_city_dist() {
        return sql_staff_city_dist;
    }

    public static String
get_sql_staff_country_dist() {
        return
sql_staff_country_dist;
    }
}
```

Step 2

Then, create a public class named **Staff**. It has 17 instance variables and 4 constructors, along with getter and setter methods for each instance variable.

The instance variables are:

- staff_id: an integer representing the staff's ID
- store_id: an integer representing the store's ID where the staff works
- fname: a string representing the staff's first name
- lname: a string representing the staff's last name
- email: a string representing the staff's email
- add_id: an integer representing the staff's address ID
- active: a boolean indicating whether the staff is currently active or not
- username: a string representing the staff's username
- password: a string representing the staff's password
- picture: a byte array representing the staff's profile picture
- last_update: a timestamp representing the last time the staff's information was updated

- cust_address: a string representing the customer's address
- cust_district: a string representing the customer's district
- cust_city: a string representing the customer's city
- cust_country: a string representing the customer's country
- cust_postal_code: a string representing the customer's postal code
- cust_phone: a string representing the customer's phone number

The class has a default constructor, a constructor with ten parameters, a constructor with eleven parameters, and a constructor with seventeen parameters. The default constructor initializes some of the instance variables to default values. The other constructors take different combinations of the instance variables as parameters and use the setter methods to initialize them.

Each instance variable has a getter and a setter method that can be used to retrieve or modify its value. The setter methods also perform some input validation to ensure that the values being set are valid.

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
```

18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71

```
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
```

```
99  package sakila;
100 import java.sql.Timestamp;
101
102 public class Staff {
103     //17 instance variables
104     private int staff_id;
105     private int store_id;
106     private String fname;
107     private String lname;
108     private String email;
109     private int add_id;
110     private boolean active;
111     private String username;
112     private String password;
113     private byte[] picture;
114     private Timestamp last_update;
115
116     private String cust_address;
117     private String cust_district;
118     private String cust_city;
119     private String cust_country;
120     private String cust_postal_code;
121     private String cust_phone;
122
123     //Default constructor
124     Staff(){
125         this(1, 1, "Fname", "Lname",
126127 "xxx@gmail.com", 1, true,
128                 "username", "password",
129 null, new
130 Timestamp(System.currentTimeMillis()));
131     }
132
133     //Constructor with ten-params
134     Staff(int store_id, String fname,
135 String lname, String email,
136             int add_id, boolean active,
137 String user, String pass, byte[] pict,
138 Timestamp lu){
139         setStoreID(store_id);
140         setFirstName(fname);
141         setLastName(lname);
142         setAddressID(add_id);
143         setEmail(email);
144         setActive(active);
145         setUserName(user);
146         setPassword(pass);
147         setPicture(pict);
148         setLastUpdate(lu);
149     }
150
151     //Constructor with eleven-params
152
```

```
153     Staff(int staff_id, int store_id,
154 String fname, String lname, String
155 email,
156             int add_id, boolean active,
157 String user, String pass, byte[] pict,
158 Timestamp lu){
159         this(store_id, fname, lname,
160 email, add_id, active, user, pass,
161 pict, lu);
162         setStaffID(staff_id);
163     }
164 
165     //Constructor with seventeen-params
166     Staff(int staff_id, int store_id,
167 String fname, String lname,
168         String email, int add_id,
169 boolean active, String user, String
170 pass, byte[] pict, Timestamp lu,
171         String cust_address, String
172 cust_district, String cust_city,
173         String cust_country, String
174 cust_postal_code, String cust_phone){
175         this(staff_id, store_id, fname,
176 lname, email, add_id, active, user,
177 pass, pict, lu);
178         this.cust_address =
179 cust_address;
180         this.cust_district =
181 cust_district;
182         this.cust_city = cust_city;
183         this.cust_country =
184 cust_country;
185         this.cust_postal_code =
186 cust_postal_code;
187         this.cust_phone = cust_phone;
188     }
189 
190     //Getter methods
191     public int getStaffID() {return
192 staff_id;}
193     public int getStoreID() {return
    store_id;}
    public String getFirstName()
    {return fname;}
    public String getLastName() {return
    lname;}
    public String getEmail() {return
    email;}
    public int getAddressID() {return
    add_id;}
    public boolean getActive(){return
    active;}
    public String getUserName(){return
    this.username;}
```

```
    public String getPassword(){return
this.password;}
    public byte[] getPicture(){return
picture;};
    public Timestamp getLastUpdate()
{return last_update;}

    public String getCustomerAddress()
{return cust_address;}
    public String getCustomerCity()
{return cust_city;}
    public String getCustomerDistrict()
{return cust_district;}
    public String getCustomerCountry()
{return cust_country;}
    public String
getCustomerPostalCode() {return
cust_postal_code;}
    public String getCustomerPhone()
{return cust_phone;}

    //Setter methods
    public void setStaffID(int id) {
        if (id <= 0) {
            throw new
IllegalArgumentException("Staff ID must be greater
than zero.");
        }
        this.staff_id = id;
    }

    public void setStoreID(int id) {
        if (id <= 0) {
            throw new
IllegalArgumentException("Store ID must be greater
than zero.");
        }
        this.store_id = id;
    }

    public void setFirstName(String
name) {
        if (name == null ||
name.trim().isEmpty()) {
            throw new
IllegalArgumentException("First name cannot be null
or empty");
        }
        if (name.length() > 45) {
            throw new
IllegalArgumentException("First name
cannot be longer than 45 characters");
```

```
        }
        this.fname = name;
    }

    public void setLastName(String
name) {
        if (name == null ||
name.trim().isEmpty()) {
            throw new
IllegalArgumentException("Last name cannot be null
or empty");
        }
        if (name.length() > 45) {
            throw new
IllegalArgumentException("Last name
cannot be longer than 45 characters");
        }
        this.lname = name;
    }

    public void setEmail(String email)
{
        if (email != null &&
email.length() > 50) {
            throw new
IllegalArgumentException("Email cannot
be longer than 50 characters");
        }
        this.email = (email != null) ?
email : ""; // set default value to
empty string if add is null
    }

    public void setAddressID(int id) {
        if (id <= 0) {
            throw new
IllegalArgumentException("Address ID must be greater
than zero.");
        }
        this.add_id = id;
    }

    public void setActive(boolean
active) {
        this.active = active;
    }

    public void setUserName(String
name) {
        if (name == null ||
name.trim().isEmpty()) {
            throw new
IllegalArgumentException("User name
```

```
cannot be null or empty");
        }
        if (name.length() > 16) {
            throw new
IllegalArgumentException("User name
cannot be longer than 16 characters");
        }
        this.username = name;
    }

    public void setPassword(String
pass) {
        if (pass != null &&
pass.length() > 50) {
            throw new
IllegalArgumentException("Password
cannot be longer than 40 characters");
        }
        this.password = (pass != null)
? pass : ""; // set default value to
empty string if add is null
    }

    public void setPicture(byte[]
pict) {
        this.picture = pict;
    }

    public void setLastUpdate(Timestamp
date){
        if (date == null) {
        throw new
IllegalArgumentException("Date cannot
be null");
        }
        this.last_update = date;
    }

    @Override
    public String toString(){
        return "\nStaff ID          : "
+ getStaffID() +
               "\nStore ID          : "
+ getStoreID() +
               "\nStaff Name        : "
+ getFirstName() + " " +  getLastName()
+
               "\nEmail             : "
+ getEmail() +
               "\nAddress ID        : "
+ getAddressID() +
               "\nActive            : "
+ getActive() +
```

```
                "\nUsername          : "
+ getUserName() +
                "\nPassword          : "
+ getPassword() +
                "\nLast Update       : "
+ getLastUpdate();
    }
}
```

Step 3

Create a new public clas
contains methods related to
staff table in the Sakila datal

The class extends a **Utility**
Java packages, includir
javax.swing, and org.jfree. It
final variables and a static va
an initial value of FIRST_IND

The class contains tl
create_staff_table(), **popu**
read_staff_table(). The **cre**
creates the staff table in tl
query. The **populate_staff**
some sample data into the s
objects, one created with a d
other created with an eleve
The **read_staff_table()** met
the staff table and creates a
printing it to the console.

The class also contains a priv
that holds a SQL query to re
of the **staff** table.

```
1  package sakila;
2  import java.awt.Dimensi
3  import java.util.loggir
4  import java.util.loggir
5  import java.sql.*;
6  import java.text.ParseE
7  import java.text.Simple
8  import java.util.Arrayl
9  import java.util.Arrays
10 import java.util.Calend
11 import java.util.HashMa
12 import java.util.Object
13 import javax.swing.JCon
```

```
14  import javax.swing.JOpt
15  import javax.swing.JPar
16  import javax.swing.even
17  import javax.swing.tabl
18  import javax.swing.tabl
19  import
20  org.jfree.data.category
21  import
22  org.jfree.data.general.
23
24  public class Staff_Util
25      public static final
26      public static final
27
28      private static int
29  FIRST_INDEX;
30      private static fina
31  Query_Staff.get_sql_id(
32
33      //Creates staff tab
34      public static void
35          try (Connectior
36  {
37              Statement s
38  conn.createStatement();
39
40  stmt.addBatch(Query_Sta
41              stmt.execut
42
43              String mess
44  String.format("Successf
45  table");
46              JOptionPane
47  message,
48
49   "INFORMATION",JOptionP
50  ;
51
52          } catch (SQLExc
53              JOptionPane
54  ex.getMessage(),
55
56   "ERROR",JOptionPane.EF
57          }
58      }
59
60      //Populates staff t
61  data
62      public static void
63          try(Connection
64              String sql
65              INSERT INTO
66  store_id, first_name, l
67
```

```
68              email, addr
69  username, password, pic
70                    VALUES(
71  ?, ?, ?)""";
72
73              //Creates a
74  default constructor
75              PreparedSta
76  conn.prepareStatement(s
77              Staff obj1
78              ps1.setInt(
79              ps1.setInt(
80
81  ps1.setString(3,obj1.ge
82
83  ps1.setString(4,obj1.ge
84              ps1.setStri
85              ps1.setInt(
86              ps1.setBool
87
88  ps1.setString(8,obj1.ge
89
90  ps1.setString(9,obj1.ge
91              ps1.setByte
92
93  ps1.setTimestamp(11,obj
94
95              // Creates
96  eleven-params construct
97              PreparedSta
98  conn.prepareStatement(s
99              Staff obj2
100 "Vivian", "Siahaan", "v
101                       2,
102 "mypassword", null, new
103 Timestamp(System.curren
104             ps2.setInt(
105             ps2.setInt(
106
107 ps2.setString(3,obj2.ge
108
109 ps2.setString(4,obj2.ge
110             ps2.setStri
111             ps2.setInt(
112             ps2.setBool
113
114 ps2.setString(8,obj2.ge
115
116 ps2.setString(9,obj2.ge
117             ps2.setByte
118
119 ps2.setTimestamp(11,obj
120
121
```

```
            ps1.execute
            ps2.execute

        }catch(SQLExcep
            JOptionPane
ex.getMessage(),

 "ERROR",JOptionPane.ER
        }
    }

    //Reads the content
    public static void
        try(Connection
            Statement s
conn.createStatement();
            ResultSet r
stmt.executeQuery("SELE

            while(rs.ne
                int sta
rs.getInt("staff_id");
                int sto
rs.getInt("store_id");
                int add
rs.getInt("address_id")
                String
rs.getString("first_nam
                String
rs.getString("last_name
                String
rs.getString("email");
                boolean
rs.getBoolean("active")
                String
rs.getString("username"
                String
rs.getString("password"
                byte[]
rs.getBytes("picture");
                Timesta
rs.getTimestamp("last_u

                //Creat
eleven-params construct
                Staff c
Staff(staff_id, store_i
add_id, active, user, p
                System.
            }
            rs.close();
            stmt.close(
```

```
        }catch(SQLExcep
            JOptionPane
ex.getMessage(),

 "ERROR",JOptionPane.ER
        }
    }
}
```

Step 4

In the driver class, **create_staff_table()**, **popu** **read_staff_table()** as shown

```
1  package sakila;
2
3  public class Sakila {
4      public static void m
5  {
6          //...
7
8  //
9  Inventory_Utils.create_i
10 //
11 Inventory_Utils.populate
12 //
13 Inventory_Utils.read_inv
14 //         InventoryForm
15 InventoryForm();
16 //         frm.setVisible
17
18 //
19 Customer_Utils.create_cu
20 //
21 Customer_Utils.populate_
22 //
23 Customer_Utils.read_cust
   //         CustomerForm f
   CustomerForm();
   //         frm.setVisible

           Staff_Utils.crea
           Staff_Utils.popu
           Staff_Utils.read
       }
   }
```

Run project to see the result

```
Staff ID          : 1
Store ID          : 1
```

	Staff Name : Fname Lr Email : xxx@gmai Address ID : 1 Active : true Username : username Password : passworc Last Update : 2023-05- Staff ID : 2 Store ID : 2 Staff Name : Vivian S Email : vivian@g Address ID : 2 Active : false Username : vivian Password : mypasswc Last Update : 2023-05-

DESIGNING GUI
DESIGNING GUI

Step 1	In the project, create a new **JFrame Form** and name it as **StaffForm.java**. In the Design tab, add twenty-two **JLabels** to the form and set their corresponding **text** properties as STAFF ID, FULL NAME, FIRST NAME, LAST NAME, ADDRESS ID, ADDRESS, EMAIL, DISTRICT, POSTAL CODE, CITY, COUNTRY, PHONE, ACTIVE, USERNAME, PASSWORD, STORE ID, STORE DISTRICT, STORE COUNTRY, STORE CITY, STORE ADDRESS, STORE PHONE, and LAST UPDATE. Then, add another **JLabel** and set its **Variable Name** as **jlPicture**. This label is used for displaying staff picture.
Step 2	Then, add twenty **JTextField** to the form and set their corresponding **Variable Name** as **jtfStaffID**, **jtfFirstName**, **jtfLastName**, **jtfAddressID**, **jtfAddress**, **jtfEmail**, **jtfPostalCode**, **jtfPhone**, **jtfDistrict**, **jtfCity**, **jtfCountry**, **jtfUsername**, **jtfPassword**, **jtfStoreID**, **jtfStoreDistrict**, **jtfStoreCountry**, **jtfStoreCity**, **jtfStoreAddress**, **jtfStorePhone**, and **jtfLastUpdate**.

Step 3	Then, add twelve **JButton** to the form and set their corresponding **Variable Name** as **jbFirst**, **jbPrev**, **jbNext**, **jbLast**, **jbEdit**, **jbInsert**, **jbDelete**, **jbAddress**, **jbCountryForm**, **jbCityForm**, **jbAddressForm**, **jbStoreForm**, and **jbChart**. Set their corresponding **text** properties as FIRST, PREV, NEXT, LAST, EDIT, INSERT, DELETE, ADDRESS FORM, COUNTRY FORM, CITY FORM, STORE FORM, and CHART.
Step 4	Then, add five **JComboBoxes** to the form and set their corresponding **Variable Name** as **jcbStaffID**, **jcbCustomerName**, **jcbStoreID**, **jcbAddressID**, and **jcbActive**.
Step 5	Lastly, add a new **JTable** to the form set set its **Variable Name** as **jtStaff**. Then, right-click on it, then choose **Table Contents...** and set the number of columns to 17 and the number of rows to 50.

Step 6	In the driver class, **Sakila.java**, create a new object of **StaffForm** class using its default constructor as shown in 16 – 17:
	
	1 package sakila;
	2
	3 public class Sakila {
	4 public static void main(String[]
	5 args) {
	6 //...
	7
	8 //
	9 Customer_Utils.create_customer_table();
	10 //
	11 Customer_Utils.populate_customer_table();
	12 //
	13 Customer_Utils.read_customer_table();
	14 // CustomerForm frm = new
	15 CustomerForm();
	16 // frm.setVisible(true);
	17
	18 //
	19 Staff_Utils.create_staff_table();

```
//
Staff_Utils.populate_staff_table();
//        Staff_Utils.read_staff_table();
        StaffForm frm = new StaffForm();
        frm.setVisible(true);
    }
}
```

Step 7

In **StaffForm**'s constructor, invoke **setLookAndFeel()** to set the look and feel of the form as shown in line 17.

```
1  package sakila;
2
3  import java.awt.Toolkit;
4  import java.awt.event.ActionEvent;
5  import
6  java.awt.event.ActionListener;
7  import javax.swing.JButton;
8  import javax.swing.JComboBox;
9  import javax.swing.JMenuItem;
10 import javax.swing.JOptionPane;
11 import javax.swing.JPopupMenu;
12 import javax.swing.JTable;
13 import javax.swing.JTextField;
14
15 public class StaffForm extends
16 javax.swing.JFrame {
17     public StaffForm() {
18         initComponents();
19
20 Utility.setLookAndFeel(this);
    }
    //...
}
```

Run the project to see the staff form as shown in Figure 14.1.

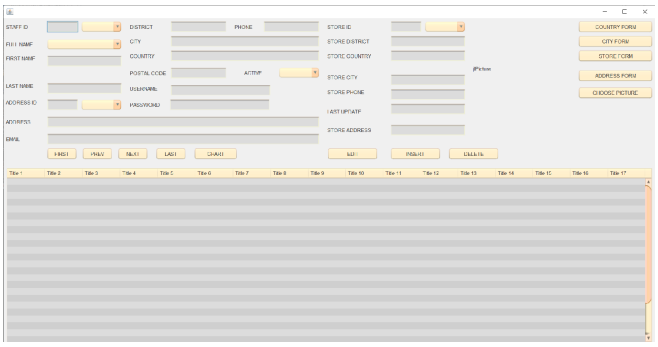


	Figure 14.1 The layout of staff form
Step 9	In **StaffForm.java**, define **getter()** method for every object in the form and place them outside and beneath its default constructor:

```
1       //Getter method for jtStaff
2       public JTable getJTStaff(){
3           return this.jtStaff;
4       }
5
6       //Getter method for
7   jlPicture
8       public JLabel getJLPicture()
9   {
10          return this.jlPicture;
11      }
12
13      //Getter method for
14  jtfStaffID
15      public JTextField
16  getJTFStaffID(){
17          return this.jtfStaffID;
18      }
19
20      //Getter method for
21  jcbStaffID
22      public JComboBox
23  getJCBStaffID(){
24          return this.jcbStaffID;
25      }
26
27      //Getter method for
28  jcbStaffName
29      public JComboBox
30  getJCBStaffName(){
31          return
32  this.jcbStaffName;
33      }
34
35      //Getter method for
36  jtfFirstName
37      public JTextField
38  getJTFFirstName(){
39          return
40  this.jtfFirstName;
41      }
42
43      //Getter method for
44  jtfLastName
45      public JTextField
46  getJTFLastName(){
```

```
47          return this.jtfLastName;
48      }
49
50      //Getter method for
51  jtfAddressID
52      public JTextField
53  getJTFAddressID(){
54          return
55  this.jtfAddressID;
56      }
57
58      //Getter method for
59  jcbAddressID
60      public JComboBox
61  getJCBAddressID(){
62          return
63  this.jcbAddressID;
64      }
65
66      //Getter method for
67  jtfAddress
68      public JTextField
69  getJTFAddress(){
70          return this.jtfAddress;
71      }
72
73      //Getter method for jtfEmail
74      public JTextField
75  getJTFEmail(){
76          return this.jtfEmail;
77      }
78
79      //Getter method for
80  jtfDistrict
81      public JTextField
82  getJTFDistrict(){
83          return this.jtfDistrict;
84      }
85
86      //Getter method for jtfCity
87      public JTextField
88  getJTFCity(){
89          return this.jtfCity;
90      }
91
92      //Getter method for
93  jtfCountry
94      public JTextField
95  getJTFCountry(){
96          return this.jtfCountry;
97      }
98
99
100
```

```
101     //Getter method for
102 jtfPostalCode
103     public JTextField
104 getJTFPostalCode(){
105         return
106 this.jtfPostalCode;
107     }
108 
109     //Getter method for jtfPhone
110     public JTextField
111 getJTFPhone(){
112         return this.jtfPhone;
113     }
114 
115     //Getter method for
116 jcbActive
117     public JComboBox
118 getJCBActive(){
119         return this.jcbActive;
120     }
121 
122     //Getter method for
123 jcbStoreID
124     public JComboBox
125 getJCBStoreID(){
126         return this.jcbStoreID;
127     }
128 
129     //Getter method for
130 jtfStoreID
131     public JTextField
132 getJTFStoreID(){
133         return this.jtfStoreID;
134     }
135 
136     //Getter method for
137 jtfLastUpdate
138     public JTextField
139 getJTFLastUpdate(){
140         return
141 this.jtfLastUpdate;
142     }
143 
144     //Getter method for
145 jtfUsername
146     public JTextField
147 getJTFUsername(){
148         return this.jtfUsername;
149     }
150 
151 
152 
153 
154 
```

155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197198
199

```
    //Getter method for jtfPassword
    public JTextField getJTFPassword(){
        return this.jtfPassword;
    }

    //Getter method for
jtfStoreDistrict
    public JTextField
getJTFStoreDistrict(){
        return this.jtfStoreDistrict;
    }

    //Getter method for jtfStoreCity
    public JTextField getJTFStoreCity()
{
        return this.jtfStoreCity;
    }

    //Getter method for jtfStoreCountry
    public JTextField
getJTFStoreCountry(){
        return this.jtfStoreCountry;
    }

    //Getter method for jtfStorePhone
    public JTextField
getJTFStorePhone(){
        return this.jtfStorePhone;
    }

    //Getter method for jtfStoreAddress
    public JTextField
getJTFStoreAddress(){
        return this.jtfStoreAddress;
    }

    //Getter method for jbEdit
    public JButton getJBEdit(){
        return this.jbEdit;
    }

    //Getter method for jbInsert
    public JButton getJBInsert(){
        return this.jbInsert;
    }

    //Getter method for jbDelete
    public JButton getJBDelete(){
        return this.jbDelete;
    }

    //Getter method for jbChart
```

```
    public JButton getJBChart(){
        return this.jbChart;
    }

    //Getter method for jbFirst
    public JButton getJBFirst(){
        return this.jbFirst;
    }

    //Getter method for jbPrev
    public JButton getJBPrev(){
        return this.jbPrev;
    }

    //Getter method for jbNext
    public JButton getJBNext(){
        return this.jbNext;
    }

    //Getter method for jbLast
    public JButton getJBLast(){
        return this.jbLast;
    }

    //Getter method for jbAddressForm
    public JButton getJBAddressForm(){
        return this.jbAddressForm;
    }

    //Getter method for jbCountryForm
    public JButton getJBCountryForm(){
        return this.jbCountryForm;
    }

    //Getter method for jbCityForm
    public JButton getJBCityForm(){
        return this.jbCityForm;
    }

    //Getter method for jbStoreForm
    public JButton getJBStoreForm(){
        return this.jbStoreForm;
    }

    //Getter method for jbChoosePicture
    public JButton getJBChoosePicture()
{
        return this.jbChoosePicture;
    }
```

POPULATING TABLE AND COMBOBOXES
POPULATING TABLE AND COMBOBOXES

Step 1

In **Staff_Utils.java**, add two new methods: **get_staff_list()** and **sho**
get_staff_list(), retrieves staff data from a database using a SQL query
objects. The **Staff** class likely has fields that correspond to the columns i
three parameters: a **StaffForm** object (which is likely the GUI window tha
as a **String**, and an item parameter that is used to filter the query base
uses a try-with-resources statement to create a database connection and a
parameters if necessary, executes the query, and then iterates over the r
add them to the **ArrayList**. If an exception is thrown, it displays an error
an empty **ArrayList**.

The second method, **show_table_staff()**, takes the **StaffForm** object an
parameters, and populates a **JTable** component in the **StaffForm** GUI wit
a **DefaultTableModel** object with 17 columns (one for each field in the **S**
using an array of **Strings**, and sets the table model for the **JTable** compo
the **ArrayList** and adds each **Staff** object's fields to an **Object** array ro
model using the **addRow()** method.

```
1     private static ArrayList<Staff> get_staff_list(StaffForm
2  frm, String sql, String item){
3          ArrayList<Staff> list = new ArrayList<>();
4
5          try(Connection conn = getConnection();
6              PreparedStatement ps = conn.prepareStatement(sql))
7  {
8              if (item.equalsIgnoreCase("none")==false) {
9                   ps.setString(1,item);
10             }
11             ResultSet rs = ps.executeQuery();
12             Staff obj;
13
14             while(rs.next()){
15                 //Using seventeen-params constructor
16                 obj = new Staff(rs.getInt("staff_id"),
17                         rs.getInt("store_id"),
18                         rs.getString("first_name"),
19                         rs.getString("last_name"),
20                         rs.getString("email"),
21                         rs.getInt("address_id"),
22                         rs.getBoolean("active"),
23                         rs.getString("username"),
24                         rs.getString("password"),
25                         rs.getBytes("picture"),
26                         rs.getTimestamp("last_update"),
27                         rs.getString("address"),
28                         rs.getString("district"),
```

```
                        rs.getString("city"),
                        rs.getString("country"),
                        rs.getString("postal_code"),
                        rs.getString("phone"));

                list.add(obj);
            }
        }catch (SQLException ex){
            JOptionPane.showMessageDialog(frm,
ex.getMessage(),
                "ERROR",JOptionPane.ERROR_MESSAGE);
        }
        return list;
    }

    private static void show_table_staff(StaffForm frm,
ArrayList<Staff> list) throws SQLException{
        DefaultTableModel model = new DefaultTableModel(0,0);

        String header[] = {"Customer ID", "Store ID", "First
Name", "Last Name", "Email", "Address ID", "Active",
"Username", "Password", "Picture", "Address", "District",
"City", "Country", "Postal Code", "Phone", "Last Update"};


model.setColumnIdentifiers(set_column_header(frm.getJTStaff(),
header));
        frm.getJTStaff().setModel(model);

        Object[] row = new Object[17];

        for(int i=0; i<list.size(); i++){
            row[0] = list.get(i).getStaffID();
            row[1] = list.get(i).getStoreID();
            row[2] = list.get(i).getFirstName();
            row[3] = list.get(i).getLastName();
            row[4] = list.get(i).getEmail();
            row[5] = list.get(i).getAddressID();
            row[6] = list.get(i).getActive();
            row[7] = list.get(i).getUserName();
            row[8] = list.get(i).getPassword();
            row[9] = list.get(i).getPicture();
            row[10] = list.get(i).getCustomerAddress();
            row[11] = list.get(i).getCustomerDistrict();
            row[12] = list.get(i).getCustomerCity();
            row[13] = list.get(i).getCustomerCountry();
            row[14] = list.get(i).getCustomerPostalCode();
            row[15] = list.get(i).getCustomerPhone();
            row[16] = list.get(i).getLastUpdate();

            model.addRow(row);
        }
    }
```

Step 2

In **Staff_Utils.java**, define **refresh_controls()** method. It calls **get_staff_l** the database and then calls **show_table_staff()** to populate the table view also populates several comboboxes on the form with data retrieved from th

```
1     public static void
2  refresh_controls(StaffForm frm){
3
4  frm.setLocationRelativeTo(null);
5          frm.setTitle("STAFF FORM");
6
7          //Shows the content of
8  staff table and populates combobox
9          try{
10             //Makes alternating
11 color for table rows
12
13 table_renderer(frm.getJTStaff());
14
15             //Populates table
16             ArrayList<Staff> list =
17 get_staff_list(frm,
18 Query_Staff.get_sql_staff_joint() +
19 " ORDER BY staff_id", "none");
20             show_table_staff(frm,
21 list);
22
23             //Populates jcbStaffID
24             String sql_staff_id =
25  "SELECT staff_id FROM staff ORDER
26 BY staff_id";
27
28 populate_combobox(sql_staff_id,
29 frm.getJCBStaffID(), frm);
30
31             //Populates
32 jcbStaffName
33             String sql_staff_name =
34 "SELECT DISTINCT
35 CONCAT(first_name,' ', last_name)
36 FROM staff ORDER BY
37 CONCAT(first_name,' ', last_name)";
38
39 populate_combobox(sql_staff_name,
40 frm.getJCBStaffName(), frm);
41
42             //Populates jcbStoreID
               String sql_store_id =
 "SELECT store_id FROM store ORDER
BY store_id";
```

```
populate_combobox(sql_store_id,
frm.getJCBStoreID(), frm);

            //Populates jcbActive
            String sql_act =
 "SELECT DISTINCT active FROM
customer ORDER BY active";

populate_combobox(sql_act,
frm.getJCBActive(), frm);

            //Populates
jcbAddressID
            String sql_add_id =
 "SELECT address_id FROM address
ORDER BY address_id";

populate_combobox(sql_add_id,
frm.getJCBAddressID(), frm);

        }catch (SQLException ex){

JOptionPane.showMessageDialog(frm,
ex.getMessage(),

 "ERROR",JOptionPane.ERROR_MESSAGE)
;
        }
    }
```

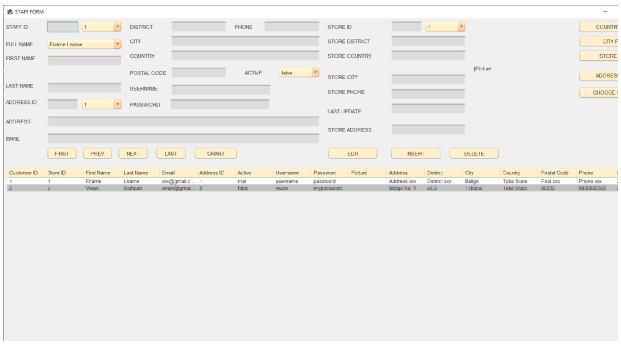


Figure 14.2 The content of **staff** table displayed

Step 3	In **StaffForm**'s default constructor, the **Staff_Utils.refresh_controls(th** the controls in the **StaffForm** with data from a database using the **re Staff_Utils** class. The **this.setIconImage()** method sets the icon of the **S this.setDefaultCloseOperation(this.HIDE_ON_CLOSE)** method sets th to hide the form instead of exiting the application when the close button is

```java
package sakila;
import java.awt.Toolkit;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.IOException;
import java.text.ParseException;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.JButton;
import javax.swing.JComboBox;
import javax.swing.JLabel;
import javax.swing.JTable;
import javax.swing.JTextField;

public class StaffForm extends javax.swing.JFrame {
    public StaffForm() {
        initComponents();
        Utility.setLookAndFeel(this);
        Staff_Utils.refresh_controls(this);

 this.setIconImage(Toolkit.getDefaultToolkit().getImage(getClass().
;
        this.setDefaultCloseOperation(this.HIDE_ON_CLOSE);
    }
    //...
}
```

Step 4

Run the project to see the content of **staff** table displayed in **jtStaff** as sho

If you use the data from **Sakila** MySQL database available in the internet, table displayed in **jtStaff** as shown in Figure 14.3.

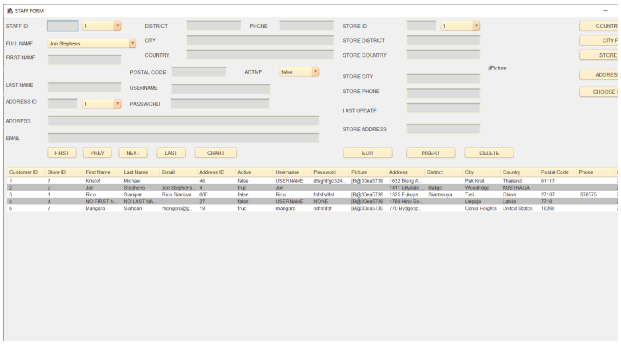


Figure 14.3 The the content of **staff** table in original **Sakila** database a **jtStaff**

DISPLAYING AND NAVIGATING DATA ROW BY ROW
DISPLAYING AND NAVIGATING DATA ROW BY ROW

Step 1

In **Staff_Utils**, define two new methods named **clear_store_cor** **display_store_data()**. The first method, **clear_store_controls(StaffForn** a **CustomerForm** object as input and sets the text of various text fields i empty strings.

The second method, **display_store_data(StaffForm frm, String sql, T i** **StaffForm** object, an SQL query string, and an object of type T as input the SQL query and displays the resulting data row by row in the text field If the query returns no rows, the **clear_store_controls()** method is calle form fields. If the query returns one or more rows, the values of the fields corresponding values from the first row of the result **find_combo_value_selected()** method is also called to select the appro the **jcbStoreID** JComboBox based on the value of **store_id** returned from

```
1     private static void clear_store_controls(StaffForm frm){
2         frm.getJTFStoreID().setText("");
3         frm.getJTFStoreAddress().setText("");
4         frm.getJTFStoreDistrict().setText("");
5         frm.getJTFStorePhone().setText("");
6         frm.getJTFStoreCity().setText("");
7         frm.getJTFStoreCountry().setText("");
8     }
9
10   //Displays store data result row by row
11    private static <T> void display_store_data(StaffForm frm,
12 String sql, T item){
13        try(Connection conn = getConnection()){
14            PreparedStatement ps = conn.prepareStatement(sql);
15            ps.setObject(1,item);
16            ResultSet rs = ps.executeQuery();
17
18            if (!rs.next()) {
19                // no row found, clear the form fields
20                clear_store_controls(frm);
21                return;
22            }
23
24            do{
25
26 frm.getJTFStoreID().setText(String.valueOf(rs.getInt("store_id")));
27
28 frm.getJTFStoreAddress().setText(rs.getString("address"));
29
30 frm.getJTFStoreDistrict().setText(rs.getString("district"));
31
32 frm.getJTFStorePhone().setText(rs.getString("phone"));
```

```
33
34 frm.getJTFStoreCity().setText(rs.getString("city"));
35
36 frm.getJTFStoreCountry().setText(rs.getString("country"));
37
38                 // Determines item selected from jcbStoreID
39                 find_combo_value_selected(frm.getJCBStoreID(),
40 rs.getInt("store_id"));
41
42             }while(rs.next());
43
44             rs.close();
45             ps.close();
          }catch(SQLException ex){
              JOptionPane.showMessageDialog(frm, ex.getMessage(),
                      "ERROR",JOptionPane.ERROR_MESSAGE);
          }
      }
```

Step 2

Still in the same class, define another method named **jcbStore_handler()** handles the event when the user selects an item from the **jcbStoreID** com **StaffForm**. It gets the selected item from the combo box an **display_store_data()** method, passing in the **StaffForm** object and the SC a placeholder for the **store_id** parameter. The selected item is used as t value for the query to retrieve the store data that matches the selected **display_store_data()** method then displays the store data on the form.

```
1     public static void
2 jcbStore_handler(StaffForm frm) {
3         Object item =
4 frm.getJCBStoreID().getSelectedItem();
5         display_store_data(frm,
  Query_Store.get_sql_store_joint() + "
  WHERE store_id = ?", item);
      }
```

Step 3

In **StaffForm**, double click on **jcbStoreID** combobox and define its c event handler as follows:

```
1     private void
2 jcbStoreIDActionPerformed(java.awt.event.ActionEvent
3 evt) {
          Staff_Utils.jcbStore_handler(this);
      }
```

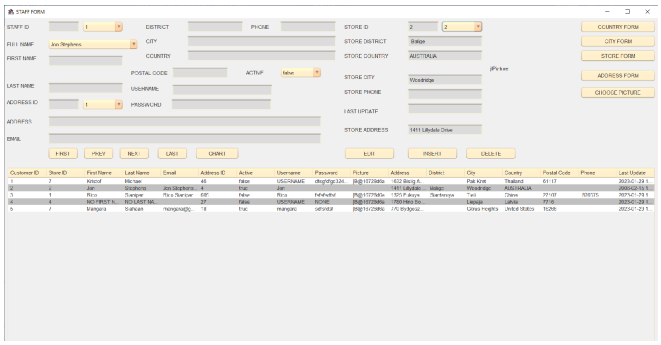


Figure 14.4 Displaying row by row the content of **store** table

Step 4

Run the project. Choose one of items in **jcbStoreID** combobox to see ro content of **store** table as shown in Figure 14.4.

Step 5

In **Staff_Utils**, define two new methods named **clear_address_co display_address_data()**. The **clear_address_controls()** method clears t for the address information on the customer form when it is called. **StaffForm** object as a parameter and sets the text for each of the relevant an empty string.

The **display_address_data()** method displays the address data for a give customer form. It takes in a **StaffForm** object, a SQL string to query the the relevant data, and an item object. The method retrieves the data from and sets the text for the relevant text fields on the form to display the selects the item in the **JComboBox** for the address ID. If no data is found item, the method calls the **clear_address_controls()** method to clear the

```
1     private static void clear_address_controls(StaffForm frm){
2         frm.getJTFAddressID().setText("");
3         frm.getJTFAddress().setText("");
4         frm.getJTFDistrict().setText("");
5         frm.getJTFPhone().setText("");
6         frm.getJTFPostalCode().setText("");
7         frm.getJTFCity().setText("");
8         frm.getJTFCountry().setText("");
9     }
10
11    //Displays address data result row by row
12     private static <T> void display_address_data(CustomerForm frm,
13 String sql, T item){
14         try(Connection conn = getConnection()){
15             PreparedStatement ps = conn.prepareStatement(sql);
16             ps.setObject(1,item);
17             ResultSet rs = ps.executeQuery();
18
19             if (!rs.next()) {
20                 // no row found, clear the form fields
21                 clear_address_controls(frm);
```

```
                return;
            }

            do{

frm.getJTFAddressID().setText(String.valueOf(rs.getInt("address_id"
                frm.getJTFAddress().setText(rs.getString("address")
                frm.getJTFDistrict().setText(rs.getString("district
                frm.getJTFPhone().setText(rs.getString("phone"));

frm.getJTFPostalCode().setText(rs.getString("postal_code"));
                frm.getJTFCity().setText(rs.getString("city"));
                frm.getJTFCountry().setText(rs.getString("country")

                // Determines item selected from jcbAddressID
                find_combo_value_selected(frm.getJCBAddressID(),
rs.getInt("address_id"));

            }while(rs.next());

            rs.close();
            ps.close();
        }catch(SQLException ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }
```

Step 6

Still in the same class, define another method named **jcbAddress_handle** method that handles the selection of an address from the address ID dro the customer form. It gets the selected item from the dropdown list, and t to the **display_address_data()** method along with a SQL query to retriev data associated with that ID.

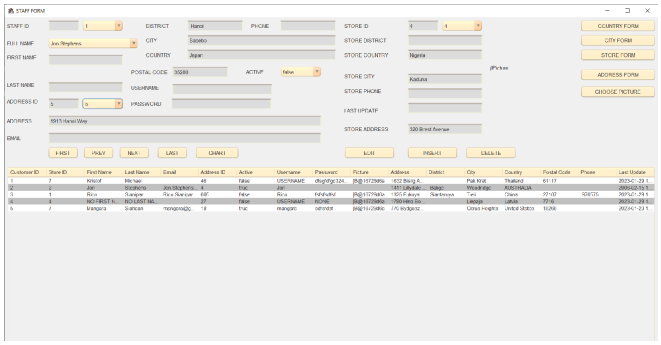
The **display_address_data()** method then executes the SQL query, sets form fields to the retrieved data, and also sets the selected item in the dro match the retrieved ID. If no rows are found in the result set, it clears the

```
    public static void
jcbAddress_handler(StaffForm frm) {
        Object item =
frm.getJCBAddressID().getSelectedItem();
        display_address_data(frm,
Query_Address.get_sql_address_joint() +
" WHERE address_id = ?", item);
    }
```

Step

In **StaffForm**, double click on **jcbAddressID** combobox and define its c

7	event handler as follows:
	
	`private void jcbAddressIDActionPerformed(java.awt.event.ActionEvent evt) {`
	`Staff_Utils.jcbAddress_handler(this);`
	`}`

Step 8	Run the project. Choose one of items in **jcbAddressID** combobox to see rc content of **address** table as shown in Figure 14.5.  **Figure 14.5** Displaying row by row the content of **address** tab
Step 9	In **Staff_Utils**, define two new methods named **clear_staff_cor display_staff_data()**. The **clear_staff_controls()** method takes a **StaffForm** object as an a clears the text fields and image displayed in the form. Specifically, it sets for **jtfStaffID**, **jtfFirstName**, **jtfLastName**, **jtfEmail**, **jtfUsername**, **jtfP jtfLastUpdate** to an empty string and sets the icon for **jlPicture** to null. The **display_staff_data()** method takes a **StaffForm** object, a SQL query object as arguments. It displays the results of the SQL query in the te image displayed in the form. First, it connects to the database and executes the SQL query using the ite parameter. If the result set is empty, the **clear_staff_controls()** metho clear the form fields. If the result set contains data, the method iterates row and populates the text fields with the corresponding data. If an image is included in the result set, the method converts the image ImageIcon object and sets it as the icon for **jlPicture**. If there is no im **jlPicture** icon is set to null. Finally, the method calls the **find_combo_value_selected()** method for th **jcbStaffName**, **jcbAddressID**, **jcbStoreID**, and **jcbActive** combo boxes

which item should be selected based on the corresponding data in the resu

```
    private static void clear_staff_controls(StaffForm frm){
        frm.getJTFStaffID().setText("");
        frm.getJTFFirstName().setText("");
        frm.getJTFLastName().setText("");
        frm.getJTFEmail().setText("");
        frm.getJTFUsername().setText("");
        frm.getJTFPassword().setText("");
        frm.getJTFLastUpdate().setText("");
        frm.getJLPicture().setIcon(null);
    }

   //Displays staff data result row by row
    private static <T> void display_staff_data(StaffForm frm, Strin
item){
        try(Connection conn = getConnection()){
            PreparedStatement ps = conn.prepareStatement(sql);
            ps.setObject(1,item);
            ResultSet rs = ps.executeQuery();

            if (!rs.next()) {
                // no row found, clear the form fields
                clear_staff_controls(frm);
                return;
            }

            do{

frm.getJTFStaffID().setText(String.valueOf(rs.getInt("staff_id")));
                frm.getJTFEmail().setText(rs.getString("email"));
                frm.getJTFFirstName().setText(rs.getString("first_n
                frm.getJTFLastName().setText(rs.getString("last_nam
                frm.getJTFUsername().setText(rs.getString("username
                frm.getJTFPassword().setText(rs.getString("password

frm.getJTFLastUpdate().setText(String.valueOf(rs.getTimestamp("last

                //Displays image
                if(rs.getBytes("picture") != null) {
                    ImageIcon imageIcon = new
ImageIcon(rs.getBytes("picture"));
                    frm.getJLPicture().setIcon(imageIcon);
                } else {
                    frm.getJLPicture().setIcon(null);
                }

                // Determines item selected from jcbStaffID
                find_combo_value_selected(frm.getJCBStaffID(),
rs.getInt("staff_id"));

                // Determines item selected from jcbStaffName
```

```
                String full_name = rs.getString("first_name") + " "
rs.getString("last_name");
                find_combo_value_selected(frm.getJCBStaffName(), fu

                // Determines item selected from jcbAddressID
                find_combo_value_selected(frm.getJCBAddressID(),
rs.getInt("address_id"));

                // Determines item selected from jcbStoreID
                find_combo_value_selected(frm.getJCBStoreID(),
rs.getInt("store_id"));

                // Determines item selected from jcbActive
                find_combo_value_selected(frm.getJCBActive(),
rs.getBoolean("active"));
            }while(rs.next());

            rs.close();
            ps.close();
        }catch(SQLException ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }
```

Step 10

Still in the same class, define another method named **jcbStaff_handler()**. is an event handler for two **JComboBox** components in a **StaffForm**. current selected item from the **JComboBox** component as an argume determines which of the two **JComboBox** components it is handling by cc the reference to the two possible components in the **StaffForm** object.

It then sets the SQL query for retrieving the staff data based on whic **JComboBox** components was selected. This is done by calling a static **Query_Staff** class, which returns the appropriate SQL query based on **JComboBox** component.

Finally, it calls the **display_staff_data()** method to display the data ret SQL query in the form's controls, passing in the **StaffForm** object, the SC the selected item as arguments.

```
    public static void
jcbStaff_handler(StaffForm frm,
JComboBox<String> jcb1) {
        Object item1 =
jcb1.getSelectedItem();
        String sql = "";
        if
(jcb1.equals(frm.getJCBStaffID())) {

```

```
            sql =
Query_Staff.get_sql_id();
        } else if
(jcb1.equals(frm.getJCBStaffName()))
{
            sql =
Query_Staff.get_sql_name();
        }

        display_staff_data(frm, sql,
item1);
    }
```

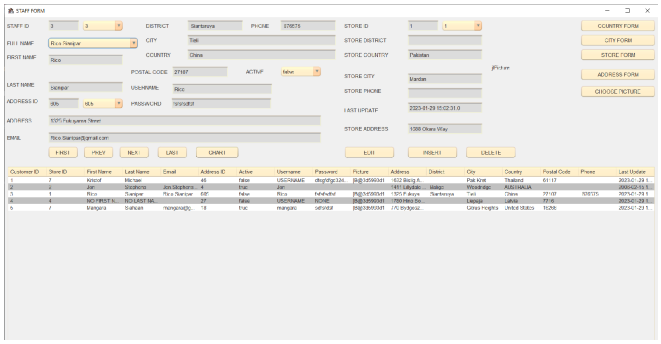


Figure 14.6 Displaying row by row the content of **staff** table

Step 11

In **StaffForm**, double click on **jcbStaffID** and **jcbStaffName** comboboxe their corresponding event handlers as follows:

```
    private void
jcbStaffIDActionPerformed(java.awt.event.ActionEvent
evt) {
        Staff_Utils.jcbStaff_handler(this,
this.jcbStaffID);
    }

    private void
jcbStaffNameActionPerformed(java.awt.event.ActionEvent
evt) {
        Staff_Utils.jcbStaff_handler(this,
this.jcbStaffName);
    }
```

These two methods are event handlers for the **JComboBox** components **jc jcbStaffName** respectively. When an action (e.g. selecting an item) is p the **JComboBox**, the corresponding event handler method is triggered.

	The event handler methods call the **jcbStaff_handler()** method in th class, passing the current instance of the **StaffForm** and the **JComboBo** that triggered the event as parameters. The **jcbStaff_handler()** method selected item from the **JComboBox**, determines which SQL query to exec the **JComboBox** that was passed in, and calls the **display_staff_data** display the data in the form.
Step 12	Run the project. Choose one of items in **jcbStaffID** and/or **jcbStaffName** see row by row the content of **staff** table as shown in Figure 14.6.
Step 13	Define four navigating methods in **Staff_Utils** class. These are methods the first, last, previous, and next rows of data in the **StaffForm**. They navigate through the results of a SQL query executed on the **staff** table. explanation of each method: 1. **show_first_row(StaffForm frm)**: This method displays the first r the staff table. It gets the first item in the staff ID combo box, correspond to the first row in the table, and passes **display_staff_data()** method to display the data. 2. **show_last_row(StaffForm frm)**: This method displays the last r the **staff** table. It gets the last item in the staff ID combo box, correspond to the last row in the table, and passes **display_staff_data()** method to display the data. 3. **show_prev_row(StaffForm frm)**: This method displays the pre data in the **staff** table. It decrements the **currentIndex** variable t previous row, and if it goes below the first index, it sets it bac index. It then gets the item at the new index in the staff ID co passes it to the **display_staff_data()** method to display the data. 4. **show_next_row(StaffForm frm)**: This method displays the next r the staff table. It increments the **currentIndex** variable to mov row, and if it goes above the last index, it sets it back to the last gets the item at the new index in the staff ID combo box and pa **display_staff_data()** method to display the data. All of these methods use the **display_staff_data()** method to actually dis in the form. They also update the **currentIndex** variable, which keeps t row is currently being displayed.

```
1     public static void show_first_row(StaffForm frm){
2         String item =
3 String.valueOf(frm.getJCBStaffID().getItemAt(FIRST_INDEX));
4         display_staff_data(frm, SQL_ID, item);
5         currentIndex = FIRST_INDEX;
6     }
7
8     public static void show_last_row(StaffForm frm){
```

```
        int endIndex = frm.getJCBStaffID().getItemCount() -
1;
        String item =
String.valueOf(frm.getJCBStaffID().getItemAt(endIndex));
        display_staff_data(frm, SQL_ID, item);
        currentIndex = endIndex;
    }

    public static void show_prev_row(StaffForm frm){
        currentIndex--;
        if(currentIndex < FIRST_INDEX){
            currentIndex = FIRST_INDEX;
            return;
        }
        String item =
String.valueOf(frm.getJCBStaffID().getItemAt(currentIndex));
        display_staff_data(frm, SQL_ID, item);
    }

    public static void show_next_row(StaffForm frm){
        int endIndex = frm.getJCBStaffID().getItemCount() -
1;
        currentIndex++;
        if(currentIndex > endIndex){
            currentIndex = endIndex;
            return;
        }
        String item =
String.valueOf(frm.getJCBStaffID().getItemAt(currentIndex));
        display_staff_data(frm, SQL_ID, item);
    }
```

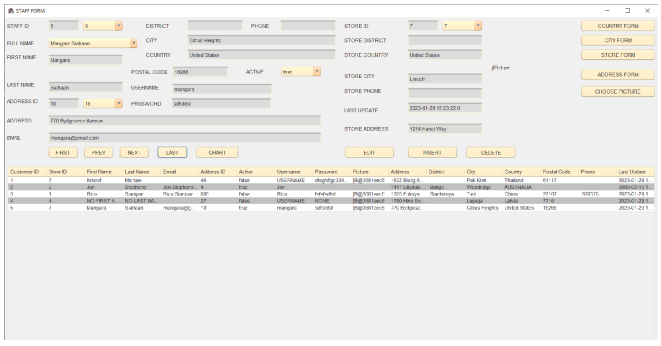


Figure 14.7 User clicks on one or more navigation buttons on staf

Step 14 Then in **StaffForm**, double click on each navigation buttons to corresponding event handler:

```
    private void
jbFirstActionPerformed(java.awt.event.ActionEvent
evt) {
        Staff_Utils.show_first_row(this);
```

```
    }

    private void
jbPrevActionPerformed(java.awt.event.ActionEvent
evt) {
        Staff_Utils.show_prev_row(this);
    }

    private void
jbNextActionPerformed(java.awt.event.ActionEvent
evt) {
        Staff_Utils.show_next_row(this);
    }

    private void
jbLastActionPerformed(java.awt.event.ActionEvent
evt) {
        Staff_Utils.show_last_row(this);
    }
```

These are event handler methods for the "First", "Prev", "Next", and "Las the **StaffForm** GUI. When one of these buttons is clicked, the correspond called, which in turn calls a method from the **Staff_Utils** class to display previous row, next row, or last row of data in the store table, respectiv keyword in the method calls refers to the current **StaffForm** instance, wh as a parameter to the **Staff_Utils** methods.

Step 15

Run the project. Click on one or more navigation buttons to see the resul Figure 14.7.

Step 16

Define **mouse_pressed_handler()** method in **Staff_Utils** class. This me the event when the user clicks on a row in the staff data table. It retrieves row index and checks if a row has been selected or not. If no row is select a message prompting the user to select a row. Otherwise, it retrieves the the first column of the selected row and displays the corresponding sta form.

The method uses a try-with-resources block to establish a database co retrieve the staff data. If an **SQLException** occurs, it logs the error an error message with the exception message and stack trace.

```
    public static void mouse_pressed_handler(StaffForm frm) {
        Objects.requireNonNull(frm, "frm must not be null");

        int selectedIndex = frm.getJTStaff().getSelectedRow();
        if (selectedIndex == -1) {
            JOptionPane.showMessageDialog(frm, "Please select a row to view its
data.",
```

```
                "No row selected", JOptionPane.INFORMATION_MESSAGE)
            return;
        }

        try (Connection conn = getConnection()) {
            String id =
String.valueOf(frm.getJTStaff().getModel().getValueAt(selectedIndex
0));

            // Displays staff data
            display_staff_data(frm, SQL_ID, id);

        } catch (SQLException ex) {

Logger.getLogger(StaffForm.class.getName()).log(Level.SEVERE, "Erro
displaying staff data", ex);
            String message = "Error displaying staff data: " +
ex.getMessage();
            String stackTrace = Arrays.toString(ex.getStackTrace())
            JOptionPane.showMessageDialog(frm, message + "\n\n" +
stackTrace, "ERROR", JOptionPane.ERROR_MESSAGE);
        }
    }
```

Step 17

Right click on **jtStaff**. Then, choose **Events > Mouse > mousePresse** event handler:

```
    private void
jtStaffMousePressed(java.awt.event.MouseEvent
evt) {

 Staff_Utils.mouse_pressed_handler(this);
    }
```

Step 18

Run the project. Double click on any row in **jtStaff** table. You corresponding row of **staff** table displayed in textfields and comboboxes Figure 14.8.

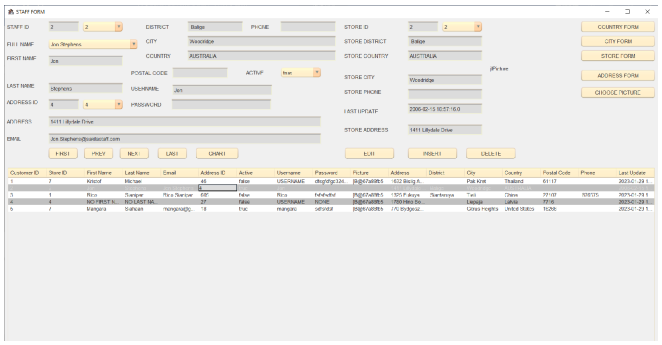


Figure 14.8 User double-clicks on any row in **jtStaff**

UPDATING RECORD
UPDATING RECORD

Step 1

In **Staff_Utils** class, define a new method named **jbChoosePicture_handler()**. It is responsible for handling the user's choice of an image for a staff member. The method uses an instance of the **Utility** class to read the image and display it in the **JLabel** component of the **StaffForm** object.

```
    //If user chooses one of items
in jbChoosePicture
    public static void
jbChoosePicture_handler(StaffForm
frm){
        Utility obj = new
Utility();
        obj.read_image(frm,
frm.getJLPicture());
    }
```

Then, double click on **jbChoosePicture** on **StaffForm** to define its event listener:

```
    private void
jbChoosePictureActionPerformed(java.awt.event.ActionEvent
evt) {
        Staff_Utils.jbChoosePicture_handler(this);
    }
```

Step 2

In **Staff_Utils** class, define a new method named **update_row_by_staff_id()**. The method first creates a connection to the

database using the **getConnection()** method. It then executes a query to check if a row with the given **staff_id** exists in the **staff** table. If the row exists, it creates a new **Staff** object using the given parameters. It then executes an update query to update the row in the staff table with the new data. Finally, it closes the **ResultSets**, **PreparedStatements**, and connection.

If an **SQLException** or **NumberFormatException** is caught, the method logs the error, displays an error message to the user, and closes any open **ResultSets**, **PreparedStatements**, and connection.

```
    //Updates row of data in staff tabel by staff_id
    public static void update_row_by_staff_id(StaffForm frm,
int staff_id, int store_id, String fname, String lname,
        String email, int add_id, boolean active, String
user, String pass, byte[] pict) throws SQLException{
        Connection conn = getConnection();
        ResultSet rs = null;
        String query_id = "SELECT staff_id FROM staff WHERE
staff_id = ?";
        String update_query = """
            UPDATE staff SET store_id = ?, first_name = ?,
last_name = ?,
            email = ?, address_id = ?, active = ?, username =
?, password = ?,
            picture = ? WHERE staff_id = ?""";
        try(PreparedStatement idPs =
conn.prepareStatement(query_id,

ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_READ_ONLY);
        PreparedStatement updatePS =
conn.prepareStatement(update_query,
                ResultSet.TYPE_SCROLL_SENSITIVE,
ResultSet.CONCUR_UPDATABLE))
        {
            idPs.setInt(1,staff_id);
            if(!idPs.execute()){
                String message = "Can't find staff_id " +
staff_id;

                JOptionPane.showMessageDialog(frm, message,
                    "ERROR",JOptionPane.ERROR_MESSAGE);
            } else{
                rs = idPs.getResultSet();
                rs.next();

                //Creates a Staff object using nine-params
constructor
                Staff obj = new Staff(staff_id, store_id,
fname, lname,
                    email, add_id, active, user, pass, pict,
 new Timestamp(System.currentTimeMillis()));
```

```
                updatePS.setInt(1, obj.getStoreID());
                updatePS.setString(2, obj.getFirstName());
                updatePS.setString(3, obj.getLastName());
                updatePS.setString(4, obj.getEmail());
                updatePS.setInt(5, obj.getAddressID());
                updatePS.setBoolean(6, obj.getActive());
                updatePS.setString(7, obj.getUserName());
                updatePS.setString(8, obj.getPassword());
                updatePS.setBytes(9, obj.getPicture());
                updatePS.setInt(10, obj.getStaffID());

                updatePS.executeUpdate();
                rs.close();
                updatePS.close();
                idPs.close();
                conn.close();
            }
        }catch(SQLException ex){

Logger.getLogger(StaffForm.class.getName()).log(Level.SEVERE,
"Error updating customer data", ex);
            String message = "Error updating customer data: "
+ ex.getMessage();
            String stackTrace =
Arrays.toString(ex.getStackTrace());
            JOptionPane.showMessageDialog(null, message +
"\n\n" + stackTrace, "ERROR", JOptionPane.ERROR_MESSAGE);
        }catch(java.lang.NumberFormatException ex){

Logger.getLogger(StaffForm.class.getName()).log(Level.SEVERE,
"Invalid Input", ex);
            String message = "Invalid Input: " +
ex.getMessage();
            String stackTrace =
Arrays.toString(ex.getStackTrace());
            JOptionPane.showMessageDialog(null, message +
"\n\n" + stackTrace, "ERROR", JOptionPane.ERROR_MESSAGE);
        }
    }
```

Step 3	Then in the same class, define a new method **read_inputs()**. It reads input data from a GUI form and performs input validation on the data. The method takes a single argument, which is an instance of the **StaffForm** class. The form contains several input fields, such as staff ID, store ID, first name, last name, email, address ID, active status, username, and password. The method reads the input data from the form fields and stores it in a **HashMap** object called input_data. It then performs input validation on

the data by checking for invalid values and throwing exceptions if necessary. For example, if the address ID is not a valid integer or is negative or zero, the method throws an **IllegalArgumentException** with an error message indicating the problem.

Finally, the method puts the valid input data into the **input_data HashMap** and returns it. The calling code can then use this input data to perform further processing, such as adding a new staff member to a database.

```
    private static HashMap<String, String>
read_inputs(StaffForm frm) {
        HashMap<String, String> input_data = new
HashMap<>();
        String staff_id =
String.valueOf(frm.getJCBStaffID().getSelectedItem());
        String store_id =
String.valueOf(frm.getJCBStoreID().getSelectedItem());
        String add_id =
String.valueOf(frm.getJCBAddressID().getSelectedItem());
        String fname = frm.getJTFFirstName().getText();
        String lname = frm.getJTFLastName().getText();
        String email = frm.getJTFEmail().getText();
        String active =
String.valueOf(frm.getJCBActive().getSelectedItem());
        String user = frm.getJTFUsername().getText();
        String pass = frm.getJTFPassword().getText();

        // Validate user input
        int add_id_int = 0;
        try {
            add_id_int = Integer.parseInt(add_id);
            if (add_id_int <= 0) {
                throw new
IllegalArgumentException("Address ID cannot be negative
or zero");
            }
        } catch (NumberFormatException ex) {
            JOptionPane.showMessageDialog(frm, "Invalid
Address ID: " + add_id,
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        } catch (IllegalArgumentException ex) {
            JOptionPane.showMessageDialog(frm,
ex.getMessage(),
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        }

        int store_id_int = 0;
        try {
            store_id_int = Integer.parseInt(store_id);
```

```
            if (store_id_int <= 0) {
                throw new
IllegalArgumentException("Store ID cannot be negative or
zero");
            }
        } catch (NumberFormatException ex) {
            JOptionPane.showMessageDialog(frm, "Invalid
Store ID: " + store_id,
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        } catch (IllegalArgumentException ex) {
            JOptionPane.showMessageDialog(frm,
ex.getMessage(),
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        }

        int staff_id_int = 0;
        try {
            staff_id_int = Integer.parseInt(staff_id);
            if (staff_id_int <= 0) {
                throw new
IllegalArgumentException("Staff ID cannot be negative or
zero");
            }
        } catch (NumberFormatException ex) {
            JOptionPane.showMessageDialog(frm, "Invalid
Staff ID: " + staff_id,
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        } catch (IllegalArgumentException ex) {
            JOptionPane.showMessageDialog(frm,
ex.getMessage(),
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        }

        if (fname == null || fname.isEmpty()) {
            JOptionPane.showMessageDialog(frm, "First
name cannot be empty",
            "Error", JOptionPane.ERROR_MESSAGE);
            throw new IllegalArgumentException("First
name cannot be empty");
        }

        if (lname == null || lname.isEmpty()) {
            JOptionPane.showMessageDialog(frm, "Last
name cannot be empty",
            "Error", JOptionPane.ERROR_MESSAGE);
            throw new IllegalArgumentException("Last
name cannot be empty");
        }
```

```
        if (user == null || user.isEmpty()) {
            JOptionPane.showMessageDialog(frm, "User
name cannot be empty",
            "Error", JOptionPane.ERROR_MESSAGE);
            throw new IllegalArgumentException("User
name cannot be empty");
        }

        input_data.put("staff_id", staff_id);
        input_data.put("store_id", store_id);
        input_data.put("fname", fname);
        input_data.put("lname", lname);
        input_data.put("email", email);
        input_data.put("add_id", add_id);
        input_data.put("active", active);
        input_data.put("user", user);
        input_data.put("pass", pass);

        return input_data;
    }
```

Step 4

Still in the same class, define another method named **edit_actual()**. It is responsible for updating an existing record of a staff member in the database. It first reads the input data from the form using the **read_inputs()** method, which validates the input and throws exceptions if the input is invalid. Then, it converts the image displayed on the form into a byte array and calls the **update_row_by_staff_id()** method to update the record in the database.

If an exception occurs while updating the record, it displays an error message. Otherwise, it refreshes all objects on the form using the **refresh_controls()** method.

```
1   private static void edit_actual(StaffForm
2  frm) throws ParseException, IOException{
3           try{
4               HashMap<String, String> input_data =
5  read_inputs(frm);
6               int staff_id =
7  Integer.parseInt(input_data.get("staff_id"));
8               int store_id =
9  Integer.parseInt(input_data.get("store_id"));
10              int add_id =
11 Integer.parseInt(input_data.get("add_id"));
12              String fname =
13 input_data.get("fname");
14              String lname =
15 input_data.get("lname");
16              String email =
17 input_data.get("email");
```

```
            String active =
input_data.get("active");
            String user = input_data.get("user");
            String pass = input_data.get("pass");

            boolean bool_active = false;
            if(active.equalsIgnoreCase("true")){
                bool_active = true;
            }

            //Reads image in jLabel and converts
it into byte[]
            BufferedImage image = new
BufferedImage(frm.getJLPicture().getWidth(),
frm.getJLPicture().getHeight(),
BufferedImage.TYPE_INT_RGB);

frm.getJLPicture().paint(image.createGraphics());
            ByteArrayOutputStream baos = new
ByteArrayOutputStream();
            ImageIO.write(image, "jpg", baos);

            byte[] pict = baos.toByteArray();
            update_row_by_staff_id(frm, staff_id,
store_id, fname, lname, email, add_id,
bool_active, user, pass, pict);

            //Refreshes all objects on form
            refresh_controls(frm);

        }catch(SQLException ex){
            JOptionPane.showMessageDialog(frm,
ex.getMessage(),

 "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }
```

Step 5

Lastly, define two new methods named **enable_controls()** and **edit_handler()**. The **enable_controls()** is a private method that takes in a boolean value and a **StaffForm** object as parameters. It sets the enabled state of several GUI controls (such as buttons and text fields) in the **StaffForm** object based on the given boolean value. If the boolean value is true, the controls will be enabled; otherwise, they will be disabled.

edit_handler() is a public static method that takes in a **StaffForm** object as a parameter. It is used as an event handler for the "Edit" button in the **StaffForm**. When the button is clicked, it toggles between two states: "EDIT" and "CONFIRM". If the button text is "EDIT", it changes the text

to "CONFIRM" and calls **enable_controls()** with a false value to disable the controls. If the button text is "CONFIRM", it changes the text back to "EDIT", calls edit_actual to perform the editing operation, and then calls **enable_controls()** with a true value to re-enable the controls. If an exception is thrown during the editing operation (such as a **ParseException** or **IOException**), the exception message is displayed in a dialog box.

```
1     private static void enable_controls(boolean state, StaffForm
2  frm){
3          frm.getJBFirst().setEnabled(state);
4          frm.getJBPrev().setEnabled(state);
5          frm.getJBNext().setEnabled(state);
6          frm.getJBLast().setEnabled(state);
7          frm.getJBInsert().setEnabled(state);
8          frm.getJBDelete().setEnabled(state);
9          frm.getJTFAddressID().setEnabled(state);
10         frm.getJTFStoreID().setEnabled(state);
11         frm.getJTFAddress().setEnabled(state);
12         frm.getJTFDistrict().setEnabled(state);
13         frm.getJTFPhone().setEnabled(state);
14         frm.getJTFPostalCode().setEnabled(state);
15         frm.getJTFCity().setEnabled(state);
16         frm.getJTFCountry().setEnabled(state);
17         frm.getJTFStoreAddress().setEnabled(state);
18         frm.getJTFStoreDistrict().setEnabled(state);
19         frm.getJTFStorePhone().setEnabled(state);
20         frm.getJTFStoreCity().setEnabled(state);
21         frm.getJTFStoreCountry().setEnabled(state);
22         frm.getJTFLastUpdate().setEnabled(state);
23     }
24
25     public static void edit_handler(StaffForm frm){
26         if(frm.getJBEdit().getText().equals("EDIT")){
27             frm.getJBEdit().setText("CONFIRM");
28
29             // Disables controls
30             enable_controls(false, frm);
31         }
32
33         else {
34             try {
35                 frm.getJBEdit().setText("EDIT");
36
37                 // Actual editing
38                 edit_actual(frm);
39
40                 //Enables controls
41                 enable_controls(true, frm);
42             } catch (ParseException ex) {
43
44 Logger.getLogger(Customer_Utils.class.getName()).log(Level.SEVERE,
```

```
45 null, ex);
46             } catch (IOException ex) {
47
48 Logger.getLogger(Staff_Utils.class.getName()).log(Level.SEVERE,
49 null, ex);
               }
            }
        }
```

Step 6

Then, double click on **jbEdit** button on **StaffForm** to define its event listener:

```
1     private void
2 jbEditActionPerformed(java.awt.event.ActionEvent
3 evt) {
          Staff_Utils.edit_handler(this);
      }
```

Step 7

Run the project. Choose **staff_id** using **jcbStaffID** or **jcbStaffName** combobox. Or, you can choose one of rows in **jtStaff** (in this case, **staff_id = 5**). Then, click on EDIT button as shown in Figure 14.9.

Edit any field you want (including choosing staff's picture). Then, click on CONFIRM button. The edited row had been saved into **staff** table as shown in Figure 14.10.

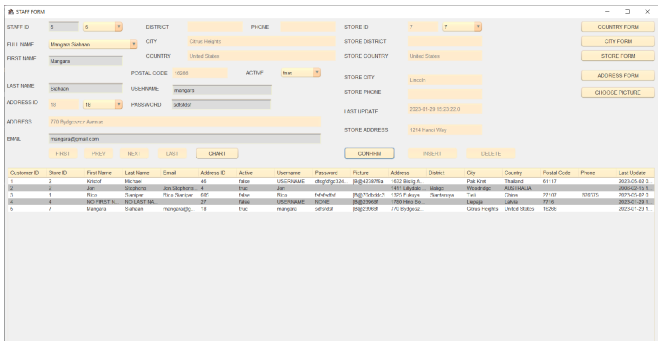


Figure 14.9 The staff form is in editing state

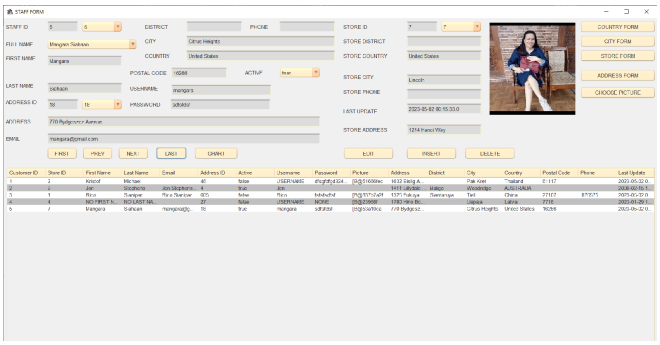


Figure 14.10 The edited row had been saved into database

INSERTING NEW RECORD
INSERTING NEW RECORD

Step 1

In **Staff_Utils** class, define a method named **insert_row()**. This method is responsible for inserting a new row into the **staff** table. Here's a brief explanation of the code:

1. The method first reads input data from the form and converts the active boolean value to a boolean variable.
2. It then reads the image from the **JLabel** component and converts it to a byte[] array.
3. The method then prepares an SQL INSERT statement and sets the parameters using a **PreparedStatement**.
4. A new **Staff** object is created using the input data and the constructor with ten parameters.
5. The staff object's fields are then used to set the parameters of the **PreparedStatement**.
6. Finally, the method executes the INSERT statement by calling the **executeUpdate()** method of the **PreparedStatement**.

If an SQL exception is caught during this process, the method logs the error and displays an error message dialog to the user.

```
1     //Inserts new row into staff table
2     private static void insert_row(StaffForm frm) throws
3  SQLException, ParseException, IOException{
4           HashMap<String, String> input_data =
5  read_inputs(frm);
6           int store_id =
7  Integer.parseInt(input_data.get("store_id"));
8           int add_id =
9  Integer.parseInt(input_data.get("add_id"));
10          String fname = input_data.get("fname");
11          String lname = input_data.get("lname");
```

```
        String email = input_data.get("email");
        String active = input_data.get("active");
        String user = input_data.get("user");
        String pass = input_data.get("pass");

        boolean bool_active = false;
        if(active.equalsIgnoreCase("true")){
            bool_active = true;
        }

        //Reads image in jLabel and converts it into byte[]
        BufferedImage image = new
BufferedImage(frm.getJLPicture().getWidth(),
frm.getJLPicture().getHeight(), BufferedImage.TYPE_INT_RGB);
        frm.getJLPicture().paint(image.createGraphics());
        ByteArrayOutputStream baos = new
ByteArrayOutputStream();
        ImageIO.write(image, "jpg", baos);
        byte[] pict = baos.toByteArray();

        // SQL insert statement
        String sql = """
            INSERT INTO staff(store_id, first_name,
last_name,
                email, address_id, active, username,
password, picture)
                VALUES(?, ?, ?, ?, ?, ?, ?, ?, ?)""";

        try(Connection conn = getConnection();
         PreparedStatement pstmt =
conn.prepareStatement(sql)){

            //Creates a Staff object ten-params constructor
            Staff obj = new Staff(store_id, fname, lname,
email, add_id, bool_active, user, pass, pict,
                    new
Timestamp(System.currentTimeMillis()));
            pstmt.setInt(1,obj.getStoreID());
            pstmt.setString(2,obj.getFirstName());
            pstmt.setString(3,obj.getLastName());
            pstmt.setString(4,obj.getEmail());
            pstmt.setInt(5, obj.getAddressID());
            pstmt.setBoolean(6, obj.getActive());
            pstmt.setString(7, obj.getUserName());
            pstmt.setString(8, obj.getPassword());
            pstmt.setBytes(9, obj.getPicture());

            //Executes the sql insert statement
            pstmt.executeUpdate();
        } catch (SQLException ex) {

Logger.getLogger(StaffForm.class.getName()).log(Level.SEVERE,
"Database error", ex);
```

```
            JOptionPane.showMessageDialog(frm, "Error: Database error\n" + ex.getMessage());
        }
    }
```

Step 2

Still in **Staff_Utils.java**, define **insert_actual()** and **insert_handler()** methods. The **insert_handler()** method is called when the user clicks the "INSERT" button. If the button text is "INSERT," it changes the text to "CONFIRM" and disables the "Edit" button and other controls. It also clears the input fields for staff information. When the user clicks the "CONFIRM" button, the **insert_actual()** method is called, which actually inserts the new staff information into the database using the **insert_row()** method.

After the **insert_actual()** method has finished inserting the new staff information, the "Edit" button and other controls are enabled again, and the input fields are cleared.

```
1   private static void insert_actual(StaffForm frm) throws
2   ParseException{
3           try{
4               insert_row(frm);
5
6               //Refreshes table and comboboxes
7               refresh_controls(frm);
8
9           }catch(SQLException ex){
10              JOptionPane.showMessageDialog(frm, ex.getMessage(),
11                      "ERROR",JOptionPane.ERROR_MESSAGE);
12          } catch (IOException ex) {
13              Logger.getLogger(Staff_Utils.class.getName()).log(Level.SEVERE,
14  null, ex);
15          }
16      }
17
18      public static void insert_handler(StaffForm frm){
19          if(frm.getJBInsert().getText().equals("INSERT")){
20              frm.getJBInsert().setText("CONFIRM");
21
22              //Disables jbEdit
23              frm.getJBEdit().setEnabled(false);
24
25              // Disables controls
26              enable_controls(false, frm);
27              frm.getJCBStaffID().setEnabled(false);
28              frm.getJCBStaffName().setEnabled(false);
29
30              // Clears controls
31              clear_staff_controls(frm);
```

```
32
33              // Enables
34              frm.getJBInsert().setEnabled(true);
35          }
36
37          else {
38              frm.getJBInsert().setText("INSERT");
39
40              try {
41                  // Actual insertion
42                  insert_actual(frm);
43              } catch (ParseException ex) {
44
45  Logger.getLogger(Staff_Utils.class.getName()).log(Level.SEVERE,
46  null, ex);
47              }
48
49              //Enables jbEdit
50              frm.getJBEdit().setEnabled(true);
51
52              //Enables controls
53              enable_controls(true, frm);
54              frm.getJCBStaffID().setEnabled(true);
                frm.getJCBStaffName().setEnabled(true);
            }
        }
```

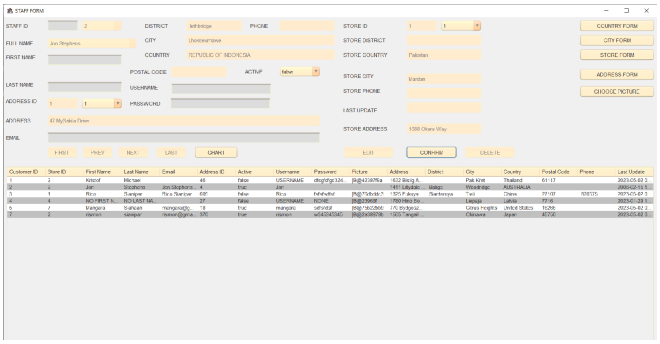


Figure 14.11 When user clicks on INSERT button, the staff form will be in state of insertion

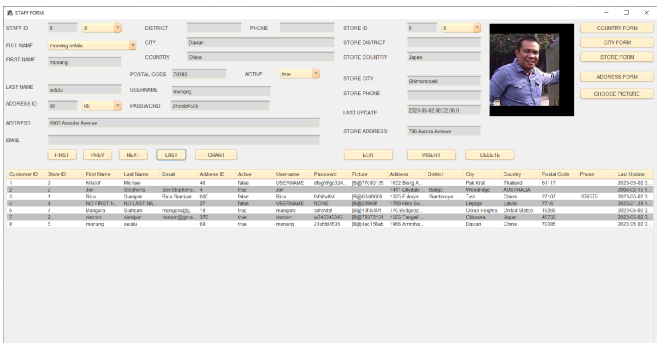


Figure 14.12 The new data had been saved into **staff** table

Step 3

In **StaffForm.java**, double click on INSERT button to create its event listener:

```
    private void jbInsertActionPerformed(java.awt.event.ActionEvent evt) {
        Staff_Utils.insert_handler(this);
    }
```

Step 4

Run the project. Click on INSERT button. You will see the state of staff form when insertion is in progress as shown in Figure 14.11.

Then, fill in all fields. Then, click CONFIRM button to save the new record into **staff** table as shown in Figure 14.12.

DELETING RECORD

DELETING RECORD

Step 1

Then in **Staff_Utils** class, define **delete_handler()** method. The method by displaying a confirmation dialog box to the user to confirm whethe want to delete the row of data. If the user clicks the "Yes" button, the m retrieves the selected staff ID from the **JComboBox** control in the **Staf** object.

Next, the method constructs an SQL DELETE statement using a pre statement to avoid SQL injection attacks. The statement includes a para for the staff ID to be deleted, which is set using the setInt method prepared statement object.

The method then executes the SQL statement using the **executeUp** method of the prepared statement object, which returns the number o affected by the delete operation (in this case, it should be 1).

Finally, the method calls the **refresh_controls()** method to refresh the and comboboxes with the updated data from the database. If an SQL exc occurs during the delete operation, the method displays an error message user using a message dialog box.

```
    public static void delete_handler(StaffForm frm){
        int dialogButton = JOptionPane.YES_NO_OPTION;
```

```
        int staff_id =
Integer.parseInt(String.valueOf(frm.getJCBStaffID().getSelectedItem

        String message = String.format("Are you sure you want to de
the row Staff ID: %d)", staff_id);
        int answer = JOptionPane.showConfirmDialog(frm, message,
"DELETING ROW OF DATA", dialogButton);

        if(answer == JOptionPane.YES_OPTION){
            String query = """
                DELETE FROM staff WHERE staff_id = ?""";
            try(Connection conn = getConnection();
                PreparedStatement ps = conn.prepareStatement(query)
                // Use PreparedStatement to avoid SQL injection att
                ps.setInt(1, staff_id);
                ps.executeUpdate();

                // Refresh table and comboboxes
                refresh_controls(frm);

            } catch (SQLException ex){
                JOptionPane.showMessageDialog(frm, ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
            }
        }
    }
```

Step 2

In **StaffForm.java**, double click on DELETE button to generate its listener:

```
    private void
jbDeleteActionPerformed(java.awt.event.ActionEvent
evt) {
        Staff_Utils.delete_handler(this);
    }
```

Step 3

Run the project. Choose **staff_id** using **jcbStaffID** or **jcbStaffName** coml
Then, Click on DELETE button. The corresponding row of data had been d
from database.

PLOTTING CHART
PLOTTING CHART

Step	Create a new **JFrame** and save it as **Charts_Customer.java**.

1	

Step 2	In **StaffForm.java**, add two **JPanels** and set their corresponding **Variable Name** as **jPanel1** and **jPanel2**. Then, add getter method for each object as follows:
	
	`//Getter method for jPanel1`
	`public JPanel getJPanel1(){`
	`  return this.jPanel1;`
	`}`
	
	`//Getter method for jPanel2`
	`public JPanel getJPanel2(){`
	`  return this.jPanel2;`
	`}`

Step 3	In **Staff_Utils** class, define two new methods. These two methods are responsible for drawing pie charts on the **StaffForm**. They take a **StaffForm** object and a **JPanel** object as parameters. The **JPanel** object is used to draw the pie chart.
	
	The **draw_pie_chart_staff_by_country()** method creates a **DefaultPieDataset** object using the **create_pie_dataset()** method, which returns a dataset object based on the query results for staff distribution by country. Then it calls the **draw_piechart_with_dataset()** method with the dataset object and the title for the chart.
	
	The **draw_pie_chart_staff_by_city()** method is similar, but it creates a dataset based on the query results for staff distribution by city.
	
	Both methods use the **draw_piechart_with_dataset()** method to draw the pie chart on the **JPanel**.
	
	`private static void draw_pie_chart_staff_by_country(StaffForm frm, JPanel jp){`
	`  jp.setPreferredSize(new Dimension(jp.getWidth(), jp.getHeight()));`
	`  DefaultPieDataset dataset = create_pie_dataset(Query_Staff.get_sql_staff_country_dist(), "number_of_staffs", "staff_country");`
	
	`  //Draws piechart staff distribution by country`
	`  draw_piechart_with_dataset(frm, jp, dataset, "TOP 10 STAFF DISTRIBUTION BY COUNTRY");`
	`}`

```
13
14      private static void
15  draw_pie_chart_staff_by_city(StaffForm frm, JPanel jp){
16          jp.setPreferredSize(new Dimension(jp.getWidth(),
17  jp.getHeight()));
18          DefaultPieDataset dataset =
19  create_pie_dataset(Query_Staff.get_sql_staff_city_dist(),
20  "number_of_staffs", "staff_city");
21
22          //Draws piechart staff distribution by city
            draw_piechart_with_dataset(frm, jp, dataset, "TOP 10
STAFF DISTRIBUTION BY CITY");
    }
```

Step 4

In **Staff_Utils** class, define a new method named **jbchart_handler()**.

```
1
2
3
4
```

```
    public static void
jbchart_handler(StaffForm frm){
        //Draws piechart staff
distribution by country

draw_pie_chart_staff_by_country(frm,
frm.getJPanel1());

        //Draws piechart staff
distribution by city

draw_pie_chart_staff_by_city(frm,
frm.getJPanel2());
    }
```

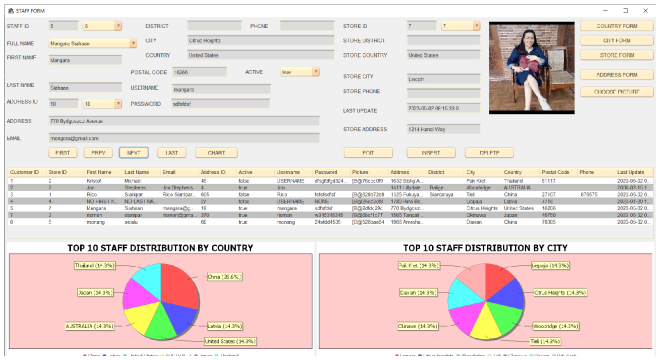


Figure 14.13 The top 10 staff distribution by country and the top 10 staff distribution by city

Step 5	In **StaffForm**, double click on **jbCha** its event listener: `private void jbChartActionPerformed(java.awt. evt) { Staff_Utils.jbchart_hand }`
Step 6	Run the project. Click on CHART b You will see the two charts displaye shown in Figure 14.13.

This is the full version of **Staff_Utils.java**:

```java
package sakila;
import java.awt.Dimension;
import java.awt.image.BufferedImage;
import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.util.logging.Level;
import java.util.logging.Logger;
import java.sql.*;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Calendar;
import java.util.HashMap;
import java.util.Objects;
import javax.imageio.ImageIO;
import javax.swing.ImageIcon;
import javax.swing.JComboBox;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.event.TableModelEvent;
import javax.swing.table.DefaultTableModel;
import javax.swing.table.TableModel;
import org.jfree.data.category.DefaultCategoryDataset;
import org.jfree.data.general.DefaultPieDataset;

public class Staff_Utils extends Utility{
    public static final int FIRST_INDEX = 0;
    public static final int INVALID_INDEX = -1;

    private static int currentIndex = FIRST_INDEX;
    private static final String SQL_ID = Query_Staff.get_sql_id();

    //Creates staff table
    public static void create_staff_table() {
        try (Connection conn = getConnection()) {
            Statement stmt = conn.createStatement();
            stmt.addBatch(Query_Staff.get_sql_staff());
            stmt.executeBatch();

            String message = String.format("Successfully creates staff table'
            JOptionPane.showMessageDialog(null, message,
            "INFORMATION",JOptionPane.INFORMATION_MESSAGE);

        } catch (SQLException ex) {
            JOptionPane.showMessageDialog(null, ex.getMessage(),
                "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    //Populates staff table with some rows of data
    public static void populate_staff_table(){
        try(Connection conn = getConnection()){
```

```
            String sql = """
                INSERT INTO staff(staff_id, store_id, first_name, last_name,
                    email, address_id, active, username, password, picture,
last_update)
                VALUES(?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)""";

            //Creates a new Staff class with default constructor
            PreparedStatement ps1 = conn.prepareStatement(sql);
            Staff obj1 = new Staff();
            ps1.setInt(1,obj1.getStaffID());
            ps1.setInt(2,obj1.getStoreID());
            ps1.setString(3,obj1.getFirstName());
            ps1.setString(4,obj1.getLastName());
            ps1.setString(5,obj1.getEmail());
            ps1.setInt(6,obj1.getAddressID());
            ps1.setBoolean(7,obj1.getActive());
            ps1.setString(8,obj1.getUserName());
            ps1.setString(9,obj1.getPassword());
            ps1.setBytes(10,obj1.getPicture());
            ps1.setTimestamp(11,obj1.getLastUpdate());

            // Creates a new Staff class with eleven-params constructor
            PreparedStatement ps2 = conn.prepareStatement(sql);
            Staff obj2 = new Staff(2, 2, "Vivian", "Siahaan",
"vivian@gmail.com",
                    2, false, "vivian", "mypassword", null, new
Timestamp(System.currentTimeMillis()));
            ps2.setInt(1,obj2.getStaffID());
            ps2.setInt(2,obj2.getStoreID());
            ps2.setString(3,obj2.getFirstName());
            ps2.setString(4,obj2.getLastName());
            ps2.setString(5,obj2.getEmail());
            ps2.setInt(6,obj2.getAddressID());
            ps2.setBoolean(7,obj2.getActive());
            ps2.setString(8,obj2.getUserName());
            ps2.setString(9,obj2.getPassword());
            ps2.setBytes(10,obj2.getPicture());
            ps2.setTimestamp(11,obj2.getLastUpdate());

            ps1.executeUpdate();
            ps2.executeUpdate();

        }catch(SQLException ex){
            JOptionPane.showMessageDialog(null, ex.getMessage(),
                "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    //Reads the content of staff table
    public static void read_staff_table(){
        try(Connection conn = getConnection()){
            Statement stmt = conn.createStatement();
            ResultSet rs = stmt.executeQuery("SELECT * FROM staff");
```

```
            while(rs.next()){
                int staff_id = rs.getInt("staff_id");
                int store_id = rs.getInt("store_id");
                int add_id = rs.getInt("address_id");
                String fname = rs.getString("first_name");
                String lname = rs.getString("last_name");
                String email = rs.getString("email");
                boolean active = rs.getBoolean("active");
                String user = rs.getString("username");
                String pass = rs.getString("password");
                byte[] pict = rs.getBytes("picture");
                Timestamp lu = rs.getTimestamp("last_update");

                //Creates a Staff object using eleven-params constructor
                Staff obj = new  Staff(staff_id, store_id, fname, lname, ema
add_id, active, user, pass, pict, lu);
                System.out.println(obj);
            }
            rs.close();
            stmt.close();

        }catch(SQLException ex){
            JOptionPane.showMessageDialog(null, ex.getMessage(),
                "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    private static ArrayList<Staff> get_staff_list(StaffForm frm, String sql,
String item){
        ArrayList<Staff> list = new ArrayList<>();

        try(Connection conn = getConnection();
            PreparedStatement ps = conn.prepareStatement(sql)){
            if (item.equalsIgnoreCase("none")==false) {
                 ps.setString(1,item);
            }
            ResultSet rs = ps.executeQuery();
            Staff obj;

            while(rs.next()){
                //Using seventeen-params constructor
                obj = new Staff(rs.getInt("staff_id"),
                        rs.getInt("store_id"),
                        rs.getString("first_name"),
                        rs.getString("last_name"),
                        rs.getString("email"),
                        rs.getInt("address_id"),
                        rs.getBoolean("active"),
                        rs.getString("username"),
                        rs.getString("password"),
                        rs.getBytes("picture"),
                        rs.getTimestamp("last_update"),
```

```
                        rs.getString("address"),
                        rs.getString("district"),
                        rs.getString("city"),
                        rs.getString("country"),
                        rs.getString("postal_code"),
                        rs.getString("phone"));

                list.add(obj);
            }
        }catch (SQLException ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
                "ERROR",JOptionPane.ERROR_MESSAGE);
        }
        return list;
    }

    private static void show_table_staff(StaffForm frm, ArrayList<Staff> list
throws SQLException{
        DefaultTableModel model = new DefaultTableModel(0,0);

        String header[] = {"Customer ID", "Store ID", "First Name", "Last
Name", "Email",
            "Address ID", "Active", "Username", "Password", "Picture",
"Address", "District",
            "City", "Country", "Postal Code", "Phone", "Last Update"};

        model.setColumnIdentifiers(set_column_header(frm.getJTStaff(),
header));
        frm.getJTStaff().setModel(model);

        Object[] row = new Object[17];

        for(int i=0; i<list.size(); i++){
            row[0] = list.get(i).getStaffID();
            row[1] = list.get(i).getStoreID();
            row[2] = list.get(i).getFirstName();
            row[3] = list.get(i).getLastName();
            row[4] = list.get(i).getEmail();
            row[5] = list.get(i).getAddressID();
            row[6] = list.get(i).getActive();
            row[7] = list.get(i).getUserName();
            row[8] = list.get(i).getPassword();
            row[9] = list.get(i).getPicture();
            row[10] = list.get(i).getCustomerAddress();
            row[11] = list.get(i).getCustomerDistrict();
            row[12] = list.get(i).getCustomerCity();
            row[13] = list.get(i).getCustomerCountry();
            row[14] = list.get(i).getCustomerPostalCode();
            row[15] = list.get(i).getCustomerPhone();
            row[16] = list.get(i).getLastUpdate();

            model.addRow(row);
        }
```

```
    }

    public static void refresh_controls(StaffForm frm){
        frm.setLocationRelativeTo(null);
        frm.setTitle("STAFF FORM");

        //Shows the content of staff table and populates combobox
        try{
            //Makes alternating color for table rows
            table_renderer(frm.getJTStaff());

            //Populates table
            ArrayList<Staff> list = get_staff_list(frm,
Query_Staff.get_sql_staff_joint() + " ORDER BY staff_id", "none");
            show_table_staff(frm, list);

            //Populates jcbStaffID
            String sql_staff_id = "SELECT staff_id FROM staff ORDER BY
staff_id";
            populate_combobox(sql_staff_id, frm.getJCBStaffID(), frm);

            //Populates jcbStaffName
            String sql_staff_name = "SELECT DISTINCT CONCAT(first_name,' ',
last_name) FROM staff ORDER BY CONCAT(first_name,' ', last_name)";
            populate_combobox(sql_staff_name, frm.getJCBStaffName(), frm);

            //Populates jcbStoreID
            String sql_store_id = "SELECT store_id FROM store ORDER BY
store_id";
            populate_combobox(sql_store_id, frm.getJCBStoreID(), frm);

            //Populates jcbActive
            String sql_act = "SELECT DISTINCT active FROM customer ORDER BY
active";
            populate_combobox(sql_act, frm.getJCBActive(), frm);

            //Populates jcbAddressID
            String sql_add_id = "SELECT address_id FROM address ORDER BY
address_id";
            populate_combobox(sql_add_id, frm.getJCBAddressID(), frm);

        }catch (SQLException ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
            "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    private static void clear_store_controls(StaffForm frm){
        frm.getJTFStoreID().setText("");
        frm.getJTFStoreAddress().setText("");
        frm.getJTFStoreDistrict().setText("");
        frm.getJTFStorePhone().setText("");
        frm.getJTFStoreCity().setText("");
```

```
        frm.getJTFStoreCountry().setText("");
    }

   //Displays store data result row by row
    private static <T> void display_store_data(StaffForm frm, String sql, T
item){
        try(Connection conn = getConnection()){
            PreparedStatement ps = conn.prepareStatement(sql);
            ps.setObject(1,item);
            ResultSet rs = ps.executeQuery();

            if (!rs.next()) {
                // no row found, clear the form fields
                clear_store_controls(frm);
                return;
            }

            do{

frm.getJTFStoreID().setText(String.valueOf(rs.getInt("store_id")));
                frm.getJTFStoreAddress().setText(rs.getString("address"));
                frm.getJTFStoreDistrict().setText(rs.getString("district"));
                frm.getJTFStorePhone().setText(rs.getString("phone"));
                frm.getJTFStoreCity().setText(rs.getString("city"));
                frm.getJTFStoreCountry().setText(rs.getString("country"));

                // Determines item selected from jcbStoreID
                find_combo_value_selected(frm.getJCBStoreID(),
rs.getInt("store_id"));

            }while(rs.next());

            rs.close();
            ps.close();
        }catch(SQLException ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    public static void jcbStore_handler(StaffForm frm) {
        Object item = frm.getJCBStoreID().getSelectedItem();
        display_store_data(frm, Query_Store.get_sql_store_joint() + " WHERE
store_id = ?", item);
    }

    private static void clear_address_controls(StaffForm frm){
        frm.getJTFAddressID().setText("");
        frm.getJTFAddress().setText("");
        frm.getJTFDistrict().setText("");
        frm.getJTFPhone().setText("");
        frm.getJTFPostalCode().setText("");
        frm.getJTFCity().setText("");
```

```
        frm.getJTFCountry().setText("");
    }

   //Displays address data result row by row
    private static <T> void display_address_data(StaffForm frm, String sql, T
item){
        try(Connection conn = getConnection()){
            PreparedStatement ps = conn.prepareStatement(sql);
            ps.setObject(1,item);
            ResultSet rs = ps.executeQuery();

            if (!rs.next()) {
                // no row found, clear the form fields
                clear_address_controls(frm);
                return;
            }

            do{

frm.getJTFAddressID().setText(String.valueOf(rs.getInt("address_id")));
                frm.getJTFAddress().setText(rs.getString("address"));
                frm.getJTFDistrict().setText(rs.getString("district"));
                frm.getJTFPhone().setText(rs.getString("phone"));
                frm.getJTFPostalCode().setText(rs.getString("postal_code"));
                frm.getJTFCity().setText(rs.getString("city"));
                frm.getJTFCountry().setText(rs.getString("country"));

                // Determines item selected from jcbAddressID
                find_combo_value_selected(frm.getJCBAddressID(),
rs.getInt("address_id"));

            }while(rs.next());

            rs.close();
            ps.close();
        }catch(SQLException ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    public static void jcbAddress_handler(StaffForm frm) {
        Object item = frm.getJCBAddressID().getSelectedItem();
        display_address_data(frm, Query_Address.get_sql_address_joint() + "
WHERE address_id = ?", item);
    }

    private static void clear_staff_controls(StaffForm frm){
        frm.getJTFStaffID().setText("");
        frm.getJTFFirstName().setText("");
        frm.getJTFLastName().setText("");
        frm.getJTFEmail().setText("");
        frm.getJTFUsername().setText("");
```

```
        frm.getJTFPassword().setText("");
        frm.getJTFLastUpdate().setText("");
        frm.getJLPicture().setIcon(null);
    }

   //Displays staff data result row by row
    private static <T> void display_staff_data(StaffForm frm, String sql, T
item){
        try(Connection conn = getConnection()){
            PreparedStatement ps = conn.prepareStatement(sql);
            ps.setObject(1,item);
            ResultSet rs = ps.executeQuery();

            if (!rs.next()) {
                // no row found, clear the form fields
                clear_staff_controls(frm);
                return;
            }

            do{

frm.getJTFStaffID().setText(String.valueOf(rs.getInt("staff_id")));
                frm.getJTFEmail().setText(rs.getString("email"));
                frm.getJTFFirstName().setText(rs.getString("first_name"));
                frm.getJTFLastName().setText(rs.getString("last_name"));
                frm.getJTFUsername().setText(rs.getString("username"));
                frm.getJTFPassword().setText(rs.getString("password"));

frm.getJTFLastUpdate().setText(String.valueOf(rs.getTimestamp("last_update"))

                //Displays image
                if(rs.getBytes("picture") != null) {
                    ImageIcon imageIcon = new
ImageIcon(rs.getBytes("picture"));
                    frm.getJLPicture().setIcon(imageIcon);
                } else {
                    frm.getJLPicture().setIcon(null);
                }

                // Determines item selected from jcbStaffID
                find_combo_value_selected(frm.getJCBStaffID(),
rs.getInt("staff_id"));

                // Determines item selected from jcbStaffName
                String full_name = rs.getString("first_name") + " " +
rs.getString("last_name");
                find_combo_value_selected(frm.getJCBStaffName(), full_name);

                // Determines item selected from jcbAddressID
                find_combo_value_selected(frm.getJCBAddressID(),
rs.getInt("address_id"));

                // Determines item selected from jcbStoreID
```

```
                find_combo_value_selected(frm.getJCBStoreID(),
rs.getInt("store_id"));

                // Determines item selected from jcbActive
                find_combo_value_selected(frm.getJCBActive(),
rs.getBoolean("active"));
            }while(rs.next());

            rs.close();
            ps.close();
        }catch(SQLException ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    public static void jcbStaff_handler(StaffForm frm, JComboBox<String> jcb1
{
        Object item1 = jcb1.getSelectedItem();
        String sql = "";
        if (jcb1.equals(frm.getJCBStaffID())) {
            sql = Query_Staff.get_sql_id();
        } else if (jcb1.equals(frm.getJCBStaffName())) {
            sql = Query_Staff.get_sql_name();
        }

        display_staff_data(frm, sql, item1);
    }

    public static void show_first_row(StaffForm frm){
        String item =
String.valueOf(frm.getJCBStaffID().getItemAt(FIRST_INDEX));
        display_staff_data(frm, SQL_ID, item);
        currentIndex = FIRST_INDEX;
    }

    public static void show_last_row(StaffForm frm){
        int endIndex = frm.getJCBStaffID().getItemCount() - 1;
        String item = String.valueOf(frm.getJCBStaffID().getItemAt(endIndex)
        display_staff_data(frm, SQL_ID, item);
        currentIndex = endIndex;
    }

    public static void show_prev_row(StaffForm frm){
        currentIndex--;
        if(currentIndex < FIRST_INDEX){
            currentIndex = FIRST_INDEX;
            return;
        }
        String item =
String.valueOf(frm.getJCBStaffID().getItemAt(currentIndex));
        display_staff_data(frm, SQL_ID, item);
    }
```

```
    public static void show_next_row(StaffForm frm){
        int endIndex = frm.getJCBStaffID().getItemCount() - 1;
        currentIndex++;
        if(currentIndex > endIndex){
            currentIndex = endIndex;
            return;
        }
        String item =
String.valueOf(frm.getJCBStaffID().getItemAt(currentIndex));
        display_staff_data(frm, SQL_ID, item);
    }

    public static void mouse_pressed_handler(StaffForm frm) {
        Objects.requireNonNull(frm, "frm must not be null");

        int selectedIndex = frm.getJTStaff().getSelectedRow();
        if (selectedIndex == -1) {
            JOptionPane.showMessageDialog(frm, "Please select a row to view 
data.",
                "No row selected", JOptionPane.INFORMATION_MESSAGE);
            return;
        }

        try (Connection conn = getConnection()) {
            String id =
String.valueOf(frm.getJTStaff().getModel().getValueAt(selectedIndex, 0));

            // Displays staff data
            display_staff_data(frm, SQL_ID, id);

        } catch (SQLException ex) {
            Logger.getLogger(StaffForm.class.getName()).log(Level.SEVERE,
"Error displaying staff data", ex);
            String message = "Error displaying staff data: " + ex.getMessage
            String stackTrace = Arrays.toString(ex.getStackTrace());
            JOptionPane.showMessageDialog(frm, message + "\n\n" + stackTrace,
"ERROR", JOptionPane.ERROR_MESSAGE);
        }
    }

    //If user chooses one of items in jbChoosePicture
    public static void jbChoosePicture_handler(StaffForm frm){
        Utility obj = new Utility();
        obj.read_image(frm, frm.getJLPicture());
    }

    //Updates row of data in staff tabel by staff_id
    public static void update_row_by_staff_id(StaffForm frm, int staff_id, in
store_id, String fname, String lname,
        String email, int add_id, boolean active, String user, String pass,
byte[] pict) throws SQLException{
        Connection conn = getConnection();
```

```
        ResultSet rs = null;
        String query_id = "SELECT staff_id FROM staff WHERE staff_id = ?";
        String update_query = """
            UPDATE staff SET store_id = ?, first_name = ?, last_name = ?,
            email = ?, address_id = ?, active = ?, username = ?, password = ?
            picture = ? WHERE staff_id = ?""";
        try(PreparedStatement idPs = conn.prepareStatement(query_id,

ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_READ_ONLY);
        PreparedStatement updatePS = conn.prepareStatement(update_query,
                ResultSet.TYPE_SCROLL_SENSITIVE, ResultSet.CONCUR_UPDATABLE))
        {
            idPs.setInt(1,staff_id);
            if(!idPs.execute()){
                String message = "Can't find staff_id " + staff_id;

                JOptionPane.showMessageDialog(frm, message,
                    "ERROR",JOptionPane.ERROR_MESSAGE);
            } else{
                rs = idPs.getResultSet();
                rs.next();

                //Creates a Staff object using nine-params constructor
                Staff obj = new Staff(staff_id, store_id, fname, lname,
                    email, add_id, active, user, pass, pict, new
Timestamp(System.currentTimeMillis()));
                updatePS.setInt(1, obj.getStoreID());
                updatePS.setString(2, obj.getFirstName());
                updatePS.setString(3, obj.getLastName());
                updatePS.setString(4, obj.getEmail());
                updatePS.setInt(5, obj.getAddressID());
                updatePS.setBoolean(6, obj.getActive());
                updatePS.setString(7, obj.getUserName());
                updatePS.setString(8, obj.getPassword());
                updatePS.setBytes(9, obj.getPicture());
                updatePS.setInt(10, obj.getStaffID());

                updatePS.executeUpdate();
                rs.close();
                updatePS.close();
                idPs.close();
                conn.close();
            }
        }catch(SQLException ex){
            Logger.getLogger(StaffForm.class.getName()).log(Level.SEVERE,
"Error updating customer data", ex);
            String message = "Error updating customer data: " +
ex.getMessage();
            String stackTrace = Arrays.toString(ex.getStackTrace());
            JOptionPane.showMessageDialog(null, message + "\n\n" + stackTrace
"ERROR", JOptionPane.ERROR_MESSAGE);
        }catch(java.lang.NumberFormatException ex){
            Logger.getLogger(StaffForm.class.getName()).log(Level.SEVERE,
"Invalid Input", ex);
```

```
            String message = "Invalid Input: " + ex.getMessage();
            String stackTrace = Arrays.toString(ex.getStackTrace());
            JOptionPane.showMessageDialog(null, message + "\n\n" + stackTrace
"ERROR", JOptionPane.ERROR_MESSAGE);
        }
    }

    private static HashMap<String, String> read_inputs(StaffForm frm) {
        HashMap<String, String> input_data = new HashMap<>();
        String staff_id =
String.valueOf(frm.getJCBStaffID().getSelectedItem());
        String store_id =
String.valueOf(frm.getJCBStoreID().getSelectedItem());
        String add_id =
String.valueOf(frm.getJCBAddressID().getSelectedItem());
        String fname = frm.getJTFFirstName().getText();
        String lname = frm.getJTFLastName().getText();
        String email = frm.getJTFEmail().getText();
        String active = String.valueOf(frm.getJCBActive().getSelectedItem());
        String user = frm.getJTFUsername().getText();
        String pass = frm.getJTFPassword().getText();

        // Validate user input
        int add_id_int = 0;
        try {
            add_id_int = Integer.parseInt(add_id);
            if (add_id_int <= 0) {
                throw new IllegalArgumentException("Address ID cannot be
negative or zero");
            }
        } catch (NumberFormatException ex) {
            JOptionPane.showMessageDialog(frm, "Invalid Address ID: " + add_i
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        } catch (IllegalArgumentException ex) {
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        }

        int store_id_int = 0;
        try {
            store_id_int = Integer.parseInt(store_id);
            if (store_id_int <= 0) {
                throw new IllegalArgumentException("Store ID cannot be negati
or zero");
            }
        } catch (NumberFormatException ex) {
            JOptionPane.showMessageDialog(frm, "Invalid Store ID: " + store_i
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        } catch (IllegalArgumentException ex) {
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
```

```
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        }

        int staff_id_int = 0;
        try {
            staff_id_int = Integer.parseInt(staff_id);
            if (staff_id_int <= 0) {
                throw new IllegalArgumentException("Staff ID cannot be negat
or zero");
            }
        } catch (NumberFormatException ex) {
            JOptionPane.showMessageDialog(frm, "Invalid Staff ID: " + staff_
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        } catch (IllegalArgumentException ex) {
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        }

        if (fname == null || fname.isEmpty()) {
            JOptionPane.showMessageDialog(frm, "First name cannot be empty",
            "Error", JOptionPane.ERROR_MESSAGE);
            throw new IllegalArgumentException("First name cannot be empty");
        }

        if (lname == null || lname.isEmpty()) {
            JOptionPane.showMessageDialog(frm, "Last name cannot be empty",
            "Error", JOptionPane.ERROR_MESSAGE);
            throw new IllegalArgumentException("Last name cannot be empty");
        }

        if (user == null || user.isEmpty()) {
            JOptionPane.showMessageDialog(frm, "User name cannot be empty",
            "Error", JOptionPane.ERROR_MESSAGE);
            throw new IllegalArgumentException("User name cannot be empty");
        }

        input_data.put("staff_id", staff_id);
        input_data.put("store_id", store_id);
        input_data.put("fname", fname);
        input_data.put("lname", lname);
        input_data.put("email", email);
        input_data.put("add_id", add_id);
        input_data.put("active", active);
        input_data.put("user", user);
        input_data.put("pass", pass);

        return input_data;
    }
```

```
    private static void edit_actual(StaffForm frm) throws ParseException,
IOException{
        try{
            HashMap<String, String> input_data = read_inputs(frm);
            int staff_id = Integer.parseInt(input_data.get("staff_id"));
            int store_id = Integer.parseInt(input_data.get("store_id"));
            int add_id = Integer.parseInt(input_data.get("add_id"));
            String fname = input_data.get("fname");
            String lname = input_data.get("lname");
            String email = input_data.get("email");
            String active = input_data.get("active");
            String user = input_data.get("user");
            String pass = input_data.get("pass");

            boolean bool_active = false;
            if(active.equalsIgnoreCase("true")){
                bool_active = true;
            }

            //Reads image in jLabel and converts it into byte[]
            BufferedImage image = new
BufferedImage(frm.getJLPicture().getWidth(), frm.getJLPicture().getHeight(),
BufferedImage.TYPE_INT_RGB);
            frm.getJLPicture().paint(image.createGraphics());
            ByteArrayOutputStream baos = new ByteArrayOutputStream();
            ImageIO.write(image, "jpg", baos);
            byte[] pict = baos.toByteArray();

            update_row_by_staff_id(frm, staff_id, store_id, fname, lname,
email, add_id, bool_active, user, pass, pict);

            //Refreshes all objects on form
            refresh_controls(frm);

        }catch(SQLException ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    private static void enable_controls(boolean state, StaffForm frm){
        frm.getJBFirst().setEnabled(state);
        frm.getJBPrev().setEnabled(state);
        frm.getJBNext().setEnabled(state);
        frm.getJBLast().setEnabled(state);
        frm.getJBInsert().setEnabled(state);
        frm.getJBDelete().setEnabled(state);
        frm.getJTFAddressID().setEnabled(state);
        frm.getJTFStoreID().setEnabled(state);
        frm.getJTFAddress().setEnabled(state);
        frm.getJTFDistrict().setEnabled(state);
        frm.getJTFPhone().setEnabled(state);
        frm.getJTFPostalCode().setEnabled(state);
```

```
        frm.getJTFCity().setEnabled(state);
        frm.getJTFCountry().setEnabled(state);
        frm.getJTFStoreAddress().setEnabled(state);
        frm.getJTFStoreDistrict().setEnabled(state);
        frm.getJTFStorePhone().setEnabled(state);
        frm.getJTFStoreCity().setEnabled(state);
        frm.getJTFStoreCountry().setEnabled(state);
        frm.getJTFLastUpdate().setEnabled(state);
    }

    public static void edit_handler(StaffForm frm){
        if(frm.getJBEdit().getText().equals("EDIT")){
            frm.getJBEdit().setText("CONFIRM");

            // Disables controls
            enable_controls(false, frm);
        }

        else {
            try {
                frm.getJBEdit().setText("EDIT");

                // Actual editing
                edit_actual(frm);

                //Enables controls
                enable_controls(true, frm);
            } catch (ParseException ex) {
                Logger.getLogger(Staff_Utils.class.getName()).log(Level.SEVE
null, ex);
            } catch (IOException ex) {
                Logger.getLogger(Staff_Utils.class.getName()).log(Level.SEVE
null, ex);
            }
        }
    }

    //Inserts new row into staff table
    private static void insert_row(StaffForm frm) throws SQLException,
ParseException, IOException{
        HashMap<String, String> input_data = read_inputs(frm);
        int store_id = Integer.parseInt(input_data.get("store_id"));
        int add_id = Integer.parseInt(input_data.get("add_id"));
        String fname = input_data.get("fname");
        String lname = input_data.get("lname");
        String email = input_data.get("email");
        String active = input_data.get("active");
        String user = input_data.get("user");
        String pass = input_data.get("pass");

        boolean bool_active = false;
        if(active.equalsIgnoreCase("true")){
            bool_active = true;
        }
```

```
        //Reads image in jLabel and converts it into byte[]
        BufferedImage image = new BufferedImage(frm.getJLPicture().getWidth(),
frm.getJLPicture().getHeight(), BufferedImage.TYPE_INT_RGB);
        frm.getJLPicture().paint(image.createGraphics());
        ByteArrayOutputStream baos = new ByteArrayOutputStream();
        ImageIO.write(image, "jpg", baos);
        byte[] pict = baos.toByteArray();

        // SQL insert statement
        String sql = """
            INSERT INTO staff(store_id, first_name, last_name,
                email, address_id, active, username, password, picture)
                VALUES(?, ?, ?, ?, ?, ?, ?, ?, ?)""";

        try(Connection conn = getConnection();
         PreparedStatement pstmt = conn.prepareStatement(sql)){

            //Creates a Staff object ten-params constructor
            Staff obj = new Staff(store_id, fname, lname, email, add_id,
bool_active, user, pass, pict,
                    new Timestamp(System.currentTimeMillis()));
            pstmt.setInt(1,obj.getStoreID());
            pstmt.setString(2,obj.getFirstName());
            pstmt.setString(3,obj.getLastName());
            pstmt.setString(4,obj.getEmail());
            pstmt.setInt(5, obj.getAddressID());
            pstmt.setBoolean(6, obj.getActive());
            pstmt.setString(7, obj.getUserName());
            pstmt.setString(8, obj.getPassword());
            pstmt.setBytes(9, obj.getPicture());

            //Executes the sql insert statement
            pstmt.executeUpdate();
        } catch (SQLException ex) {
            Logger.getLogger(StaffForm.class.getName()).log(Level.SEVERE,
"Database error", ex);
            JOptionPane.showMessageDialog(frm, "Error: Database error\n" +
ex.getMessage());
        }
    }

    private static void insert_actual(StaffForm frm) throws ParseException{
        try{
            insert_row(frm);

            //Refreshes table and comboboxes
            refresh_controls(frm);

        }catch(SQLException ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
        } catch (IOException ex) {
```

```
            Logger.getLogger(Staff_Utils.class.getName()).log(Level.SEVERE,
null, ex);
        }
    }

    public static void insert_handler(StaffForm frm){
        if(frm.getJBInsert().getText().equals("INSERT")){
            frm.getJBInsert().setText("CONFIRM");

            //Disables jbEdit
            frm.getJBEdit().setEnabled(false);

            // Disables controls
            enable_controls(false, frm);
            frm.getJCBStaffID().setEnabled(false);
            frm.getJCBStaffName().setEnabled(false);

            // Clears controls
            clear_staff_controls(frm);

            // Enables
            frm.getJBInsert().setEnabled(true);
        }

        else {
            frm.getJBInsert().setText("INSERT");

            try {
                // Actual insertion
                insert_actual(frm);
            } catch (ParseException ex) {
                Logger.getLogger(Staff_Utils.class.getName()).log(Level.SEVE
null, ex);
            }

            //Enables jbEdit
            frm.getJBEdit().setEnabled(true);

            //Enables controls
            enable_controls(true, frm);
            frm.getJCBStaffID().setEnabled(true);
            frm.getJCBStaffName().setEnabled(true);
        }
    }

    public static void delete_handler(StaffForm frm){
        int dialogButton = JOptionPane.YES_NO_OPTION;
        int staff_id =
Integer.parseInt(String.valueOf(frm.getJCBStaffID().getSelectedItem()));

        String message = String.format("Are you sure you want to delete the
Staff ID: %d)", staff_id);
```

```
        int answer = JOptionPane.showConfirmDialog(frm, message, "DELETING R(
OF DATA", dialogButton);

        if(answer == JOptionPane.YES_OPTION){
            String query = """
                DELETE FROM staff WHERE staff_id = ?""";
            try(Connection conn = getConnection();
                PreparedStatement ps = conn.prepareStatement(query)){
                // Use PreparedStatement to avoid SQL injection attacks
                ps.setInt(1, staff_id);
                ps.executeUpdate();

                // Refresh table and comboboxes
                refresh_controls(frm);

            } catch (SQLException ex){
                JOptionPane.showMessageDialog(frm, ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
            }
        }
    }

    private static void draw_pie_chart_staff_by_country(StaffForm frm, JPanel
jp){
        jp.setPreferredSize(new Dimension(jp.getWidth(), jp.getHeight()));
        DefaultPieDataset dataset =
create_pie_dataset(Query_Staff.get_sql_staff_country_dist(),
"number_of_staffs", "staff_country");

        //Draws piechart staff distribution by country
        draw_piechart_with_dataset(frm, jp, dataset, "TOP 10 STAFF DISTRIBUTI
BY COUNTRY");
    }

    private static void draw_pie_chart_staff_by_city(StaffForm frm, JPanel jp
        jp.setPreferredSize(new Dimension(jp.getWidth(), jp.getHeight()));
        DefaultPieDataset dataset =
create_pie_dataset(Query_Staff.get_sql_staff_city_dist(), "number_of_staffs",
"staff_city");

        //Draws piechart staff distribution by city
        draw_piechart_with_dataset(frm, jp, dataset, "TOP 10 STAFF DISTRIBUTI
BY CITY");
    }

    public static void jbchart_handler(StaffForm frm){
        //Draws piechart staff distribution by country
        draw_pie_chart_staff_by_country(frm, frm.getJPanel1());

        //Draws piechart staff distribution by city
        draw_pie_chart_staff_by_city(frm, frm.getJPanel2());
    }
}
```

This is the full version of **StaffForm.java**:

```
package sakila;
import java.awt.Toolkit;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JComboBox;
import javax.swing.JLabel;
import javax.swing.JMenuItem;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JPopupMenu;
import javax.swing.JTable;
import javax.swing.JTextField;

public class StaffForm extends javax.swing.JFrame {
    public StaffForm() {
        initComponents();
        Utility.setLookAndFeel(this);
        Staff_Utils.refresh_controls(this);

 this.setIconImage(Toolkit.getDefaultToolkit().getImage(getClass().getResourc
;
        this.setDefaultCloseOperation(this.HIDE_ON_CLOSE);
    }

    //Getter method for jtStaff
    public JTable getJTStaff(){
        return this.jtStaff;
    }

    //Getter method for jlPicture
    public JLabel getJLPicture(){
        return this.jlPicture;
    }

    //Getter method for jtfStaffID
    public JTextField getJTFStaffID(){
        return this.jtfStaffID;
    }

    //Getter method for jcbStaffID
    public JComboBox getJCBStaffID(){
        return this.jcbStaffID;
    }

    //Getter method for jcbStaffName
    public JComboBox getJCBStaffName(){
```

```
        return this.jcbStaffName;
    }

    //Getter method for jtfFirstName
    public JTextField getJTFFirstName(){
        return this.jtfFirstName;
    }

    //Getter method for jtfLastName
    public JTextField getJTFLastName(){
        return this.jtfLastName;
    }

    //Getter method for jtfAddressID
    public JTextField getJTFAddressID(){
        return this.jtfAddressID;
    }

    //Getter method for jcbAddressID
    public JComboBox getJCBAddressID(){
        return this.jcbAddressID;
    }

    //Getter method for jtfAddress
    public JTextField getJTFAddress(){
        return this.jtfAddress;
    }

    //Getter method for jtfEmail
    public JTextField getJTFEmail(){
        return this.jtfEmail;
    }

    //Getter method for jtfDistrict
    public JTextField getJTFDistrict(){
        return this.jtfDistrict;
    }

    //Getter method for jtfCity
    public JTextField getJTFCity(){
        return this.jtfCity;
    }

    //Getter method for jtfCountry
    public JTextField getJTFCountry(){
        return this.jtfCountry;
    }

    //Getter method for jtfPostalCode
    public JTextField getJTFPostalCode(){
        return this.jtfPostalCode;
    }
```

```
    //Getter method for jtfPhone
    public JTextField getJTFPhone(){
        return this.jtfPhone;
    }

    //Getter method for jcbActive
    public JComboBox getJCBActive(){
        return this.jcbActive;
    }

    //Getter method for jcbStoreID
    public JComboBox getJCBStoreID(){
        return this.jcbStoreID;
    }

    //Getter method for jtfStoreID
    public JTextField getJTFStoreID(){
        return this.jtfStoreID;
    }

    //Getter method for jtfLastUpdate
    public JTextField getJTFLastUpdate(){
        return this.jtfLastUpdate;
```

```
    }

    //Getter method for jtfUsername
    public JTextField getJTFUsername(){
        return this.jtfUsername;
    }

    //Getter method for jtfPassword
    public JTextField getJTFPassword(){
        return this.jtfPassword;
    }

    //Getter method for jtfStoreDistrict
    public JTextField getJTFStoreDistrict(){
        return this.jtfStoreDistrict;
    }

    //Getter method for jtfStoreCity
    public JTextField getJTFStoreCity(){
        return this.jtfStoreCity;
    }

    //Getter method for jtfStoreCountry
    public JTextField getJTFStoreCountry(){
        return this.jtfStoreCountry;
    }

    //Getter method for jtfStorePhone
    public JTextField getJTFStorePhone(){
        return this.jtfStorePhone;
    }

    //Getter method for jtfStoreAddress
    public JTextField getJTFStoreAddress(){
        return this.jtfStoreAddress;
    }

    //Getter method for jbEdit
    public JButton getJBEdit(){
        return this.jbEdit;
    }

    //Getter method for jbInsert
    public JButton getJBInsert(){
        return this.jbInsert;
    }

    //Getter method for jbDelete
    public JButton getJBDelete(){
        return this.jbDelete;
    }
```

```
    //Getter method for jbChart
    public JButton getJBChart(){
        return this.jbChart;
    }

    //Getter method for jbFirst
    public JButton getJBFirst(){
        return this.jbFirst;
    }

    //Getter method for jbPrev
    public JButton getJBPrev(){
        return this.jbPrev;
    }

    //Getter method for jbNext
    public JButton getJBNext(){
        return this.jbNext;
    }

    //Getter method for jbLast
    public JButton getJBLast(){
        return this.jbLast;
    }

    //Getter method for jbAddressForm
    public JButton getJBAddressForm(){
        return this.jbAddressForm;
    }

    //Getter method for jbCountryForm
    public JButton getJBCountryForm(){
        return this.jbCountryForm;
    }

    //Getter method for jbCityForm
    public JButton getJBCityForm(){
        return this.jbCityForm;
    }

    //Getter method for jbStoreForm
    public JButton getJBStoreForm(){
        return this.jbStoreForm;
    }

    //Getter method for jbChoosePicture
    public JButton getJBChoosePicture(){
        return this.jbChoosePicture;
    }

    //Getter method for jPanel1
    public JPanel getJPanel1(){
```

```
        return this.jPanel1;
    }

    //Getter method for jPanel2
    public JPanel getJPanel2(){
        return this.jPanel2;
    }

    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">
    private void initComponents() {
        //...
        pack();
    }// </editor-fold>

    private void jtStaffMousePressed(java.awt.event.MouseEvent evt) {
        Staff_Utils.mouse_pressed_handler(this);
    }

    private void jbFirstActionPerformed(java.awt.event.ActionEvent evt) {
        Staff_Utils.show_first_row(this);
    }

    private void jbPrevActionPerformed(java.awt.event.ActionEvent evt) {
        Staff_Utils.show_prev_row(this);
    }

    private void jbNextActionPerformed(java.awt.event.ActionEvent evt) {
        Staff_Utils.show_next_row(this);
    }

    private void jbLastActionPerformed(java.awt.event.ActionEvent evt) {
        Staff_Utils.show_last_row(this);
    }

    private void jbChartActionPerformed(java.awt.event.ActionEvent evt) {
        Staff_Utils.jbchart_handler(this);
    }

    private void jbEditActionPerformed(java.awt.event.ActionEvent evt) {
        Staff_Utils.edit_handler(this);
    }

    private void jbInsertActionPerformed(java.awt.event.ActionEvent evt) {
        Staff_Utils.insert_handler(this);
    }

    private void jbDeleteActionPerformed(java.awt.event.ActionEvent evt) {
        Staff_Utils.delete_handler(this);
    }

    private void jbCountryFormActionPerformed(java.awt.event.ActionEvent evt)
```

```
        CountryForm ct_form = new CountryForm();
        ct_form.setVisible(true);
    }

    private void jbAddressFormActionPerformed(java.awt.event.ActionEvent evt)
        AddressForm add_form = new AddressForm();
        add_form.setVisible(true);
    }

    private void jbStoreFormActionPerformed(java.awt.event.ActionEvent evt) {
        StoreForm str_form = new StoreForm();
        str_form.setVisible(true);
    }

    private void jbCityFormActionPerformed(java.awt.event.ActionEvent evt) {
        CityForm cty_form = new CityForm();
        cty_form.setVisible(true);
    }

    private void jcbStaffIDActionPerformed(java.awt.event.ActionEvent evt) {
        Staff_Utils.jcbStaff_handler(this, this.jcbStaffID);
    }

    private void jcbAddressIDActionPerformed(java.awt.event.ActionEvent evt)
        Staff_Utils.jcbAddress_handler(this);
    }

    private void jcbStoreIDActionPerformed(java.awt.event.ActionEvent evt) {
        Staff_Utils.jcbStore_handler(this);
    }

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

    private void jcbStaffNameActionPerformed(java.awt.event.ActionEvent evt)
        Staff_Utils.jcbStaff_handler(this, this.jcbStaffName);
    }

    private void jtStaffMouseClicked(java.awt.event.MouseEvent evt) {
//        // instantiate CustomerTableModelListener and add it as a listener
//        CustomerTableModelListener tableModelListener = new
CustomerTableModelListener(this.getJTCustomer(), this);
//        this.getJTCustomer().getModel().addTableModelListener(tableModelLis
    }

    private void jbChoosePictureActionPerformed(java.awt.event.ActionEvent ev
        Staff_Utils.jbChoosePicture_handler(this);
    }

    public static void main(String args[]) {
        try {
```

```
            for (javax.swing.UIManager.LookAndFeelInfo info :
javax.swing.UIManager.getInstalledLookAndFeels()) {
                if ("Nimbus".equals(info.getName())) {
                    javax.swing.UIManager.setLookAndFeel(info.getClassName())
                    break;
                }
            }
        } catch (ClassNotFoundException ex) {

java.util.logging.Logger.getLogger(StaffForm.class.getName()).log(java.util.l
null, ex);
        } catch (InstantiationException ex) {

java.util.logging.Logger.getLogger(StaffForm.class.getName()).log(java.util.l
null, ex);
        } catch (IllegalAccessException ex) {

java.util.logging.Logger.getLogger(StaffForm.class.getName()).log(java.util.l
null, ex);
        } catch (javax.swing.UnsupportedLookAndFeelException ex) {

java.util.logging.Logger.getLogger(StaffForm.class.getName()).log(java.util.l
null, ex);
        }
        /* Create and display the form */
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new StaffForm().setVisible(true);
            }
        });
    }

    // Variables declaration - do not modify
    private javax.swing.JLabel jLabel10;
    private javax.swing.JLabel jLabel11;
    private javax.swing.JLabel jLabel12;
    private javax.swing.JLabel jLabel13;
    private javax.swing.JLabel jLabel14;
    private javax.swing.JLabel jLabel15;
    private javax.swing.JLabel jLabel16;
    private javax.swing.JLabel jLabel17;
    private javax.swing.JLabel jLabel18;
    private javax.swing.JLabel jLabel19;
    private javax.swing.JLabel jLabel20;
    private javax.swing.JLabel jLabel21;
    private javax.swing.JLabel jLabel22;
    private javax.swing.JLabel jLabel23;
    private javax.swing.JLabel jLabel24;
    private javax.swing.JLabel jLabel3;
    private javax.swing.JLabel jLabel4;
    private javax.swing.JLabel jLabel5;
    private javax.swing.JLabel jLabel6;
    private javax.swing.JLabel jLabel7;
```

```
    private javax.swing.JLabel jLabel8;
    private javax.swing.JLabel jLabel9;
    private javax.swing.JPanel jPanel1;
    private javax.swing.JPanel jPanel2;
    private javax.swing.JScrollPane jScrollPane1;
    private javax.swing.JButton jbAddressForm;
    private javax.swing.JButton jbChart;
    private javax.swing.JButton jbChoosePicture;
    private javax.swing.JButton jbCityForm;
    private javax.swing.JButton jbCountryForm;
    private javax.swing.JButton jbDelete;
    private javax.swing.JButton jbEdit;
    private javax.swing.JButton jbFirst;
    private javax.swing.JButton jbInsert;
    private javax.swing.JButton jbLast;
    private javax.swing.JButton jbNext;
    private javax.swing.JButton jbPrev;
    private javax.swing.JButton jbStoreForm;
    private javax.swing.JComboBox<String> jcbActive;
    private javax.swing.JComboBox<String> jcbAddressID;
    private javax.swing.JComboBox<String> jcbStaffID;
    private javax.swing.JComboBox<String> jcbStaffName;
    private javax.swing.JComboBox<String> jcbStoreID;
    private javax.swing.JLabel jlPicture;
    private javax.swing.JTable jtStaff;
    private javax.swing.JTextField jtfAddress;
    private javax.swing.JTextField jtfAddressID;
    private javax.swing.JTextField jtfCity;
    private javax.swing.JTextField jtfCountry;
    private javax.swing.JTextField jtfDistrict;
    private javax.swing.JTextField jtfEmail;
    private javax.swing.JTextField jtfFirstName;
    private javax.swing.JTextField jtfLastName;
    private javax.swing.JTextField jtfLastUpdate;
    private javax.swing.JTextField jtfPassword;
    private javax.swing.JTextField jtfPhone;
    private javax.swing.JTextField jtfPostalCode;
    private javax.swing.JTextField jtfStaffID;
    private javax.swing.JTextField jtfStoreAddress;
    private javax.swing.JTextField jtfStoreCity;
    private javax.swing.JTextField jtfStoreCountry;
    private javax.swing.JTextField jtfStoreDistrict;
    private javax.swing.JTextField jtfStoreID;
    private javax.swing.JTextField jtfStorePhone;
    private javax.swing.JTextField jtfUsername;
    // End of variables declaration
}
```

RENTAL FORM

RENTAL FORM

CREATING AND POPULATING RENTAL TABLE

CREATING AND POPULATING RENTAL TABLE

Step 1	Create a new class named **Query_Rental**. It contains multiple private static final String

variables that store SQL queries. These queries are used to retrieve information from the rental table in the Sakila database.

The first three queries **sql_min**, **sql_max**, and **sql_id** are simple queries to retrieve the minimum **rental_id**, maximum **rental_id**, and details of a specific **rental_id** respectively.

The next six queries are used to retrieve various statistics related to the films rented.

sql_rental_film_year_dist returns the distribution of the number of films rented per release year, where the films are joined with the inventory table and film table, and grouped by the year of release.

sql_rental_film_duration_dist returns the distribution of the number of films rented per rental duration (in days), where the films are joined with the inventory table and film table, and grouped by the rental duration.

sql_rental_film_rating_dist returns the distribution of the number of films rented per rating, where the films are joined with the inventory table and film table, and grouped by the rating.

sql_rental_staff_name_dist returns the distribution of the number of films rented per staff member, where the rentals are joined with the staff table, and grouped by staff member.

sql_rental_film_language_dist returns the distribution of the number of films rented per language, where the rentals are joined with the inventory table, film table, and language table, and grouped by language.

sql_rental_film_title_dist returns the distribution of the number of films rented per film title, where the rentals are joined with the inventory table and film table, and grouped by film title.

The final six queries are used to retrieve various statistics related to the customers who rented the films.

sql_rental_customer_active_dist returns the distribution of the number of films rented per customer activity status, where the rentals are joined with the customer table, and grouped by customer activity status.

sql_rental_customer_district_dist returns the distribution of the number of films rented per customer district, where the rentals are joined with the customer table, address table, city table, and country table, and grouped by district.

sql_rental_customer_country_dist returns the distribution of the number of films rented per customer country, where the rentals are joined with the customer table, address table, city table, and country table, and grouped by country.

sql_rental_customer_city_dist returns the distribution of the number of films rented per customer city, where the rentals are joined with the customer table, address table, city table, and country table, and grouped by city.

sql_rental_customer_name_dist returns the distribution of the number of films rented per customer name, where the rentals are joined with the customer table, address table, city table, and country table, and grouped by customer name.

```
1       package sakila;
2
3       public class Query_Rental {
4           private static final String sql_min
5       = "SELECT MIN(rental_id) FROM rental";
6           private static final String sql_max
7       = "SELECT MAX(rental_id) FROM rental";
8           private static final String sql_id
9       = "SELECT * FROM rental WHERE rental_id
10      = ?";
11
12          private static final String
13      sql_rental_quarter = """
14              SELECT
15                  CASE
```

```
                WHEN MONTH(rental_date)
BETWEEN 1 AND 3 THEN 'January to March'
                WHEN MONTH(rental_date)
BETWEEN 4 AND 6 THEN 'April to June'
                WHEN MONTH(rental_date)
BETWEEN 7 AND 9 THEN 'July to
September'
                ELSE 'October to
December'
            END AS quarter,
        COUNT(*) AS number_of_films
        FROM rental
        GROUP BY quarter
        ORDER BY Count(*) DESC,
number_of_films DESC;""";

    private static final String
sql_rental_day = """
        SELECT DAYNAME(rental_date) AS
day, COUNT(*) AS number_of_films
        FROM rental
        GROUP BY day
        ORDER BY Count(*) DESC,
number_of_films DESC;""";

    private static final String
sql_rental_week = """
        SELECT WEEK(rental_date) AS
week, COUNT(*) AS number_of_films
        FROM rental
        GROUP BY week
        ORDER BY Count(*) DESC,
number_of_films DESC
        LIMIT 10;""";

    private static final String
sql_rental_month = """
        SELECT MONTHNAME(rental_date)
AS month, COUNT(*) AS number_of_films
        FROM rental
        GROUP BY month
        ORDER BY Count(*) DESC,
number_of_films DESC;""";

    private static final String
sql_rental_year = """
        SELECT YEAR(rental_date) AS
year, COUNT(*) AS number_of_films
        FROM rental
        GROUP BY year
        ORDER BY Count(*) DESC,
number_of_films DESC;""";


```

```
    private static final String
sql_rental_film_year_dist = """
        SELECT YEAR(f.release_year) AS
film_year, COUNT(*) AS number_of_films
        FROM rental r
        JOIN inventory i ON
r.inventory_id = i.inventory_id
        JOIN film f ON i.film_id =
f.film_id
        GROUP BY film_year
        ORDER BY Count(*) DESC,
number_of_films DESC;""";

    private static final String
sql_rental_film_duration_dist = """
        SELECT f.rental_duration AS
film_duration, COUNT(*) AS
number_of_films
        FROM rental r
        JOIN inventory i ON
r.inventory_id = i.inventory_id
        JOIN film f ON i.film_id =
f.film_id
        GROUP BY film_duration
        ORDER BY Count(*) DESC,
number_of_films DESC;""";

    private static final String
sql_rental_film_rating_dist = """
        SELECT f.rating AS film_rating,
COUNT(*) AS number_of_films
        FROM rental r
        JOIN inventory i ON
r.inventory_id = i.inventory_id
        JOIN film f ON i.film_id =
f.film_id
        GROUP BY film_rating
        ORDER BY number_of_films
DESC;""";

    private static final String
sql_rental_staff_name_dist = """
        SELECT CONCAT(st.first_name,'
',st.last_name) AS staff_name, COUNT(*)
AS number_of_films
        FROM rental r
        JOIN staff st ON st.staff_id =
r.staff_id
        GROUP BY staff_name
        ORDER BY number_of_films DESC
        LIMIT 10;""";

    private static final String
sql_rental_film_language_dist = """
```

```
124         SELECT l.name AS film_language,
125     COUNT(*) AS number_of_films
126127      FROM rental r
128         JOIN inventory i ON
129     r.inventory_id = i.inventory_id
130         JOIN film f ON i.film_id =
131     f.film_id
132         JOIN language l ON
133     l.language_id = f.language_id
134         GROUP BY film_language
135         ORDER BY number_of_films DESC
136         LIMIT 10;""";
137
138         private static final String
139     sql_rental_film_title_dist = """
140         SELECT f.title AS film_title,
141     COUNT(*) AS number_of_films
142         FROM rental r
143         JOIN inventory i ON
144     r.inventory_id = i.inventory_id
145         JOIN film f ON i.film_id =
146     f.film_id
147         GROUP BY film_title
148         ORDER BY number_of_films DESC
149         LIMIT 10;""";
150
151         private static final String
152     sql_rental_customer_active_dist = """
153         SELECT cu.active AS
154     customer_active, COUNT(*) AS
155     number_of_films
156         FROM rental r
157         JOIN customer cu ON
158     cu.customer_id = r.customer_id
159         GROUP BY customer_active
160         ORDER BY number_of_films DESC
161         LIMIT 10;""";
162
163         private static final String
164     sql_rental_customer_district_dist = """
165         SELECT IFNULL(ad.district,
166     'UNKNOWN') AS customer_district,
167     COUNT(*) AS number_of_films
168         FROM rental r
169         JOIN customer cu ON
170     cu.customer_id = r.customer_id
171         JOIN address ad ON
172     ad.address_id = cu.address_id
173         JOIN city ci ON ci.city_id =
174     ad.city_id
175         JOIN country co ON
176     co.country_id = ci.country_id
177         GROUP BY customer_district
178
```

```
179         ORDER BY number_of_films DESC
180         LIMIT 10;""";
181 
182     private static final String
183 sql_rental_customer_country_dist = """
184         SELECT co.country AS
185 customer_country, COUNT(*) AS
186 number_of_films
187         FROM rental r
188         JOIN customer cu ON
189 cu.customer_id = r.customer_id
190         JOIN address ad ON
191 ad.address_id = cu.address_id
192         JOIN city ci ON ci.city_id =
193 ad.city_id
194         JOIN country co ON
195 co.country_id = ci.country_id
196         GROUP BY customer_country
197         ORDER BY number_of_films DESC
198         LIMIT 10;""";
199 
200     private static final String
201 sql_rental_customer_city_dist = """
202         SELECT ci.city AS
203 customer_city, COUNT(*) AS
204 number_of_films
205         FROM rental r
206         JOIN customer cu ON
207 cu.customer_id = r.customer_id
208         JOIN address ad ON
209 ad.address_id = cu.address_id
210         JOIN city ci ON ci.city_id =
211 ad.city_id
212         JOIN country co ON
213 co.country_id = ci.country_id
214         GROUP BY customer_city
215         ORDER BY number_of_films DESC
216         LIMIT 10;""";
217 
218     private static final String
219 sql_rental_customer_name_dist = """
220         SELECT CONCAT(cu.first_name,'
221 ',cu.last_name) AS customer_name,
222 COUNT(*) AS number_of_films
223         FROM rental r
224         JOIN customer cu ON
225 cu.customer_id = r.customer_id
226         JOIN address ad ON
227 ad.address_id = cu.address_id
228         JOIN city ci ON ci.city_id =
229 ad.city_id
230         JOIN country co ON
231 co.country_id = ci.country_id
232         GROUP BY customer_name
```

```
233         ORDER BY number_of_films DESC
234         LIMIT 10;""";
235 
236     private static final String
237 sql_rental_actor_dist = """
238         SELECT CONCAT(a.first_name,'
239 ',a.last_name) AS actor_name, COUNT(*)
240 AS number_of_films
241         FROM rental r
242         JOIN inventory i ON
243 r.inventory_id = i.inventory_id
244         JOIN film f ON i.film_id =
245 f.film_id
246247        JOIN language l ON
248 l.language_id = f.language_id
249         JOIN film_actor fa ON f.film_id
250 = fa.actor_id
251         JOIN actor a ON fa.actor_id =
252 a.actor_id
253         GROUP BY actor_name
254         ORDER BY number_of_films DESC
255         LIMIT 10;""";
256 
257     private static final String
258 sql_rental_category_dist = """
259         SELECT c.name AS film_category,
260 COUNT(*) AS number_of_films
261         FROM rental r
262         JOIN inventory i ON
263 r.inventory_id = i.inventory_id
264         JOIN film f ON i.film_id =
265 f.film_id
266         JOIN language l ON
267 l.language_id = f.language_id
268         JOIN film_category fc ON
269 f.film_id = fc.film_id
270         JOIN category c ON
271 fc.category_id = c.category_id
272         GROUP BY film_category
273         ORDER BY number_of_films DESC
274         LIMIT 10;""";
275 
276     private static final String
277 sql_inventory_joint = """
278         SELECT i.inventory_id,
279 f.film_id, f.title, f.description,
280 f.release_year,
281             f.rental_duration,
282 f.rental_rate, f.length,
283 f.replacement_cost, f.rating,
284             f.special_features, l.name
285 AS language_name, c.name AS
286 category_name,
287 
```

```
            CONCAT(a.first_name,'
',a.last_name) AS actor_name,
so.store_id,
            ci.city AS store_city,
co.country AS store_country,
ad.address_id,
            ad.address AS
store_address,
            ad.phone AS store_phone,
ad.district AS store_district,
            ad.postal_code AS
store_postal_code
        FROM inventory i
        JOIN store so ON so.store_id =
i.store_id
        JOIN address ad ON
ad.address_id = so.address_id
        JOIN city ci ON ci.city_id =
ad.city_id
        JOIN country co ON
co.country_id = ci.country_id
        JOIN film f ON i.film_id =
f.film_id
        JOIN language l ON
l.language_id = f.language_id
        JOIN film_category fc ON
f.film_id = fc.film_id
        JOIN category c ON
fc.category_id = c.category_id
        JOIN film_actor fa ON f.film_id
= fa.film_id
        JOIN actor a ON fa.actor_id =
a.actor_id""";

    private static final String
sql_rental_joint = """
        SELECT r.rental_id,
r.rental_date, r.inventory_id,
r.customer_id,
            r.return_date, r.staff_id,
r.last_update,
            f.film_id, f.title,
f.description, f.release_year,
            f.rental_duration,
f.rental_rate, f.length,
f.replacement_cost,
            f.rating,
f.special_features, l.name AS
language_name,
            c.name AS category_name,
            CONCAT(a.first_name,'
',a.last_name) AS actor_name,
            CONCAT(cu.first_name,'
',cu.last_name) AS customer_name,
```

```
342             cu.email AS customer_email,
343             CONCAT(st.first_name,'
342 ',st.last_name) AS staff_name,
343             st.email AS staff_email,
344 ci.city AS store_city,
            co.country AS
store_country, ad.address_id,
            ad.address AS
store_address,
            ad.phone AS store_phone,
ad.district AS store_district,
            ad.postal_code AS
store_postal_code, so.store_id
        FROM rental r
        JOIN inventory i ON
r.inventory_id = i.inventory_id
        JOIN store so ON so.store_id =
i.store_id
        JOIN address ad ON
ad.address_id = so.address_id
        JOIN city ci ON ci.city_id =
ad.city_id
        JOIN country co ON
co.country_id = ci.country_id
        JOIN staff st ON st.staff_id =
r.staff_id
        JOIN customer cu ON
cu.customer_id = r.customer_id
        JOIN film f ON i.film_id =
f.film_id
        JOIN language l ON
l.language_id = f.language_id
        JOIN film_category fc ON
f.film_id = fc.film_id
        JOIN category c ON
fc.category_id = c.category_id
        JOIN film_actor fa ON f.film_id
= fa.film_id
        JOIN actor a ON fa.actor_id =
a.actor_id""";

    private static final String
sql_rental = """
        CREATE TABLE rental (
          rental_id INT NOT NULL
AUTO_INCREMENT,
          rental_date DATETIME NOT
NULL,
          inventory_id MEDIUMINT
UNSIGNED NOT NULL,
          customer_id SMALLINT UNSIGNED
NOT NULL,
          return_date DATETIME DEFAULT
NULL,
```

```
            staff_id TINYINT UNSIGNED NOT
NULL,
            last_update TIMESTAMP NOT
NULL DEFAULT CURRENT_TIMESTAMP ON
UPDATE CURRENT_TIMESTAMP,
            PRIMARY KEY (rental_id),
            UNIQUE KEY
(rental_date,inventory_id,customer_id),
            KEY idx_fk_inventory_id
(inventory_id),
            KEY idx_fk_customer_id
(customer_id),
            KEY idx_fk_staff_id
(staff_id),
            CONSTRAINT fk_rental_staff
FOREIGN KEY (staff_id) REFERENCES staff
(staff_id) ON DELETE RESTRICT ON UPDATE
CASCADE,
            CONSTRAINT
fk_rental_inventory FOREIGN KEY
(inventory_id) REFERENCES inventory
(inventory_id) ON DELETE RESTRICT ON
UPDATE CASCADE,
            CONSTRAINT fk_rental_customer
FOREIGN KEY (customer_id) REFERENCES
customer (customer_id) ON DELETE
RESTRICT ON UPDATE CASCADE
          ) ENGINE=InnoDB DEFAULT
CHARSET=utf8mb4;""";

    //Getter methods
    public static String get_sql_min()
{
        return sql_min;
    }

    public static String get_sql_max()
{
        return sql_max;
    }

    public static String get_sql_id() {
        return sql_id;
    }

    public static String
get_sql_rental() {
        return sql_rental;
    }

    public static String
get_sql_rental_joint() {
        return sql_rental_joint;
```

```
    }

    public static String
get_sql_inventory_joint() {
        return sql_inventory_joint;
    }

    public static String
get_sql_rental_category_dist() {
        return
sql_rental_category_dist;
    }

    public static String
get_sql_rental_actor_dist() {
        return sql_rental_actor_dist;
    }

    public static String
get_sql_rental_customer_name_dist() {
        return
sql_rental_customer_name_dist;
    }

    public static String
get_sql_rental_customer_city_dist() {
        return
sql_rental_customer_city_dist;
    }

    public static String
get_sql_rental_customer_country_dist()
{
        return
sql_rental_customer_country_dist;
    }

    public static String
get_sql_rental_customer_district_dist()
{
        return
sql_rental_customer_district_dist;
    }

    public static String
get_sql_rental_customer_active_dist() {
        return
sql_rental_customer_active_dist;
    }

    public static String
get_sql_rental_film_title_dist() {
```

```
        return
sql_rental_film_title_dist;
    }

    public static String
get_sql_rental_film_language_dist() {
        return
sql_rental_film_language_dist;
    }

    public static String
get_sql_rental_staff_name_dist() {
        return
sql_rental_staff_name_dist;
    }

    public static String
get_sql_rental_film_rating_dist() {
        return
sql_rental_film_rating_dist;
    }
```

```
    public static String
get_sql_rental_film_year_dist() {
        return
sql_rental_film_year_dist;
    }

    public static String
get_sql_rental_quarter() {
        return sql_rental_quarter;
    }

    public static String
get_sql_rental_day() {
        return sql_rental_day;
    }

    public static String
get_sql_rental_week() {
        return sql_rental_week;
    }

    public static String
get_sql_rental_month() {
        return sql_rental_month;
    }

    public static String
get_sql_rental_year() {
        return sql_rental_year;
    }
}
```

Step 2

Then, create a public class nam
instance variables and various cons
get and set the values of these varia

The class represents a rental record
containing information such as
inventory ID, customer ID, return
title, description, release year, rent
length, replacement cost, rating, sp
name, category, actor name, cus
email, staff name, staff email, store
district, and store postal code.

The class has three constructor
constructor with predefined va
constructor, and a 28-parameter co

various getter and setter methods t
values of the instance variables.

Overall, the class **Rental** is a rep
record in a movie rental store, and
way to access and modify the vario
associated with a rental record.

```
1   package sakila;
2   import java.sql.Timestamp;
3   import java.sql.Date;
4   import java.util.Calendar;
5
6   public class Rental {
7       //28 instance variables
8       private int rental_id;
9       private Date rental_dat
10      private int inv_id;
11      private int cust_id;
12      private Date return_dat
13      private int staff_id;
14      private Timestamp last_
15
16      private int film_id;
17      private String title;
18      private String descript
19      private int release_yea
20      private int rental_dura
21
22      private double rental_r
23      private int length;
24      private double replacem
25      private String rating;
26      private String special_
27
28      private String language
29      private String category
30      private String actor_na
31      private String customer
32      private String customer
33
34      private String staff_na
35      private String staff_em
36      private String store_ci
37      private String store_co
38      private String store_di
39
40      private String store_po
41
42      //Default constructor
43      Rental(){
44
```

```
45          this(1, new
46      Date(Calendar.getInstance()
47                  1, 1, new
48      Date(Calendar.getInstance()
49      1,
50                  new
51      Timestamp(System.currentTim
52      }
53
54          //Six-params constructo
55          Rental(Date rent_date,
56      cust_id, Date ret_date, int
57      lu){
58          setRentalDate(rent_
59          setInventoryID(inv_
60          setCustomerID(cust_
61          setReturnDate(ret_d
62          setStaffID(staff_id
63          setLastUpdate(lu);
64      }
65
66          //Seven-params construc
67          Rental(int rental_id, D
68      inv_id, int cust_id, Date r
69      Timestamp lu){
70          this(rent_date, inv
71      ret_date, staff_id, lu);
72          setRentalID(rental_
73      }
74
75          //28-params constructor
76          Rental(int rental_id, D
77      inv_id, int cust_id, Date r
78      Timestamp lu, int film_id,
79      description, int year, int
80      int length, double cost, St
81      features, String lang_name,
82      String actor_name, String c
83      cust_email, String staff_na
84      staff_email, String store_c
85      store_country, String store
86      store_post){
87          this(rental_id, ren
88      cust_id, ret_date, staff_id
89          this.film_id = film
90          this.title = title;
91          this.description =
92          this.release_year =
93          this.rental_duratio
94          this.rental_rate =
95          this.length = lengt
96          this.replacement_co
97          this.rating = ratin
98          this.special_featur
```

```
99          this.language_name
100         this.category = cat
101         this.actor_name = a
102         this.customer_name
103         this.customer_email
104         this.staff_name = s
105         this.staff_email =
106         this.store_city = s
107         this.store_country
108         this.store_district
109         this.store_postal_c
110     }
111
112     //Getter methods
113     public int getRentalID(
114     public Date getRentalDa
115 this.rental_date;}
116     public int getInventory
117     public int getCustomerI
118     public Date getReturnDa
119 this.return_date;}
120     public int getStaffID()
121     public Timestamp getLas
122 last_update;}
123     public int getFilmID()
124     public String getTitle(
125     public String getDescri
126127 description;}
128     public int getReleaseYe
129 release_year;}
130     public int getRentalDur
131 rental_duration;}
132     public double getRental
133 rental_rate;}
134     public int getLength()
135     public double getReplac
136 replacement_cost;}
137     public String getRating
138     public String getSpecia
139 special_features;}
140     public String getLangua
141 language_name;}
142
143     public String getCatego
144 category;}
145     public String getActorN
146 actor_name;}
147     public String getCustom
148 customer_name;}
149     public String getCustom
150 staff_name;}
151     public String getStaffN
152 actor_name;}
153
```

```
154     public String getStaffE
155 staff_email;}
156
157     public String getStore(
158 store_city;}
159     public String getStore(
160 store_country;}
161     public String getStoreD
162 store_district;}
163     public String getStoreP
164 store_postal_code;}
165
166     //Setter methods
167     public void setRentalID
168         if (id <= 0) {
169             throw new
170 IllegalArgumentException("P
171 greater than zero.");
172         }
173         this.rental_id = id
174     }
175
176     public void setRentalDa
177         this.rental_date =
178     }
179
180     public void setInventor
181         if (id <= 0) {
182             throw new Illegal
183 ID must be greater than zero.");
184         }
185         this.inv_id = id;
    }

    public void setCustomer
        if (id <= 0) {
            throw new Illegal
must be greater than zero.");
        }
        this.cust_id = id;
    }

    public void setReturnDa
        this.return_date =
    }

    public void setStaffID(
        if (id <= 0) {
            throw new Illegal
must be greater than zero.");
        }
        this.staff_id = id;
    }
```

```
    public void setLastUpda
        if (date == null) {
        throw new IllegalAr
cannot be null");
        }
        this.last_update =
    }

    @Override
    public String toString(
        return "\nRental ID
getRentalID() +
                "\nRental Da
getRentalDate() +
                "\nInventory
getInventoryID() +
                "\nCustomer
getCustomerID() +
                "\nReturn Da
getReturnDate() +
                "\nStaff ID
getStaffID() +
                "\nLast Upda
getLastUpdate();
    }
}
```

Step 3

Create a new public class named **R** various methods for managing renta imports several packages includi java.util.logging.Level, java.util.log etc.

The class defines several constants FIRST_INDEX, INVALID_INDEX, cu

There are three public methods defi

1. **create_rental_table()** - Thi table in the database for st using a SQL statement defin class. If the table is success dialog is displayed with a exception occurs, an err displayed with the exception
2. **populate_rental_table()** - rows of data to the **renta** using prepared statements.

PreparedStatement objec
constructor and anothe
constructor, then sets va
parameters using two diffe
the data is successfully a
message dialog is displayed
If an exception occurs, an
displayed with the exception

3. **read_rental_table()** - Th
contents of the rental table
ResultSet object and a SQ
retrieves each row of data
creates a new **Rental** objec
constructor to store the dat
then printed to the consol
method. If an exception oc
dialog is displayed with the e

```
1  package sakila;
2  import java.awt.Dimension;
3  import java.util.logging.Level
4  import java.util.logging.Logge
5  import java.sql.*;
6  import java.text.ParseExceptio
7  import java.text.SimpleDateFor
8  import java.util.ArrayList;
9  import java.util.Arrays;
10 import java.util.Calendar;
11 import java.util.HashMap;
12 import java.util.Objects;
13 import javax.swing.JComboBox;
14 import javax.swing.JOptionPane
15 import javax.swing.JPanel;
16 import javax.swing.event.Table
17 import javax.swing.table.Defau
18 import javax.swing.table.Table
19 import
20 org.jfree.data.category.Defaul
21 import org.jfree.data.general.
22
23 public class Rental_Utils exte
24     public static final int FI
25     public static final int IN
26
27     private static int current
28 FIRST_INDEX;
29     private static final Strin
30 Query_Rental.get_sql_id();
31
32     //Creates rental table
```

```
    public static void create_
        try (Connection conn =
            Statement stmt =
conn.createStatement();

stmt.addBatch(Query_Rental.get
            stmt.executeBatch(

            String message =
String.format("Successfully cr
table");
            JOptionPane.showMe
message,

 "INFORMATION",JOptionPane.INF

        } catch (SQLException
            JOptionPane.showMe
ex.getMessage(),

 "ERROR",JOptionPane.ERROR_MES
        }
    }

    //Populates rental table w
data
    public static void populat
        try(Connection conn =
            String sql = """
                INSERT INTO re
rental_date, inventory_id, cus
return_date, staff_id, last_up
                VALUES(?, ?, ?

            //Creates a new Re
default constructor
            PreparedStatement
conn.prepareStatement(sql);
            Rental obj1 = new
            ps1.setInt(1,obj1.
            ps1.setDate(2,obj1
            ps1.setInt(3,obj1.
            ps1.setInt(4,obj1.
            ps1.setDate(5,obj1
            ps1.setInt(6,obj1.

ps1.setTimestamp(7,obj1.getLas

            // Creates a new P
nine-params constructor
            PreparedStatement
conn.prepareStatement(sql);
```

```
87              SimpleDateFormat d
88  SimpleDateFormat("yyyy-MM-dd")
89              String str_rental_
90  03";
91              java.util.Date ren
92  dateFormat.parse(str_rental_da
93              java.sql.Date rent
94  java.sql.Date(rent_date.getTim
95
96              String str_ret_dat
97              java.util.Date ret
98  dateFormat.parse(str_ret_date)
99              java.sql.Date retu
100 java.sql.Date(ret_date.getTime
101
102             Rental obj2 = new
103 rental_sql_date, 2, 2, return_
104 Timestamp(System.currentTimeMi
105             ps2.setInt(1,obj2.
106             ps2.setDate(2,obj2
107             ps2.setInt(3,obj2.
108             ps2.setInt(4,obj2.
109             ps2.setDate(5,obj2
110             ps2.setInt(6,obj2.
111
112 ps2.setTimestamp(7,obj2.getLas
113
114             ps1.executeUpdate(
115             ps2.executeUpdate(
116
117         }catch(SQLException ex
118             JOptionPane.showMe
119 ex.getMessage(),
120
121  "ERROR",JOptionPane.ERROR_MES
122         } catch (ParseExceptio
123
124 Logger.getLogger(Rental_Utils.class.g
125 null, ex);
126         }
    }

    //Reads the content of ren
    public static void read_re
        try(Connection conn =
            Statement stmt =
conn.createStatement();
            ResultSet rs =
stmt.executeQuery("SELECT * FR

            while(rs.next()){
                int rental_id
rs.getInt("rental_id");
```

```
                Date rent_date
rs.getDate("rental_date");
                int inv_id =
rs.getInt("inventory_id");
                int cust_id =
rs.getInt("customer_id");
                Date ret_date
rs.getDate("return_date");
                int staff_id =
rs.getInt("staff_id");
                Timestamp lu =
rs.getTimestamp("last_update")

                //Creates a Re
seven-params constructor
                Rental obj = n
Rental(rental_id, rent_date, i
ret_date, staff_id, lu);
                System.out.pri
            }
            rs.close();
            stmt.close();

        }catch(SQLException ex
            JOptionPane.showMe
ex.getMessage(),

 "ERROR",JOptionPane.ERROR_MES
        }
    }
}
```

Step 4

In the driver class, **create_rental_table()**, **populate** **read_rental_table()** as shown in lin

```
1  package sakila;
2
3  public class Sakila {
4      public static void main(Str
5  args) {
6          //...
7
8  //
9  Customer_Utils.create_customer_
10 //
11 Customer_Utils.populate_custome
12 //
13 Customer_Utils.read_customer_ta
14
15
```

```
//        CustomerForm frm = ne
CustomerForm();
//        frm.setVisible(true);

//
Staff_Utils.create_staff_table(
//
Staff_Utils.populate_staff_tabl
//        Staff_Utils.read_staf
//        StaffForm frm = new
StaffForm();
//        frm.setVisible(true);


Rental_Utils.create_rental_tabl

Rental_Utils.populate_rental_ta
        Rental_Utils.read_renta
    }
}
```

Run project to see the result in cons

```
Rental ID             : 1
Rental Date           : 2023-05-03
Inventory ID          : 1
Customer ID           : 1
Return Date           : 2023-05-03
Staff ID              : 1
Last Update           : 2023-05-03

Rental ID             : 2
Rental Date           : 2023-05-03
Inventory ID          : 2
Customer ID           : 2
Return Date           : 2023-05-19
Staff ID              : 2
Last Update           : 2023-05-03
```

DESIGNING GUI
DESIGNING GUI

Step 1	In the project, create a new **JFrame Form** and name it as **RentalForm.java**. In the Design tab, add thirty-two **JLabels** to the form and set their corresponding **text** properties as RENTAL ID, RENTAL DATE,

	RETURN DATE, INVENTORY ID, FILM ID, STORE ID, ADDRESS ID, ADDRESS, DISTRICT, CITY, COUNTRY, PHONE, POSTAL CODE, CUSTOMER ID, CUST. NAME, CUST.EMAIL, LAST UPDATE, STAFF ID, STAFF NAME, STAFF EMAIL, TITLE, DESCRIPTION, RELEASE YEAR, LENGTH, RATING, SPECIAL FEATURES, ACTOR NAME, RENTAL DURATION, RENTAL RATE, REPLACEMENT COST, and CATEGORY.
Step 2	Then, add thirty **JTextField** to the form and set their corresponding **Variable Name** jtfRentalID, jtfInventoryID, jtfFilmID, jtfStoreID, jtfAddressID, jtfAddress, jtfDistrict, jtfCity, jtfCountry, jtfPhone, jtfPostalCode, jtfCustomerID, jtfCustomerName, jtfCustomerEmail, jtfLastUpdate, jtfStaffID, jtfStaffName, jtfStaffEmail, jtfTitle, jtfDescription, jtfReleaseYear, jtfLength, jtfRating, jtfSpecialFeatures, jtfActorName, jtfRentalDuration, jtfRentalRate, jtfReplacementCost, and jtfCategory.
Step 3	Then, add sixteen **JButton** to the form and set their corresponding **Variable Name** as **jbFirst**, **jbPrev**, **jbNext**, **jbLast**, **jbEdit**, **jbInsert**, **jbDelete**, **jbAddress**, **jbCountryForm**, **jbCityForm**, **jbAddressForm**, **jbStoreForm**, **jbFilmForm**, **jbActorForm**, **jbCategoryForm**, and **jbChart**. Set their corresponding **text** properties as FIRST, PREV, NEXT, LAST, EDIT, INSERT, DELETE, CHART, ADDRESS FORM, COUNTRY FORM, CITY FORM, STORE FORM, FILM FORM, CATEGORY FORM, ACTOR FORM, and CHART.
Step 4	Then, add four **JComboBoxes** to the form and set their corresponding **Variable Name** as **jcbRentalID**, **jcbInventoryID**, **jcbStaffID**, and **jcbCustomerID**.

Step 5	Add two **JCalendars** and set their corresponding **Variable Name** as **jcRentalDate** and **jcReturnDate**.
Step 6	Lastly, add a new **JTable** to the form set set its **Variable Name** as **jtRental**. Then, right-click on it, then choose **Table Contents...** and set the number of columns to 28 and the number of rows to 50.

Step 7	In the driver class, **Sakila.java**, create a new object of **RentalForm** class using its default constructor as shown in 16 – 17:
	
	1 package sakila;
	2
	3 public class Sakila {
	4 public static void main(String[] args) {
	5 //...
	6
	7 //
	8 Staff_Utils.create_staff_table();
	9 //
	10 Staff_Utils.populate_staff_table();
	11 //
	12 Staff_Utils.read_staff_table();
	13 // StaffForm frm = new StaffForm();
	14 // frm.setVisible(true);
	15
	16 //
	17 Rental_Utils.create_rental_table();
	18 //
	19 Rental_Utils.populate_rental_table();
	//
	Rental_Utils.read_rental_table();
	RentalForm frm = new RentalForm();
	frm.setVisible(true);
	}
	}

Step 8	In **RentalForm**'s constructor, invoke **setLookAndFeel()** to set the look and feel of the form as shown in line 17.

```
1  package sakila;
2
3  import java.awt.Toolkit;
4  import java.awt.event.ActionEvent;
5  import
6  java.awt.event.ActionListener;
7  import javax.swing.JButton;
8  import javax.swing.JComboBox;
9  import javax.swing.JMenuItem;
10 import javax.swing.JOptionPane;
11 import javax.swing.JPopupMenu;
12 import javax.swing.JTable;
13 import javax.swing.JTextField;
14
15 public class RentalForm extends
16 javax.swing.JFrame {
17     public RentalForm() {
18         initComponents();
19
20 Utility.setLookAndFeel(this);
       }
       //...
   }
```

Run the project to see the rental form as shown in Figure 15.1.

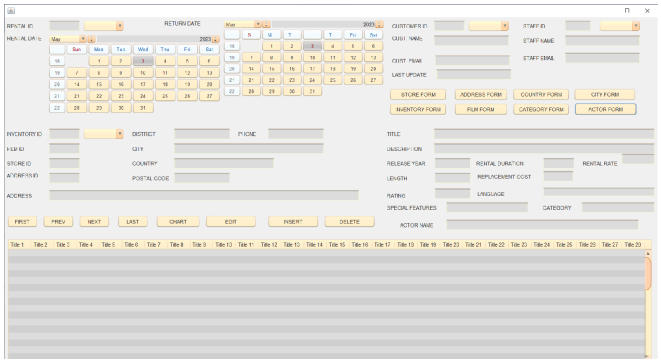


Figure 15.1 The layout of rental form

Step 9

In **RentalForm.java**, define **getter()** method for every object in the form and place them outside and beneath its default constructor:

```
1        //Getter method for
2  jtfRentalID
3      public JTextField
4  getJTFRentalID(){
5          return this.jtfRentalID;
```

```
6       }
7
8       //Getter method for
9   jcbRentalID
10      public JComboBox
11  getJCBRentalID(){
12          return this.jcbRentalID;
13      }
14
15      //Getter method for
16  jcRentalDate
17      public JCalendar
18  getJCRentalDate(){
19          return
20  this.jcRentalDate;
21      }
22
23      //Getter method for
24  jcReturnDate
25      public JCalendar
26  getJCReturnDate(){
27          return
28  this.jcReturnDate;
29      }
30
31      //Getter method for
32  jtfLastUpdate
33      public JTextField
34  getJTFLastUpdate(){
35          return
36  this.jtfLastUpdate;
37      }
38
39      //Getter method for
40  jtfInventoryID
41      public JTextField
42  getJTFInventoryID(){
43          return
44  this.jtfInventoryID;
45      }
46
47      //Getter method for
48  jcbInventoryID
49      public JComboBox
50  getJCBInventoryID(){
51          return
52  this.jcbInventoryID;
53      }
54
55      //Getter method for
56  jtfFilmID
57      public JTextField
58  getJTFFilmID(){
59
```

```
60          return this.jtfFilmID;
61      }
62
63      //Getter method for
64  jtfStoreID
65      public JTextField
66  getJTFStoreID(){
67          return this.jtfStoreID;
68      }
69
70      //Getter method for
71  jtfAddressID
72      public JTextField
73  getJTFAddressID(){
74          return
75  this.jtfAddressID;
76      }
77
78      //Getter method for
79  jtfAddress
80      public JTextField
81  getJTFAddress(){
82          return this.jtfAddress;
83      }
84
85      //Getter method for
86  jtfDistrict
87      public JTextField
88  getJTFDistrict(){
89          return this.jtfDistrict;
90      }
91
92      //Getter method for jtfPhone
93      public JTextField
94  getJTFPhone(){
95          return this.jtfPhone;
96      }
97
98      //Getter method for jtfCity
99      public JTextField
100 getJTFCity(){
101         return this.jtfCity;
102     }
103
104     //Getter method for
105 jtfCountry
106     public JTextField
107 getJTFCountry(){
108         return this.jtfCountry;
109     }
110
111     //Getter method for
112 jtfPostalCode
113
```

```
114     public JTextField
115 getJTFPostalCode(){
116         return
117 this.jtfPostalCode;
118     }
119 
120     //Getter method for jtfTitle
121     public JTextField
122 getJTFTitle(){
123         return this.jtfTitle;
124     }
125 
126     //Getter method for
127 jtfDescription
128     public JTextField
129 getJTFDescription(){
130         return
131 this.jtfDescription;
132     }
133 
134     //Getter method for
135 jtfReleaseYear
136     public JTextField
137 getJTFReleaseYear(){
138         return
139 this.jtfReleaseYear;
140     }
141 
142     //Getter method for
143 jtfRentalDuration
144     public JTextField
145 getJTFRentalDuration(){
146         return
147 this.jtfRentalDuration;
148     }
149 
150     //Getter method for
151 jtfRentalRate
152     public JTextField
153 getJTFRentalRate(){
154         return
155 this.jtfRentalRate;
156     }
157 
158     //Getter method for
159 jtfLength
160     public JTextField
161 getJTFLength(){
162         return this.jtfLength;
163     }
164 
165     //Getter method for
166 jtfReplacementCost
167 
```

```
168     public JTextField
169 getJTFReplacementCost(){
170         return
171 this.jtfReplacementCost;
172     }
173 
174     //Getter method for
175 jtfSpecialFeatures
176     public JTextField
177 getJTFSpecialFeatures(){
178         return
179 this.jtfSpecialFeatures;
180     }
181 
182     //Getter method for
183 jtfRating
184     public JTextField
185 getJTFRating(){
186         return this.jtfRating;
187     }
188 
189     //Getter method for
190 jtfLanguageID
191     public JTextField
192 getJTFLanguageID(){
193         return
194 this.jtfLanguageName;
195     }
196 
197198     //Getter method for
199 jtfCustomerID
200     public JTextField
201 getJTFCustomerID(){
202         return
203 this.jtfCustomerID;
204     }
205 
206     //Getter method for
207 jcbCustomerID
208     public JComboBox
209 getJCBCustomerID(){
210         return
211 this.jcbCustomerID;
212     }
213 
214     //Getter method for
215 jtfCustFname
216     public JTextField
217 getJTFCustomerName(){
218         return
219 this.jtfCustomerName;
220     }
221 
222 
```

```
223     //Getter method for
224 jtfCustEmail
225     public JTextField
226 getJTFCustEmail(){
227         return
228 this.jtfCustEmail;
229     }

    //Getter method for
jtfStaffID
    public JTextField
getJTFStaffID(){
        return this.jtfStaffID;
    }

    //Getter method for
jcbStaffID
    public JComboBox
getJCBStaffID(){
        return this.jcbStaffID;
    }

    //Getter method for
jtfStaffName
    public JTextField
getJTFStaffName(){
        return
this.jtfStaffName;
    }

    //Getter method for
jtfStaffEmail
    public JTextField
getJTFStaffEmail(){
        return
this.jtfStaffEmail;
    }

    //Getter method for
jtfCategory
    public JTextField
getJTFCategory(){
        return this.jtfCategory;
    }

    //Getter method for
jtfLanguageName
    public JTextField
getJTFLanguageName(){
        return
this.jtfLanguageName;
    }
```

```
    //Getter method for
jtfActorName
    public JTextField
getJTFActorName(){
        return
this.jtfActorName;
    }

    //Getter method for jtRental
    public JTable getJTRental(){
        return this.jtRental;
    }

    //Getter method for jbEdit
    public JButton getJBEdit(){
        return this.jbEdit;
    }

    //Getter method for jbInsert
    public JButton getJBInsert()
{
        return this.jbInsert;
    }

    //Getter method for jbDelete
    public JButton getJBDelete()
{
        return this.jbDelete;
    }

    //Getter method for jbChart
    public JButton getJBChart(){
        return this.jbChart;
    }

    //Getter method for jbFirst
    public JButton getJBFirst(){
        return this.jbFirst;
    }

    //Getter method for jbPrev
    public JButton getJBPrev(){
        return this.jbPrev;
    }

    //Getter method for jbNext
    public JButton getJBNext(){
        return this.jbNext;
    }

    //Getter method for jbLast
    public JButton getJBLast(){
        return this.jbLast;
```

```
    }
```

POPULATING TABLE AND COMBOBOXES

POPULATING TABLE AND COMBOBOXES

Step 1

In **Rental_Utils.java**, add two new methods: **get_rental_list()** and **show_**

- **get_rental_list()**: This method retrieves data from the database b returns an **ArrayList** of **Rental** objects. The **Rental** object is crea constructor that takes all the data retrieved from the database. used to ensure that the database connection is closed auton execution. If an **SQLException** occurs, a message dialog is display
- **show_table_rental()**: This method takes an **ArrayList** of **Rental** on the **RentalForm** frame. A **DefaultTableModel** is created with are set using the **set_column_header()** method. The **JTable** con set to use this model. The **Object** array row is used to store the da to the model using the **addRow()** method.

```
1     private static ArrayList<Rental> get_rental_list(RentalForm
2  frm, String sql, String item){
3         ArrayList<Rental> list = new ArrayList<>();
4
5         try(Connection conn = getConnection();
6             PreparedStatement ps = conn.prepareStatement(sql)){
7             if (item.equalsIgnoreCase("none")==false) {
8                  ps.setString(1,item);
9             }
10            ResultSet rs = ps.executeQuery();
11            Rental obj;
12
13            while(rs.next()){
14                //Using twenty-eight-params constructor
15                obj = new Rental(rs.getInt("rental_id"),
16                        rs.getDate("rental_date"),
17                        rs.getInt("inventory_id"),
18                        rs.getInt("customer_id"),
19                        rs.getDate("return_date"),
20                        rs.getInt("staff_id"),
21                        rs.getTimestamp("last_update"),
22                        rs.getInt("film_id"),
23                        rs.getString("title"),
24                        rs.getString("description"),
25                        rs.getInt("release_year"),
26                        rs.getInt("rental_duration"),
27                        rs.getDouble("rental_rate"),
28                        rs.getInt("length"),
```

```
                            rs.getDouble("replacement_cost"),
                            rs.getString("rating"),
                            rs.getString("special_features"),
                            rs.getString("language_name"),
                            rs.getString("category_name"),
                            rs.getString("actor_name"),
                            rs.getString("customer_name"),
                            rs.getString("customer_email"),
                            rs.getString("staff_name"),
                            rs.getString("staff_email"),
                            rs.getString("store_city"),
                            rs.getString("store_country"),
                            rs.getString("store_district"),
                            rs.getString("store_postal_code"));

                    list.add(obj);
                }
            }catch (SQLException ex){
                JOptionPane.showMessageDialog(frm, ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
            }
            return list;
        }

        private static void show_table_rental(RentalForm frm,
ArrayList<Rental> list) throws SQLException{
            DefaultTableModel model = new DefaultTableModel(0,0);

            String header[] = {"Rental ID", "Rental Date",
"Inventory ID", "Customer ID", "Return Date",
                "Staff ID", "Film ID", "Title", "Description",
"Release Year",
                "Rental Duration", "Rental Rate", "Length",
"Replacement Cost",
                "Rating", "Special Features", "Language",
"Category", "Actor Name",
                "Customer Name", "Customer Email", "Staff Name",
"Staff Email",
                "Store City", "Store Country", "Store District",
"Store Postal Code",
                "Last Update"};


model.setColumnIdentifiers(set_column_header(frm.getJTRental(),
header));
            frm.getJTRental().setModel(model);

            Object[] row = new Object[28];

            for(int i=0; i<list.size(); i++){
                row[0] = list.get(i).getRentalID();
                row[1] = list.get(i).getRentalRate();
                row[2] = list.get(i).getInventoryID();
                row[3] = list.get(i).getCustomerID();
```

```
83              row[4] = list.get(i).getReturnDate();
84              row[5] = list.get(i).getStaffID();
85              row[6] = list.get(i).getFilmID();
86              row[7] = list.get(i).getTitle();
87              row[8] = list.get(i).getDescription();
88              row[9] = list.get(i).getReleaseYear();
89              row[10] = list.get(i).getRentalDuration();
90              row[11] = list.get(i).getRentalRate();
91              row[12] = list.get(i).getLength();
92              row[13] = list.get(i).getReplacementCost();
93              row[14] = list.get(i).getRating();
94              row[15] = list.get(i).getSpecialFeatures();
95              row[16] = list.get(i).getLanguageName();
96              row[17] = list.get(i).getCategory();
97              row[18] = list.get(i).getActorName();
98              row[19] = list.get(i).getCustomerName();
99              row[20] = list.get(i).getCustomerEmail();
100             row[21] = list.get(i).getStaffName();
101             row[22] = list.get(i).getStaffEmail();
102             row[23] = list.get(i).getStoreCity();
103             row[24] = list.get(i).getStoreCountry();
                row[25] = list.get(i).getStoreDistrict();
                row[26] = list.get(i).getStorePostalCode();
                row[27] = list.get(i).getLastUpdate();

                model.addRow(row);
            }
        }
```

Step 2

In **Rental_Utils.java**, define **refresh_controls()** method. It refreshes the
takes a **RentalForm** object as input and updates its title, location, and se
that display rental, inventory, customer, and staff IDs.

Here's a breakdown of what the method does:

1. The **setLocationRelativeTo(null)** method centers the rental form
2. The **setTitle()** method updates the title of the rental form.
3. The **table_renderer()** method is used to apply alternating row col
4. The **get_rental_list()** method retrieves a list of rentals from th
   rental table.
5. The **populate_combobox()** method populates the **JComboBoxes**
   in the database.
6. If an exception occurs during the method execution, a dialog box w

Overall, this method updates the rental form to reflect the current state of

```
1       public static void
2   refresh_controls(RentalForm frm){
3
4   frm.setLocationRelativeTo(null);
5
```

```
        frm.setTitle("RENTAL FORM");

        //Shows the content of rental table and populates combobox
        try{
            //Makes alternating color for table rows

table_renderer(frm.getJTRental());

            //Populates table
            ArrayList<Rental> list = get_rental_list(frm, Query_Rental.get_sql_rental_joint() + " ORDER BY rental_id", "none");
            show_table_rental(frm, list);

            //Populates jcbRentalID
            String sql_rent_id = "SELECT rental_id FROM rental ORDER BY rental_id";

populate_combobox(sql_rent_id, frm.getJCBRentalID(), frm);

            //Populates jcbInventoryID
            String sql_inv_id = "SELECT inventory_id FROM inventory ORDER BY inventory_id";

populate_combobox(sql_inv_id, frm.getJCBInventoryID(), frm);

            //Populates jcbCustomerID
            String sql_cust_id = "SELECT customer_id FROM customer ORDER BY customer_id";

populate_combobox(sql_cust_id, frm.getJCBCustomerID(), frm);

            //Populates jcbStaffID
            String sql_stf_id = "SELECT staff_id FROM staff ORDER BY staff_id";

populate_combobox(sql_stf_id, frm.getJCBStaffID(), frm);
```

```
        }catch (SQLException ex){

JOptionPane.showMessageDialog(frm,
ex.getMessage(),

 "ERROR",JOptionPane.ERROR_MESSAGE)
;
        }
    }
```

Step 3

In **RentalForm**'s default constructor, the **Rental_Utils.refresh_con** populates the controls in the **RentalForm** with data from a database u from the **Rental_Utils** class.

The **this.setIconImage()** method sets the icon of the **R this.setDefaultCloseOperation(this.HIDE_ON_CLOSE)** method sets th to hide the form instead of exiting the application when the close button is

```
1  package sakila;
2  import java.awt.Toolkit;
3  import java.awt.event.ActionEvent;
4  import java.awt.event.ActionListener;
5  import java.io.IOException;
6  import java.text.ParseException;
7  import java.util.logging.Level;
8  import java.util.logging.Logger;
9  import javax.swing.JButton;
10 import javax.swing.JComboBox;
11 import javax.swing.JLabel;
12 import javax.swing.JTable;
13 import javax.swing.JTextField;
14
15 public class RentalForm extends javax.swing.JFrame {
16     public RentalForm() {
17         initComponents();
18         Utility.setLookAndFeel(this);
19         Rental_Utils.refresh_controls(this);
20
21  this.setIconImage(Toolkit.getDefaultToolkit().getImage(getClass().
22 ;
23         this.setDefaultCloseOperation(this.HIDE_ON_CLOSE);
24     }
25     //...
26 }
```

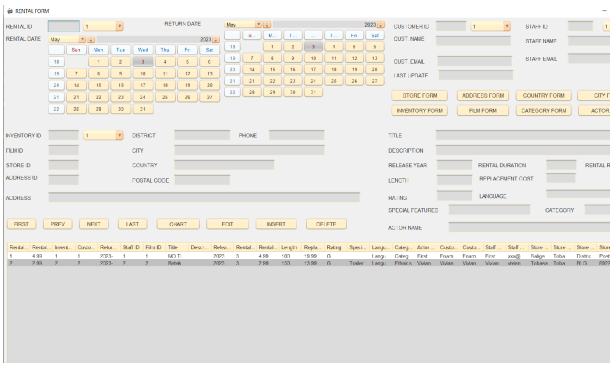


Figure 15.2 The content of **rental** table displaye

Step 4	Run the project to see the content of **rental** table displayed in **jtRental** as If you use the data from **Sakila** MySQL database available in the internet, table displayed in **jtRental** as shown in Figure 15.3.

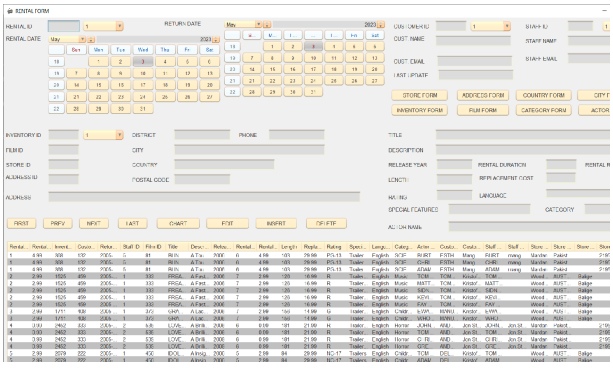


Figure 15.3 The the content of **rental** table in original **Sakila** database **jtRental**

DISPLAYING AND NAVIGATING DATA ROW BY ROW
DISPLAYING AND NAVIGATING DATA ROW BY ROW

Step 1	In **Rental_Utils**, define two new methods named **clear_sta** **display_staff_data()**. The **clear_staff_controls()** method is a private method that takes a **Renta** input and clears the text fields for staff ID, staff name, and staff email. Thi the **display_staff_data()** method when no matching data is found in the r the SQL query. The **display_staff_data()** method is a private generic method that takes a frm, an SQL query string sql, and a generic item item as input. This meth

executing an SQL query with a parameterized input value, populating t **RentalForm** object with the data returned from the query, and selectin value in the **JComboBox** for staff ID.

The try block in the **display_staff_data()** method sets up a connection prepares a **PreparedStatement** object with the provided SQL quer parameter value is set to the generic item using the **setObjec PreparedStatement** object. The query is executed using the **executeQu** returns a **ResultSet** object containing the query results.

The if statement checks if the result set is empty, and if so, calls the **cl** method to clear the text fields in the **RentalForm** object and returns. If result set, the do-while loop iterates through each row of data and popula the **RentalForm** object with the corresponding values. The **find_coml** method is called to select the corresponding value in the **JComboBox** for s

Finally, the **ResultSet** object and the **PreparedStatement** object are clos method in the finally block. If an SQL exception is caught, an error m displayed using the **JOptionPane** class.

```
1     private static void clear_staff_controls(RentalForm frm){
2         frm.getJTFStaffID().setText("");
3         frm.getJTFStaffName().setText("");
4         frm.getJTFStaffEmail().setText("");
5     }
6 
7    //Displays staff data result row by row
8     private static <T> void display_staff_data(RentalForm frm,
9 String sql, T item){
10        try(Connection conn = getConnection()){
11            PreparedStatement ps = conn.prepareStatement(sql);
12            ps.setObject(1,item);
13            ResultSet rs = ps.executeQuery();
14 
15            if (!rs.next()) {
16                // no row found, clear the form fields
17                clear_staff_controls(frm);
18                return;
19            }
20 
21            do{
22 
23 frm.getJTFStaffID().setText(String.valueOf(rs.getInt("staff_id")));
24 
25 frm.getJTFStaffName().setText(rs.getString("first_name") + " " +
26 rs.getString("last_name"));
27 
28 frm.getJTFStaffEmail().setText(rs.getString("email"));
29 
30                // Determines item selected from jcbStaffID
```

```
                find_combo_value_selected(frm.getJCBStaffID(),
rs.getInt("staff_id"));

            }while(rs.next());

            rs.close();
            ps.close();
        }catch(SQLException ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }
```

Step 2

Still in the same class, define another method named **jcbStaff_handler()**.

```
    public static void
jcbStaff_handler(RentalForm frm) {
        Object item =
frm.getJCBStaffID().getSelectedItem();
        display_staff_data(frm,
Query_Staff.get_sql_id(), item);
    }
```

Step 3

In **RentalForm**, double click on **jcbStaffID** combobox and define its handler as follows:

```
    private void
jcbStaffIDActionPerformed(java.awt.event.ActionEvent
evt) {
        Rental_Utils.jcbStaff_handler(this);
    }
```

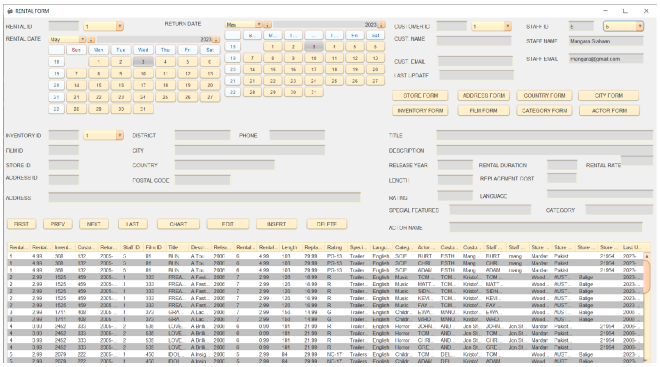


Figure 15.4 Displaying row by row the content of **staff** t

Step

Run the project. Choose one of items in **jcbStaffID** combobox to see row

4	**staff** table as shown in Figure 15.4.
Step 5	In **Rental_Utils**, define two new methods named **clear_custom** **display_customer_data()**. The **clear_customer_controls(RentalForm** the customer ID, customer name, and customer email fields on the rental called before displaying new customer data on the rental form. The **display_customer_data(RentalForm frm, String sql, T item)** customer data from the database on the rental form. It takes in three pa form object, a SQL query to retrieve the customer data, and the customer **JComboBox**. The method retrieves the customer data for the selecte populates the customer ID, customer name, and customer email fields on method also sets the selected item in the **JComboBox** to the selected cust

```
1     private static void clear_customer_controls(RentalForm frm){
2         frm.getJTFCustomerID().setText("");
3         frm.getJTFCustomerName().setText("");
4         frm.getJTFCustEmail().setText("");
5     }
6
7    //Displays customer data result row by row
8     private static <T> void display_customer_data(RentalForm frm, S
9 sql, T item){
10        try(Connection conn = getConnection()){
11            PreparedStatement ps = conn.prepareStatement(sql);
12            ps.setObject(1,item);
13            ResultSet rs = ps.executeQuery();
14
15            if (!rs.next()) {
16                // no row found, clear the form fields
17                clear_customer_controls(frm);
18                return;
19            }
20
21            do{
22
23 frm.getJTFCustomerID().setText(String.valueOf(rs.getInt("customer_i
24
25 frm.getJTFCustomerName().setText(rs.getString("first_name") + " " +
26 rs.getString("last_name"));
27                frm.getJTFCustEmail().setText(rs.getString("email")
28
29                // Determines item selected from jcbCustomerID
30                find_combo_value_selected(frm.getJCBCustomerID(),
31 rs.getInt("customer_id"));
32
33            }while(rs.next());
34
35            rs.close();
36            ps.close();
37        }catch(SQLException ex){
```

```
38              JOptionPane.showMessageDialog(frm, ex.getMessage(),
39                      "ERROR",JOptionPane.ERROR_MESSAGE);
40          }
41      }
42
43      public static void jcbCustomer_handler(RentalForm frm) {
44          Object item = frm.getJCBCustomerID().getSelectedItem();
45          display_customer_data(frm, Query_Customer.get_sql_id(), ite
        }
```

Step 6

Still in the same class, define another method named **jcbCustom** **jcbCustomer_handler(RentalForm frm)** method is an event handler th a customer is selected from the **JComboBox**. The method retrieves the se **JComboBox** and passes it as a parameter to the **display_customer_data** the customer data on the rental form.

```
1       public static void
2   jcbCustomer_handler(RentalForm frm) {
3           Object item =
4   frm.getJCBCustomerID().getSelectedItem();
5           display_customer_data(frm,
    Query_Customer.get_sql_id(), item);
        }
```

Step 7

In **RentalForm**, double click on **jcbCustomerID** combobox and define its handler as follows:

```
1       private void
2   jcbCustomerIDActionPerformed(java.awt.event.ActionEvent
3   evt) {
            Rental_Utils.jcbCustomer_handler(this);
        }
```

Step 8

Run the project. Choose one of items in **jcbCustomerID** combobox to content of **customer** table as shown in Figure 15.5.

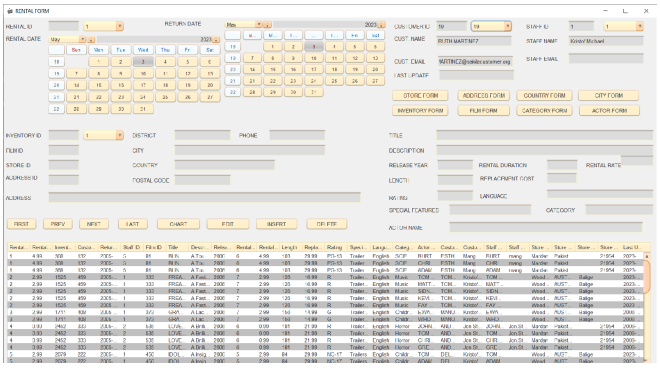


Figure 15.5 Displaying row by row the content of **custome**

Step 9

In **Rental_Utils**, define two new methods named **clear_invent** **display_inventory_data()**. The **display_inventory_data()** method take object, an SQL query string, and an item parameter of generic type T. It connection, prepares the SQL statement with the item parameter, and e no rows are returned, it clears the form fields using the **clear_inventory** Otherwise, it iterates through the result set and sets the correspondin using the **setText()** method. Finally, it selects the item in the **JC** **find_combo_value_selected()** method.

The **clear_inventory_controls()** method is used to clear all the inventory-

```
1     private static void clear_inventory_controls(RentalForm frm){
2         frm.getJTFInventoryID().setText("");
3         frm.getJTFFilmID().setText("");
4         frm.getJTFStoreID().setText("");
5         frm.getJTFAddressID().setText("");
6         frm.getJTFAddress().setText("");
7         frm.getJTFTitle().setText("");
8         frm.getJTFDescription().setText("");
9         frm.getJTFReleaseYear().setText("");
10        frm.getJTFRentalDuration().setText("");
11        frm.getJTFRentalRate().setText("");
12        frm.getJTFLength().setText("");
13        frm.getJTFReplacementCost().setText("");
14        frm.getJTFSpecialFeatures().setText("");
15        frm.getJTFCategory().setText("");
16        frm.getJTFLanguageName().setText("");
17        frm.getJTFActorName().setText("");
18        frm.getJTFDistrict().setText("");
19        frm.getJTFPhone().setText("");
20        frm.getJTFPostalCode().setText("");
21        frm.getJTFCity().setText("");
22        frm.getJTFCountry().setText("");
23    }
24   //Displays inventory data result row by row
25    private static <T> void display_inventory_data(RentalForm frm,
26 {
```

```
27          try(Connection conn = getConnection()){
28              PreparedStatement ps = conn.prepareStatement(sql);
29              ps.setObject(1,item);
30              ResultSet rs = ps.executeQuery();
31
32              if (!rs.next()) {
33                  // no row found, clear the form fields
34                  clear_inventory_controls(frm);
35                  return;
36              }
37
38              do{
39
40  frm.getJTFInventoryID().setText(String.valueOf(rs.getInt("inventory
41                  frm.getJTFFilmID().setText(String.valueOf(rs.getInt
42                  frm.getJTFStoreID().setText(String.valueOf(rs.getIn
43
44  frm.getJTFAddressID().setText(String.valueOf(rs.getInt("address_id"
45                  frm.getJTFAddress().setText(rs.getString("store_add
46                  frm.getJTFDistrict().setText(rs.getString("store_di
47                  frm.getJTFCity().setText(rs.getString("store_city")
49                  frm.getJTFCountry().setText(rs.getString("store_cou
50                  frm.getJTFPhone().setText(rs.getString("store_phone
51                  frm.getJTFPostalCode().setText(rs.getString("store_
52                  frm.getJTFTitle().setText(rs.getString("title"));
53                  frm.getJTFDescription().setText(rs.getString("descr
54
55  frm.getJTFReleaseYear().setText(String.valueOf(rs.getInt("release_y
56
57  frm.getJTFRentalDuration().setText(String.valueOf(rs.getInt("rental
58
59  frm.getJTFRentalRate().setText(String.valueOf(rs.getDouble("rental_
60                  frm.getJTFLength().setText(String.valueOf(rs.getInt
61
62  frm.getJTFReplacementCost().setText(String.valueOf(rs.getDouble("re
63                  frm.getJTFSpecialFeatures().setText(rs.getString("s
64                  frm.getJTFRating().setText(rs.getString("rating"));
65                  frm.getJTFLanguageName().setText(rs.getString("lang
66                  frm.getJTFActorName().setText(rs.getString("actor_n
67                  frm.getJTFCategory().setText(rs.getString("category
68
69                  // Determines item selected from jcbInventoryID
70                  find_combo_value_selected(frm.getJCBInventoryID(),
71  rs.getInt("inventory_id"));
72
73
74
75
76
77
78
79
80
81
```

82
83
84
85

```
            }while(rs.next());

            rs.close();
            ps.close();
        }catch(SQLException ex){

JOptionPane.showMessageDialog(frm,
ex.getMessage(),

 "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }
```

Step 10

Still in the same class, **jcbInventory_handler()**. It is an e item is selected in the **JComboBo** item from the **JComboBox** and **display_inventory_data()** method. The **display_inventory_data()** met inventory data from the databas populates the **RentalForm** fields wi

This method clears the form fields item.

```
1  public static void
2  jcbInventory_handler(RentalForm
3          Object item =
4  frm.getJCBInventoryID().getSelec
5          display_inventory_data(
   Query_Rental.get_sql_inventory_
   WHERE inventory_id = ?", item);
    }
```

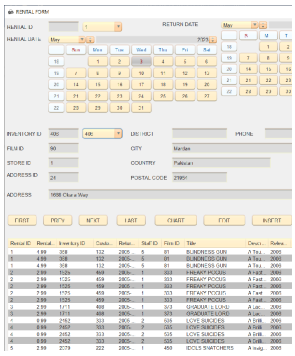
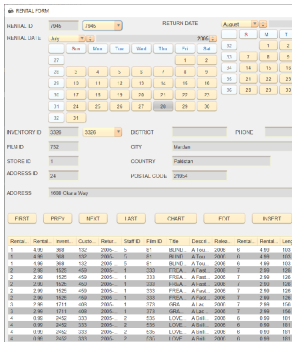


Figure 15.6 Displaying row by

Step 11	In **RentalForm**, double click on **jc** corresponding event handler as follc `1 private void jcbInventoryIDActionPerformed(ja 2 evt) { 3 Rental_Utils.jcbInventor this.jcbStaffID); }`
Step 12	Run the project. Choose one of ite row by row the content of **inventory**
Step 13	In **Rental_Utils**, define a new m displays rental data row by row in parameters: the rental form object item to be used in the query (**item** with any type of item. The method starts by establishing **getConnection()** method. It then **PreparedStatement** class and **setObject()** method. It executes th the **executeQuery()** method and sto If the result set is empty, it clears result set is not empty, it iterates t For each row, it sets the rental form the result set using the **setText()** m method for date fields. The method also calls the **find** determine which item is selected ir the result set. Finally, the method closes the result catches any SQL exceptions that oc displaying an error message dialog b `1 //Displays rental data resul 2 private static <T> void dis 3 String sql, T item){ 4 try(Connection conn = g 5 PreparedStatement p 6 ps.setObject(1,item`

```
7               ResultSet rs = ps.e
8
9               if (!rs.next()) {
10                  // no row found
11                  frm.getJTFRenta
12                  frm.getJTFLastU
13                  return;
14              }
15
16              do{
17
18  frm.getJTFRentalID().setText(St
19
20  frm.getJTFLastUpdate().setText(
21
22                  java.util.Date
23  Date(rs.getTimestamp("rental_da
24                  frm.getJCRentalD
25
26                  java.util.Date
27  Date(rs.getTimestamp("return_da
28                  frm.getJCReturnD
29
30                  // Determines i
31                  find_combo_valu
32  rs.getInt("rental_id"));
33
34                  // Determines i
35                  find_combo_valu
36  rs.getInt("inventory_id"));
37
38                  // Determines i
39                  find_combo_valu
40  rs.getInt("staff_id"));
41
42                  // Determines i
43                  find_combo_valu
44  rs.getInt("customer_id"));
45
46              }while(rs.next());
47
49              rs.close();
50              ps.close();
51          }catch(SQLException ex)
52              JOptionPane.showMes
53                      "ERROR",JOp
54          }
        }
```

Step 14	Still in the same class, define anoth It handles a selection event for a co

	The method takes a rental form obje
	
	The first line of the method retrieve using the **getSelectedItem()** metl called item.
	
	The method then calls the **displa** rental form object (**frm**), a SQL qu and the item object variable. This m in the rental form based on the selec
	
	`1 public static void jcbRental_handler(RentalForm frm`
	`2 Object item = frm.getJCBRentalID().getSelected`
	`3 display_rental_data(frm, Query_Rental.get_sql_id(), item)`
	`4 }`
	
	 
	Figure 15.7 Displaying row

Step 15	In **RentalForm**, double click on corresponding event handler as follo
	
	`1 private void jcbRentalIDActionPerformed(java.evt) {`
	`2 Rental_Utils.jcbRental_H this.jcbStaffID);`
	`3 }`

Step 16	Run the project. Choose one of iten by row the content of **rental** table a

Step 17

Define four navigating methods in
that allow the user to navigate thr
form. The methods take a rental forr

The **show_first_row()** method retri
combo box (**jcbRentalID**) and p
method, which displays the rental d
sets the currentIndex variable to the

The **show_last_row()** method retri
combo box and passes it to the
displays the rental data for that
currentIndex variable to the last in

The **show_prev_row()** method dec
retrieves the rental ID of the previc
index is less than the first index, it
and returns. Otherwise, it passes th
method, which displays the rental da

The **show_next_row()** method incr
retrieves the rental ID of the next rc
is greater than the last index, it set
returns. Otherwise, it passes the
method, which displays the rental da

```
1     public static void show_fi
2         String item =
3 String.valueOf(frm.getJCBRental
4         display_rental_data(frm
5         currentIndex = FIRST_IN
6     }
7
8     public static void show_las
9         int endIndex = frm.getJ
10 1;
11        String item =
12 String.valueOf(frm.getJCBRental
13        display_rental_data(frm
14        currentIndex = endIndex
15    }
16
17    public static void show_pre
18        currentIndex--;
19        if(currentIndex < FIRST
20            currentIndex = FIRS
21            return;
22        }
23
24
```

```
        String item =
String.valueOf(frm.getJCBRental
        display_rental_data(frm
    }

    public static void show_nex
        int endIndex = frm.getJ
1;
        currentIndex++;
        if(currentIndex > endIr
            currentIndex = endI
            return;
        }
        String item =
String.valueOf(frm.getJCBRental
        display_rental_data(frm
    }
```

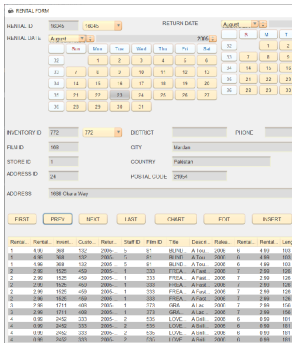


Figure 15.8 User clicks on one or

Step 18

Then in **RentalForm**, double click o
corresponding event handler:

```
    private void
jbFirstActionPerformed(java.awt
evt) {
        Rental_Utils.show_first
    }

    private void
jbPrevActionPerformed(java.awt.
evt) {
        Rental_Utils.show_prev_
    }

    private void
jbNextActionPerformed(java.awt.
evt) {
        Rental_Utils.show_next_
    }
```

```
    private void
jbLastActionPerformed(java.awt.
evt) {
        Rental_Utils.show_last_
    }
```

These are event handler methods f
buttons in the **RentalForm** GUI. W
corresponding method is called, v
Rental_Utils class to display the fi
row of data in the store table, respe
calls refers to the current **Renta**
parameter to the **Rental_Utils** meth

Step 19

Run the project. Click on one or mo
shown in Figure 15.8.

Step 20

Define **mouse_pressed_handler()**
method handles the mouse press ev
It first checks if a row is selected ar
it retrieves the rental ID of
display_rental_data() method to di
the rental form. If there is an SQL e
error message with the exception m

```
1  public static void mouse_pr
2      Objects.requireNonNull(
3
4      int selectedIndex = frm
5      if (selectedIndex == -1
6          JOptionPane.showMessa
7  data.",
8              "No row selecte
9          return;
10     }
11
12     try (Connection conn =
13         String id =
14 String.valueOf(frm.getJTRental(
15 0));
16
17         // Displays rental
18         display_rental_data
19
20     } catch (SQLException e
21
22 Logger.getLogger(RentalForm.cla
23 displaying rental data", ex);
```

```
24                String message = "E
25 ex.getMessage();
26                String stackTrace =
                  JOptionPane.showMes
stackTrace, "ERROR", JOptionPar
            }
        }
```

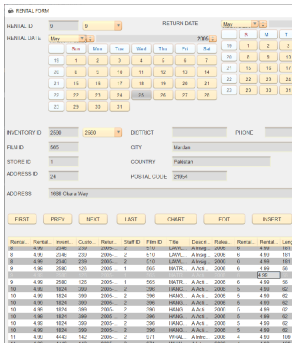


Figure 15.9 User double

Step 21

Right click on **jtRental**. Then, choo
Define its event handler:

```
1     private void
2 jtRentalMousePressed(java.awt.ev
3 evt) {

   Rental_Utils.mouse_pressed_hand
      }
```

Step 22

Run the project. Double click on an
corresponding row of **rental** table d
shown in Figure 15.9.

UPDATING RECORD

UPDATING RECORD

Step 1	In **Rental_Utils** class, define a new method named **update_row_by_rental_id()**. It updates a row of data in the **rental** table by rental ID. The method takes in a **RentalForm** object, an int representing the rental ID, five ints and a **Date** object representing the

rental date, inventory ID, customer ID, return date, staff ID respectively.

Inside the method, a connection to the database is established using the **getConnection()** method. A query is then executed to check if the rental ID exists in the **rental** table. If the rental ID is not found, an error message is displayed to the user.

If the rental ID is found, a **Rental** object is created using the parameters passed into the method, and an update query is executed to update the row of data in the rental table with the new information. The method catches **SQLException** and **NumberFormatException** and logs the error messages and stack traces.

Finally, the **ResultSet**, **PreparedStatements**, and **Connection** objects are closed to free up resources.

```
    //Updates row of data in rental tabel by rental_id
    public static void update_row_by_rental_id(RentalForm frm,
int rental_id,
            Date rent_date, int inv_id, int cust_id, Date
ret_date, int staff_id) throws SQLException{
        Connection conn = getConnection();
        ResultSet rs = null;
        String query_id = "SELECT rental_id FROM rental WHERE
rental_id = ?";
        String update_query = """
            UPDATE rental SET rental_date = ?, inventory_id =
?, customer_id = ?,
            return_date = ?, staff_id = ? WHERE rental_id =
?""";
        try(PreparedStatement idPs =
conn.prepareStatement(query_id,

ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_READ_ONLY);
        PreparedStatement updatePS =
conn.prepareStatement(update_query,
                ResultSet.TYPE_SCROLL_SENSITIVE,
ResultSet.CONCUR_UPDATABLE))
        {
            idPs.setInt(1,rental_id);
            if(!idPs.execute()){
                String message = "Can't find rental_id " +
rental_id;

                JOptionPane.showMessageDialog(frm, message,
                    "ERROR",JOptionPane.ERROR_MESSAGE);
            } else{
                rs = idPs.getResultSet();
                rs.next();

```

```
                //Creates a Rental object using seven-params
constructor
                Rental obj = new Rental(rental_id, rent_date,
inv_id, cust_id, ret_date, staff_id, new
Timestamp(System.currentTimeMillis()));
                updatePS.setDate(1, obj.getRentalDate());
                updatePS.setInt(2, obj.getInventoryID());
                updatePS.setInt(3, obj.getCustomerID());
                updatePS.setDate(4, obj.getReturnDate());
                updatePS.setInt(5, obj.getStaffID());
                updatePS.setInt(6, obj.getRentalID());

                updatePS.executeUpdate();
                rs.close();
                updatePS.close();
                idPs.close();
                conn.close();
            }
        }catch(SQLException ex){

Logger.getLogger(RentalForm.class.getName()).log(Level.SEVERE,
"Error updating rental data", ex);
            String message = "Error updating rental data: " +
ex.getMessage();
            String stackTrace =
Arrays.toString(ex.getStackTrace());
            JOptionPane.showMessageDialog(null, message +
"\n\n" + stackTrace, "ERROR", JOptionPane.ERROR_MESSAGE);
        }catch(java.lang.NumberFormatException ex){

Logger.getLogger(RentalForm.class.getName()).log(Level.SEVERE,
"Invalid Input", ex);
            String message = "Invalid Input: " +
ex.getMessage();
            String stackTrace =
Arrays.toString(ex.getStackTrace());
            JOptionPane.showMessageDialog(null, message +
"\n\n" + stackTrace, "ERROR", JOptionPane.ERROR_MESSAGE);
        }
    }
```

Step 2

Then in the same class, define a new method **read_inputs()**. It reads input data from a rental form and performs some validation. It returns a **HashMap** object that contains the input data. Here's a breakdown of the code:

- The method takes a **RentalForm** object as input, which is a GUI form used for entering rental data.

- A **HashMap** object called **input_data** is created to store the input data.
- The method reads the selected customer ID, rental ID, staff ID, and inventory ID from the **RentalForm** object using the **getSelectedItem()** method. These values are converted to strings using the **String.valueOf()** method.
- The method reads the selected rental date and return date from the **RentalForm** object using the **getDate()** method. These dates are formatted as strings using a **SimpleDateFormat** object.
- The method performs validation on the rental ID, staff ID, customer ID, and inventory ID to ensure that they are positive integers. If any of these values are not valid, an exception is thrown and an error message is displayed using a **JOptionPane** dialog box.
- The validated input data is added to the **HashMap** object using the **put()** method.
- The method returns the **HashMap** object containing the input data.

```
1   private static HashMap<String, String>
2  read_inputs(RentalForm frm) {
3          HashMap<String, String> input_data = new HashMap<>
4  ();
5          String cust_id =
6  String.valueOf(frm.getJCBCustomerID().getSelectedItem());
7          String rental_id =
8  String.valueOf(frm.getJCBRentalID().getSelectedItem());
9          String staff_id =
10 String.valueOf(frm.getJCBStaffID().getSelectedItem());
11         String inv_id =
12 String.valueOf(frm.getJCBInventoryID().getSelectedItem());
13
14         //Gets the selected date from the
15 getJCRentalDate()
16         java.util.Date rental_date =
17 frm.getJCRentalDate().getDate();
18         //Create a SimpleDateFormat object with the
19 desired date format
20         SimpleDateFormat sdf = new SimpleDateFormat("yyyy-
21 MM-dd");
22         //Formats the date as a String using the
23 SimpleDateFormat object
24         String rent_date = sdf.format(rental_date);
25
26         //Gets the selected date from the
27 getJCRentalDate()
28         java.util.Date return_date =
29 frm.getJCReturnDate().getDate();
30         //Formats the date as a String using the
31 SimpleDateFormat object
```

```
        String ret_date = sdf.format(return_date);

        // Validate user input
        int rental_id_int = 0;
        try {
            rental_id_int = Integer.parseInt(rental_id);
            if (rental_id_int <= 0) {
                throw new IllegalArgumentException("Rental
ID cannot be negative or zero");
            }
        } catch (NumberFormatException ex) {
            JOptionPane.showMessageDialog(frm, "Invalid
Rental ID: " + rental_id,
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        } catch (IllegalArgumentException ex) {
            JOptionPane.showMessageDialog(frm,
ex.getMessage(),
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        }

        int staff_id_int = 0;
        try {
            staff_id_int = Integer.parseInt(staff_id);
            if (staff_id_int <= 0) {
                throw new IllegalArgumentException("Staff
ID cannot be negative or zero");
            }
        } catch (NumberFormatException ex) {
            JOptionPane.showMessageDialog(frm, "Invalid
Staff ID: " + staff_id,
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        } catch (IllegalArgumentException ex) {
            JOptionPane.showMessageDialog(frm,
ex.getMessage(),
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        }

        int cust_id_int = 0;
        try {
            cust_id_int = Integer.parseInt(cust_id);
            if (cust_id_int <= 0) {
                throw new
IllegalArgumentException("Customer ID cannot be negative
or zero");
            }
        } catch (NumberFormatException ex) {
            JOptionPane.showMessageDialog(frm, "Invalid
Manager Customer ID: " + cust_id,
            "Error", JOptionPane.ERROR_MESSAGE);

```

```
            throw ex;
        } catch (IllegalArgumentException ex) {
            JOptionPane.showMessageDialog(frm,
ex.getMessage(),
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        }

        int inv_id_int = 0;
        try {
            inv_id_int = Integer.parseInt(inv_id);
            if (inv_id_int <= 0) {
                throw new
IllegalArgumentException("Inventory ID cannot be negative
or zero");
            }
        } catch (NumberFormatException ex) {
            JOptionPane.showMessageDialog(frm, "Invalid
Manager Inventory ID: " + inv_id,
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        } catch (IllegalArgumentException ex) {
            JOptionPane.showMessageDialog(frm,
ex.getMessage(),
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        }

        input_data.put("cust_id", cust_id);
        input_data.put("inv_id", inv_id);
        input_data.put("staff_id", staff_id);
        input_data.put("rental_id", rental_id);
        input_data.put("rent_date", rent_date);
        input_data.put("ret_date", ret_date);

        return input_data;
    }
```

Step 3

Still in the same class, define another method named **edit_actual()**. This method is responsible for updating the rental record in the database based on the input data entered by the user in the **RentalForm**. It first calls the **read_inputs()** method to retrieve the user input data and parse it into the appropriate data types. It then uses a **SimpleDateFormat** object to parse the rental date and return date strings into **java.util.Date** objects, and then converts these into **java.sql.Date** objects.

Once the necessary data has been retrieved and formatted, the method calls the **update_row_by_rental_id()** method to update the corresponding rental record in the database. If an **SQLException** is

thrown during this process, the method displays an error message dialog to the user.

Finally, the method calls the **refresh_controls()** method to update all objects on the RentalForm with the new data from the database.

```
    private static void edit_actual(RentalForm
frm) throws ParseException{
        try{
            HashMap<String, String> input_data
= read_inputs(frm);
            int cust_id =
Integer.parseInt(input_data.get("cust_id"));
            int inv_id =
Integer.parseInt(input_data.get("inv_id"));
            int staff_id =
Integer.parseInt(input_data.get("staff_id"));
            int rental_id =
Integer.parseInt(input_data.get("rental_id"));
            String str_rental_date =
input_data.get("rent_date");
            String str_ret_date =
input_data.get("ret_date");

            SimpleDateFormat dateFormat = new
SimpleDateFormat("yyyy-MM-dd");
            java.util.Date rent_date =
dateFormat.parse(str_rental_date);
            java.sql.Date rental_sql_date =
new java.sql.Date(rent_date.getTime());

            java.util.Date ret_date =
dateFormat.parse(str_ret_date);
            java.sql.Date return_sql_date =
new java.sql.Date(ret_date.getTime());

            update_row_by_rental_id(frm,
rental_id, rental_sql_date, inv_id, cust_id,
return_sql_date, staff_id);

            //Refreshes all objects on form
            refresh_controls(frm);

        }catch(SQLException ex){
            JOptionPane.showMessageDialog(frm,
ex.getMessage(),

 "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }
```

Step 4

Lastly, define two new methods named **enable_controls()** and **edit_handler()**. The **enable_controls()** method takes a boolean value and a **RentalForm** object as parameters. It then sets the enabled state of various form controls based on the boolean value passed in.

The edit_handler method first checks if the text of the **jbEdit** button is "EDIT". If it is, it changes the text to "CONFIRM" and disables all form controls using the enable_controls method. If the text is not "EDIT", it calls the **edit_actual()** method to perform the actual editing of the data and then re-enables the form controls using the **enable_controls()** method.

```
1     private static void enable_controls(boolean state,
2  RentalForm frm){
3          frm.getJBFirst().setEnabled(state);
4          frm.getJBPrev().setEnabled(state);
5          frm.getJBNext().setEnabled(state);
6          frm.getJBLast().setEnabled(state);
7          frm.getJBInsert().setEnabled(state);
8          frm.getJBDelete().setEnabled(state);
9          frm.getJTFAddressID().setEnabled(state);
10         frm.getJTFStoreID().setEnabled(state);
11         frm.getJTFAddress().setEnabled(state);
12         frm.getJTFDistrict().setEnabled(state);
13         frm.getJTFPhone().setEnabled(state);
14         frm.getJTFPostalCode().setEnabled(state);
15         frm.getJTFCity().setEnabled(state);
16         frm.getJTFCountry().setEnabled(state);
17         frm.getJTFStaffID().setEnabled(state);
18         frm.getJTFStaffName().setEnabled(state);
19         frm.getJTFStaffEmail().setEnabled(state);
20         frm.getJTFLastUpdate().setEnabled(state);
21         frm.getJTFFilmID().setEnabled(state);
22         frm.getJTFCustEmail().setEnabled(state);
23         frm.getJTFCustomerName().setEnabled(state);
24         frm.getJTFTitle().setEnabled(state);
25         frm.getJTFDescription().setEnabled(state);
26         frm.getJTFReleaseYear().setEnabled(state);
27         frm.getJTFRentalDuration().setEnabled(state);
28         frm.getJTFRentalRate().setEnabled(state);
29         frm.getJTFLength().setEnabled(state);
30         frm.getJTFReplacementCost().setEnabled(state);
31         frm.getJTFSpecialFeatures().setEnabled(state);
32         frm.getJTFCategory().setEnabled(state);
33         frm.getJTFLanguageName().setEnabled(state);
34         frm.getJTFActorName().setEnabled(state);
35     }
36
37     public static void edit_handler(RentalForm frm){
38         if(frm.getJBEdit().getText().equals("EDIT")){
39             frm.getJBEdit().setText("CONFIRM");
```

```
            // Disables controls
            enable_controls(false, frm);
        }

        else {
            try {
                frm.getJBEdit().setText("EDIT");

                // Actual editing
                edit_actual(frm);

                //Enables controls
                enable_controls(true, frm);
            } catch (ParseException ex) {

Logger.getLogger(Rental_Utils.class.getName()).log(Level.SEVERE,
null, ex);
            }
        }
    }
```

Step 5

Then, double click on **jbEdit** button on **RentalForm** to define its event listener:

```
    private void
jbEditActionPerformed(java.awt.event.ActionEvent
evt) {
        Rental_Utils.edit_handler(this);
    }
```

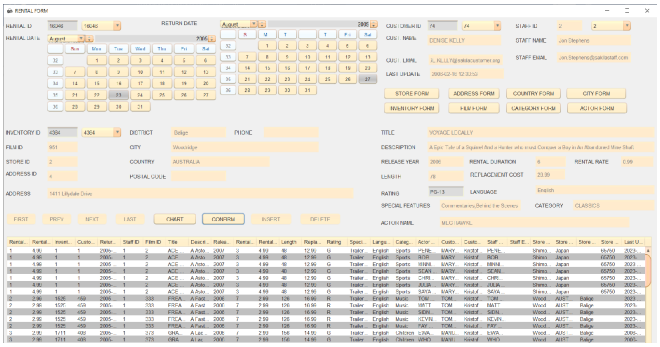


Figure 15.9 The rental form is in editing state

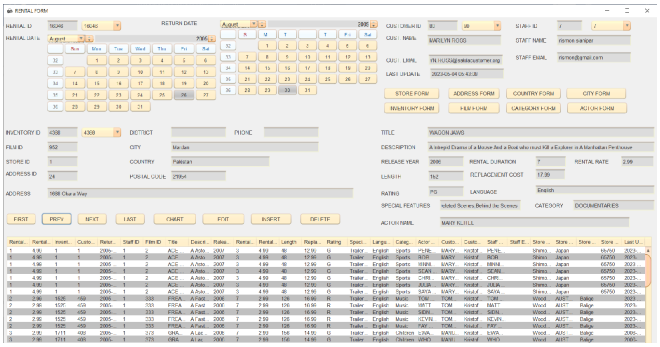


Figure 15.10 The edited row had been saved into database

Step 6	Run the project. Choose **rental_id** using **jcbRentalID** combobox. Or, you can choose one of rows in **jtRental** (in this case, **rental_id = 16046**). Then, click on EDIT button as shown in Figure 15.9. Edit any field you want (including choosing rental date and return date from **JCalendars**). Then, click on CONFIRM button. The edited row had been saved into **rental** table as shown in Figure 15.10.

INSERTING NEW RECORD
INSERTING NEW RECORD

Step 1	In **Rental_Utils** class, define a method named **insert_row()**. It is responsible for inserting a new row into the **rental** table. The method first reads the input values from the **RentalForm** object and converts the date strings to **java.sql.Date** objects. Then it creates a **Rental** object using the input values and passes it to a **PreparedStatement** object to execute the SQL INSERT statement.
	
	The INSERT statement takes the values of **rental_date**, **inventory_id**, **customer_id**, **return_date**, and **staff_id** from the **Rental** object and inserts them into the **rental** table. The **getConnection()** method establishes a connection to the database using the **DriverManager** class, and the **PreparedStatement** object is used to execute the SQL statement with the appropriate parameter values. If there is an error during the insert operation, the catch block catches the **SQLException** and displays an error message to the user using a **JOptionPane**.
	
	`1` `2` `3` `4` `5`
	`//Inserts new row into rental table`
	`private static void insert_row(RentalForm frm) throws SQLException, ParseException{`
	`HashMap<String, String> input_data = read_inputs(frm);`

```
        int cust_id =
Integer.parseInt(input_data.get("cust_id"));
        int inv_id =
Integer.parseInt(input_data.get("inv_id"));
        int staff_id =
Integer.parseInt(input_data.get("staff_id"));
        String str_rental_date = input_data.get("rent_date");
        String str_ret_date = input_data.get("ret_date");

        SimpleDateFormat dateFormat = new
SimpleDateFormat("yyyy-MM-dd");
        java.util.Date rent_date =
dateFormat.parse(str_rental_date);
        java.sql.Date rental_sql_date = new
java.sql.Date(rent_date.getTime());

        java.util.Date ret_date =
dateFormat.parse(str_ret_date);
        java.sql.Date return_sql_date = new
java.sql.Date(ret_date.getTime());

        // SQL insert statement
        String sql = """
            INSERT INTO rental(rental_date, inventory_id,
customer_id,
                return_date, staff_id) VALUES(?, ?, ?, ?,
?)""";

        try(Connection conn = getConnection();
         PreparedStatement pstmt = conn.prepareStatement(sql))
{

            //Creates a Rental object six-params constructor
            Rental obj = new Rental(rental_sql_date, inv_id,
cust_id, return_sql_date, staff_id, new
Timestamp(System.currentTimeMillis()));
            pstmt.setDate(1,obj.getRentalDate());
            pstmt.setInt(2,obj.getInventoryID());
            pstmt.setInt(3,obj.getCustomerID());
            pstmt.setDate(4,obj.getReturnDate());
            pstmt.setInt(5, obj.getStaffID());

            //Executes the sql insert statement
            pstmt.executeUpdate();
        } catch (SQLException ex) {

Logger.getLogger(RentalForm.class.getName()).log(Level.SEVERE,
"Database error", ex);
            JOptionPane.showMessageDialog(frm, "Error:
Database error\n" + ex.getMessage());
        }
    }
```

Step 2	Still in **Rental_Utils.java**, define **insert_actual()** and **insert_handler()** methods. The **insert_handler()** method is used to handle the "INSERT" button in the form. If the button is currently displaying "INSERT", it changes the button text to "CONFIRM", disables the edit button, disables some controls, and clears some input fields. If the button is currently displaying "CONFIRM", it calls **insert_actual()**, changes the button text back to "INSERT", enables the edit button, and enables the controls.

```
1     private static void insert_actual(RentalForm frm) throws
2  ParseException{
3          try{
4              insert_row(frm);
5
6              //Refreshes table and comboboxes
7              refresh_controls(frm);
8
9          }catch(SQLException ex){
10             JOptionPane.showMessageDialog(frm, ex.getMessage(),
11                     "ERROR",JOptionPane.ERROR_MESSAGE);
12         }
13     }
14
15     public static void insert_handler(RentalForm frm){
16         if(frm.getJBInsert().getText().equals("INSERT")){
17             frm.getJBInsert().setText("CONFIRM");
18
19             //Disables jbEdit
20             frm.getJBEdit().setEnabled(false);
21
22             // Disables controls
23             enable_controls(false, frm);
24             frm.getJCBRentalID().setEnabled(false);
25
26             // Clears controls
27             clear_customer_controls(frm);
28
29             // Enables
30             frm.getJBInsert().setEnabled(true);
31         }
32
33         else {
34             frm.getJBInsert().setText("INSERT");
35
36             try {
37                 // Actual insertion
38                 insert_actual(frm);
39             } catch (ParseException ex) {
40
41 Logger.getLogger(Rental_Utils.class.getName()).log(Level.SEVERE,
```

```
42 null, ex);
43             }
44
45             //Enables jbEdit
46             frm.getJBEdit().setEnabled(true);
47
48             //Enables controls
49             enable_controls(true, frm);
50             frm.getJCBRentalID().setEnabled(true);
         }
     }
```

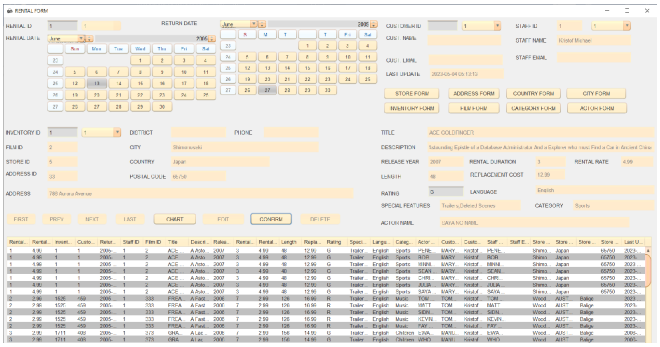


Figure 15.11 When user clicks on INSERT button, the rental form will be in state of insertion

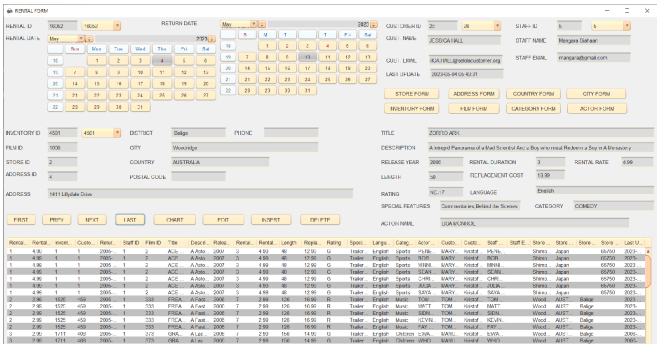


Figure 15.12 The new data had been saved into **rental** table

Step 3

In **RentalForm.java**, double click on INSERT button to create its event listener:

```
1     private void
2 jbInsertActionPerformed(java.awt.event.ActionEvent
3 evt) {
          Rental_Utils.insert_handler(this);
      }
```

Step 4	Run the project. Click on INSERT button. You will see the state of rental form when insertion is in progress as shown in Figure 15.11. Then, fill in all fields. Then, click CONFIRM button to save the new record into **rental** table as shown in Figure 15.12.

DELETING RECORD
DELETING RECORD

Step 1

Then in **Rental_Utils** class, define **delete_handler()** method. It hand deletion of a row from the **rental** table. It first prompts the user confirmation message, asking if they are sure they want to delete the rc the selected rental ID. If the user confirms, a SQL DELETE statement is ex using a **PreparedStatement** object to avoid SQL injection attack **PreparedStatement** object is created by setting the **rental_id** paramete the **setInt()** method. If the deletion is successful, the table and combobo refreshed. If an **SQLException** occurs, an error message is displayed to t using a **JOptionPane**.

```
1     public static void delete_handler(RentalForm frm){
2         int dialogButton = JOptionPane.YES_NO_OPTION;
3         int rental_id =
4 Integer.parseInt(String.valueOf(frm.getJCBRentalID().getSelectedIte
5 
6         String message = String.format("Are you sure you want to de
7 the row Rental ID: %d)", rental_id);
8         int answer = JOptionPane.showConfirmDialog(frm, message,
9 "DELETING ROW OF DATA", dialogButton);
10 
11        if(answer == JOptionPane.YES_OPTION){
12            String query = """
13                DELETE FROM rental WHERE rental_id = ?""";
14            try(Connection conn = getConnection();
15                PreparedStatement ps = conn.prepareStatement(query)
16                // Use PreparedStatement to avoid SQL injection att
17                ps.setInt(1, rental_id);
18                ps.executeUpdate();
19 
20                // Refresh table and comboboxes
21                refresh_controls(frm);
22 
23            } catch (SQLException ex){
24                JOptionPane.showMessageDialog(frm, ex.getMessage(),
25                    "ERROR",JOptionPane.ERROR_MESSAGE);
26            }
27        }
28    }
```

Step 2	In **RentalForm.java**, double click on DELETE button to generate its listener: `1 2 3` `private void jbDeleteActionPerformed(java.awt.event.ActionEvent evt) { Rental_Utils.delete_handler(this); }`
Step 3	Run the project. Choose **rental_id** using **jcbRentalID** combobox. Then, C DELETE button. The corresponding row of data had been deleted from dat

PLOTTING CHART

PLOTTING CHART

Step 1	Create a new **JFrame** and save it as **Charts_Rental.java**.
Step 2	In **Charts_Rental.java**, add six **JPanels** and set their corresponding **Va Name** as **jPanel1**, **jPanel2**, **jPanel3**, **jPanel4**, **jPanel5**, and **jPanel6**. add getter method for each object as follows: `1–24` `//Getter method for jPanel1 public JPanel getJPanel1(){ return this.jPanel1; } //Getter method for jPanel2 public JPanel getJPanel2(){ return this.jPanel2; } //Getter method for jPanel3 public JPanel getJPanel3(){ return this.jPanel3; } //Getter method for jPanel4 public JPanel getJPanel4(){ return this.jPanel4; } //Getter method for jPanel5 public JPanel getJPanel5(){ return this.jPanel5; }`

```
25
26      //Getter method for jPanel6
27      public JPanel getJPanel6(){
28          return this.jPanel6;
29      }
```

Step 3

In **Rental_Utils** class, define six new methods. These are several private methods that are responsible for drawing different types of charts rela rental data.

1. **draw_pie_chart_rental_by_rental_year(Charts_Rental frm, J jp)**: Draws a pie chart that represents the rental distribution by th of rental date. The data for the chart is obtained by callin **create_pie_dataset()** method from the **Query_Rental** class, pass the appropriate SQL query and column names for the number o rented and the year of rental date.
2. **draw_pie_chart_rental_by_rental_month(Charts_Rental frm, J jp)**: Draws a pie chart that represents the rental distribution b month of rental date. The data for the chart is obtained by calli **create_pie_dataset()** method from the **Query_Rental** class, pass the appropriate SQL query and column names for the number o rented and the month of rental date.
3. **draw_pie_chart_rental_by_rental_week(Charts_Rental frm, J jp)**: Draws a pie chart that represents the rental distribution b week of rental date. The data for the chart is obtained by calli **create_pie_dataset()** method from the **Query_Rental** class, pass the appropriate SQL query and column names for the number o rented and the week of rental date.
4. **draw_pie_chart_rental_by_rental_day(Charts_Rental frm, J jp)**: Draws a pie chart that represents the rental distribution by th of rental date. The data for the chart is obtained by callin **create_pie_dataset()** method from the **Query_Rental** class, pass the appropriate SQL query and column names for the number o rented and the day of rental date.
5. **draw_pie_chart_rental_by_rental_quarter(Charts_Rental JPanel jp)**: Draws a pie chart that represents the rental distribut the quarter of rental date. The data for the chart is obtained by c the **create_pie_dataset()** method from the **Query_Rental** passing in the appropriate SQL query and column names for the nu of films rented and the quarter of rental date.
6. **draw_bar_chart_rental_by_release_year(Charts_Rental frm, J jp)**: Draws a bar chart that represents the rental distribution b release year of the film. The data for the chart is obtained by calli **create_bar_dataset()** method from the **Query_Rental** class, pass the appropriate SQL query and column names for the number o rented and the release year of the film.

```
    private static void
draw_pie_chart_rental_by_rental_year(Charts_Rental frm, JPanel
jp){
        jp.setPreferredSize(new Dimension(jp.getWidth(),
jp.getHeight()));
        DefaultPieDataset dataset =
create_pie_dataset(Query_Rental.get_sql_rental_year(),
"number_of_films", "year");

        //Draws piechart rental distribution by the year of
rental date
        draw_piechart_with_dataset(frm, jp, dataset, "THE RENTAL
DISTRIBUTION BY YEAR OF RENTAL DATE");
    }

    private static void
draw_pie_chart_rental_by_rental_month(Charts_Rental frm, JPanel
jp){
        jp.setPreferredSize(new Dimension(jp.getWidth(),
jp.getHeight()));
        DefaultPieDataset dataset =
create_pie_dataset(Query_Rental.get_sql_rental_month(),
"number_of_films", "month");

        //Draws piechart rental distribution by the month of
rental date
        draw_piechart_with_dataset(frm, jp, dataset, "THE RENTAL
DISTRIBUTION BY MONTH OF RENTAL DATE");
    }

    private static void
draw_pie_chart_rental_by_rental_week(Charts_Rental frm, JPanel
jp){
        jp.setPreferredSize(new Dimension(jp.getWidth(),
jp.getHeight()));
        DefaultPieDataset dataset =
create_pie_dataset(Query_Rental.get_sql_rental_week(),
"number_of_films", "week");

        //Draws piechart rental distribution by the week of
rental date
        draw_piechart_with_dataset(frm, jp, dataset, "THE TOP 10
RENTAL DISTRIBUTION BY WEEK OF RENTAL DATE");
    }

    private static void
draw_pie_chart_rental_by_rental_day(Charts_Rental frm, JPanel
jp){
        jp.setPreferredSize(new Dimension(jp.getWidth(),
jp.getHeight()));
        DefaultPieDataset dataset =
create_pie_dataset(Query_Rental.get_sql_rental_day(),
"number_of_films", "day");
```

```
54
55          //Draws piechart rental distribution by the day of
56  rental date
57          draw_piechart_with_dataset(frm, jp, dataset, "THE RENTAL
58  DISTRIBUTION BY DAY OF RENTAL DATE");
59      }
60
61      private static void
62  draw_pie_chart_rental_by_rental_quarter(Charts_Rental frm,
63  JPanel jp){
64          jp.setPreferredSize(new Dimension(jp.getWidth(),
65  jp.getHeight()));
66          DefaultPieDataset dataset =
67  create_pie_dataset(Query_Rental.get_sql_rental_quarter(),
68  "number_of_films", "quarter");
69
          //Draws piechart rental distribution by the quarter of
  rental date
          draw_piechart_with_dataset(frm, jp, dataset, "THE RENTAL
  DISTRIBUTION BY QUARTER OF RENTAL DATE");
      }

      private static void
  draw_bar_chart_rental_by_release_year(Charts_Rental frm, JPanel
  jp){
          jp.setPreferredSize(new Dimension(jp.getWidth(),
  jp.getHeight()));

          DefaultCategoryDataset dataset =
  create_bar_dataset(Query_Rental.get_sql_rental_film_year_dist(),
  "number_of_films", "film_year");

          //Draws barchart rental distribution by film release
  year
          draw_barchart_with_dataset(frm, jp, dataset, "THE RENTAL
  DISTRIBUTION BY FILM RELEASE YEAR", "YEAR", "NUMBER");
      }
```

Step 4

In **Rental_Utils** class, define a new method named **jbchart_handler()**.

```
1       public static void
2   jbchart_handler(Charts_Rental frm){
3           //Draws piechart rental distribution
4   by the year of rental date
5
6   draw_pie_chart_rental_by_rental_year(frm,
7   frm.getJPanel1());
8
9           //Draws piechart rental distribution
10  by the month of rental date
11
```

```
12
13 draw_pie_chart_rental_by_rental_month(frm,
14 frm.getJPanel2());
15
16         //Draws piechart rental distribution
17 by the week of rental date
18
19 draw_pie_chart_rental_by_rental_week(frm,
   frm.getJPanel3());

         //Draws piechart rental distribution
   by the day of rental date

   draw_pie_chart_rental_by_rental_day(frm,
   frm.getJPanel4());

         //Draws piechart rental distribution
   by the quarter of rental date

   draw_pie_chart_rental_by_rental_quarter(frm,
   frm.getJPanel5());

         //Draws barchart rental distribution
   by film release year

   draw_bar_chart_rental_by_release_year(frm,
   frm.getJPanel6());
       }
```

Step 5

In **RentalForm**, double click on **jbChart** button to define its event listener

```
1     private void
2 jbChartActionPerformed(java.awt.event.ActionEvent
3 evt) {
4         Charts_Rental frm1 = new Charts_Rental();
5         frm1.setLocationRelativeTo(null);
6         frm1.setTitle("SIX DISTRIBUTIONS IN
7 RENTAL TABLE");
          frm1.setVisible(true);
          Rental_Utils.jbchart_handler(frm1);
     }
```

Step 6

Run the project. Click on CHART button on the form. You will see the six displayed on the panels as shown in Figure 15.13.

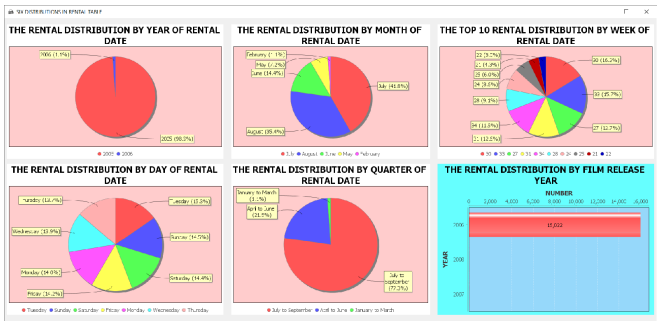


Figure 15.13 The rental distribution by year of rental date, the renta distribution by month of rental date, the 10 rental distribution by week of date, the rental distribution by day of rental date, the rental distributio quarter of rental date, and the rental distribution by film release yea

Step 7 | Create a new **JFrame** and save it as **Charts_Rental2.java**.

Step 8 | In **Charts_Rental2.java**, add six **JPanels** and set their corresponding **Va Name** as **jPanel1**, **jPanel2**, **jPanel3**, **jPanel4**, **jPanel5**, and **jPanel6**. add getter method for each object as follows:

```
1       //Getter method for jPanel1
2       public JPanel getJPanel1(){
3           return this.jPanel1;
4       }
5
6       //Getter method for jPanel2
7       public JPanel getJPanel2(){
8           return this.jPanel2;
9       }
10
11      //Getter method for jPanel3
12      public JPanel getJPanel3(){
13          return this.jPanel3;
14      }
15
16      //Getter method for jPanel4
17      public JPanel getJPanel4(){
18          return this.jPanel4;
19      }
20
21      //Getter method for jPanel5
22      public JPanel getJPanel5(){
23          return this.jPanel5;
24      }
25
26      //Getter method for jPanel6
27      public JPanel getJPanel6(){
28          return this.jPanel6;
```

```
29      }
```

Step 9

In **Rental_Utils** class, define six new methods. Here's a brief summary o each method does:

1. **draw_pie_chart_rental_by_rental_film_duration()**: Draws a pie that shows the distribution of rentals by the duration of the rented
2. **draw_pie_chart_rental_by_rental_film_rating**(): Draws a pie that shows the distribution of rentals by the rating of the rented fil
3. **draw_bar_chart_rental_by_staff_name()**: Draws a bar chart shows the distribution of rentals by the staff member who process rental.
4. **draw_pie_chart_rental_by_language()**: Draws a pie chart that the distribution of rentals by the language of the rented films.
5. **draw_bar_chart_rental_by_film_title()**: Draws a bar chart that the distribution of rentals by the title of the rented films.
6. **draw_bar_chart_rental_by_customer_active()**: Draws a bar cha shows the distribution of rentals by whether the customer who r the film was active or not.

These methods take two arguments: frm, which is an instance c **Charts_Rental2** class, and **jp**, which is a **JPanel** that the chart will be on. The methods then call other methods to create the dataset for the cha actually draw the chart on the panel.

```
1     private static void
2  draw_pie_chart_rental_by_rental_film_duration(Charts_Rental2 frm,
3  JPanel jp){
4          jp.setPreferredSize(new Dimension(jp.getWidth(),
5  jp.getHeight()));
6          DefaultPieDataset dataset =
7  create_pie_dataset(Query_Rental.get_sql_rental_film_duration_dist()
8  "number_of_films", "film_duration");
9
10         //Draws piechart rental distribution by the film duration
11         draw_piechart_with_dataset(frm, jp, dataset, "THE RENTAL
12 DISTRIBUTION BY FILM DURATION");
13     }
14
15     private static void
16 draw_pie_chart_rental_by_rental_film_rating(Charts_Rental2 frm, JPa
17 jp){
18         jp.setPreferredSize(new Dimension(jp.getWidth(),
19 jp.getHeight()));
20         DefaultPieDataset dataset =
21 create_pie_dataset(Query_Rental.get_sql_rental_film_rating_dist(),
22 "number_of_films", "film_rating");
23
24         //Draws piechart rental distribution by the film rating
```

```
        draw_piechart_with_dataset(frm, jp, dataset, "THE RENTAL
DISTRIBUTION BY FILM RATING");
    }

    private static void
draw_bar_chart_rental_by_staff_name(Charts_Rental2 frm, JPanel jp){
        jp.setPreferredSize(new Dimension(jp.getWidth(),
jp.getHeight()));

        DefaultCategoryDataset dataset =
create_bar_dataset(Query_Rental.get_sql_rental_staff_name_dist(),
"number_of_films", "staff_name");

        //Draws barchart rental distribution by staff name
        draw_barchart_with_dataset(frm, jp, dataset, "THE TOP 10
RENTAL DISTRIBUTION BY STAFF NAME", "STAFF NAME", "NUMBER OF RENTED
FILMS");
    }

    private static void
draw_pie_chart_rental_by_language(Charts_Rental2 frm, JPanel jp){
        jp.setPreferredSize(new Dimension(jp.getWidth(),
jp.getHeight()));
        DefaultPieDataset dataset =
create_pie_dataset(Query_Rental.get_sql_rental_film_language_dist()
"number_of_films", "film_language");

        //Draws piechart rental distribution by the film language
        draw_piechart_with_dataset(frm, jp, dataset, "THE RENTAL
DISTRIBUTION BY FILM LANGUAGE");
    }

    private static void
draw_bar_chart_rental_by_film_title(Charts_Rental2 frm, JPanel jp){
        jp.setPreferredSize(new Dimension(jp.getWidth(),
jp.getHeight()));

        DefaultCategoryDataset dataset =
create_bar_dataset(Query_Rental.get_sql_rental_film_title_dist(),
"number_of_films", "film_title");

        //Draws barchart rental distribution by film title
        draw_barchart_with_dataset(frm, jp, dataset, "THE TOP 10
RENTAL DISTRIBUTION BY FILM TITLE", "FILM TITLE", "NUMBER OF RENTED
FILMS");
    }

    private static void
draw_bar_chart_rental_by_customer_active(Charts_Rental2 frm, JPanel
jp){
        jp.setPreferredSize(new Dimension(jp.getWidth(),
jp.getHeight()));
```

```
        DefaultCategoryDataset dataset =
create_bar_dataset(Query_Rental.get_sql_rental_customer_active_dist
"number_of_films", "customer_active");

        //Draws barchart rental distribution by customer active/not
active
        draw_barchart_with_dataset(frm, jp, dataset, "THE RENTAL
DISTRIBUTION BY CUSTOMER ACTIVE", "CUSTOMER ACTIVE/NOT ACTIVE",
"NUMBER OF RENTED FILMS");
    }
```

Step 10

In **Rental_Utils** class, define a new method named **jbchart_handler()**.

```
    public static void
jbchart_handler2(Charts_Rental2 frm){
        //Draws piechart rental distribution by
the film duration

draw_pie_chart_rental_by_rental_film_duration(frm,
frm.getJPanel1());

        //Draws piechart rental distribution by
the film rating

draw_pie_chart_rental_by_rental_film_rating(frm,
frm.getJPanel2());

        //Draws barchart rental distribution by
staff name
        draw_bar_chart_rental_by_staff_name(frm,
frm.getJPanel3());

        //Draws piechart rental distribution by
the film language
        draw_pie_chart_rental_by_language(frm,
frm.getJPanel4());

        //Draws barchart rental distribution by
film title
        draw_bar_chart_rental_by_film_title(frm,
frm.getJPanel5());

        //Draws barchart rental distribution by
customer active/not active

draw_bar_chart_rental_by_customer_active(frm,
frm.getJPanel6());

    }
```

Step 11	In **RentalForm**, double click on **jbChart** button to modify its event liste adding code in line 8 – 12:

```
1     private void
2  jbChartActionPerformed(java.awt.event.ActionEvent
3  evt) {
4          Charts_Rental frm1 = new Charts_Rental();
5          frm1.setLocationRelativeTo(null);
6          frm1.setTitle("SIX DISTRIBUTIONS IN
7  RENTAL TABLE");
8          frm1.setVisible(true);
9          Rental_Utils.jbchart_handler(frm1);
10
11         Charts_Rental2 frm2 = new
12 Charts_Rental2();
13         frm2.setLocationRelativeTo(null);
           frm2.setTitle("SIX DISTRIBUTIONS IN
   RENTAL TABLE");
           frm2.setVisible(true);
           Rental_Utils.jbchart_handler2(frm2);
       }
```

Step 12	Run the project. Click on CHART button on the form. You will see the six displayed on the panels as shown in Figure 15.14.

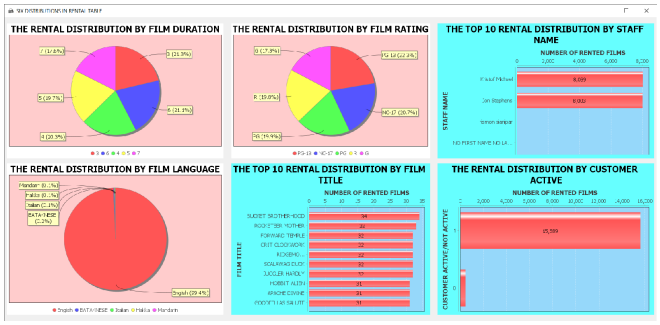


Figure 15.14 The rental distribution by film duration, the rental distribut film rating, the top 10 rental distribution by staff name, the rental distrib by film language, the top 10 rental distribution by film title, and the ren distribution by customer active

Step 13	Create a new **JFrame** and save it as **Charts_Rental3.java**.
Step 14	In **Charts_Rental3.java**, add six **JPanels** and set their corresponding **Va Name** as **jPanel1**, **jPanel2**, **jPanel3**, **jPanel4**, **jPanel5**, and **jPanel6**. add getter method for each object as follows:

```
    //Getter method for jPanel1
    public JPanel getJPanel1(){
        return this.jPanel1;
    }

    //Getter method for jPanel2
    public JPanel getJPanel2(){
        return this.jPanel2;
    }

    //Getter method for jPanel3
    public JPanel getJPanel3(){
        return this.jPanel3;
    }

    //Getter method for jPanel4
    public JPanel getJPanel4(){
        return this.jPanel4;
    }

    //Getter method for jPanel5
    public JPanel getJPanel5(){
        return this.jPanel5;
    }

    //Getter method for jPanel6
    public JPanel getJPanel6(){
        return this.jPanel6;
    }
```

Step 15

In **Rental_Utils** class, define six new methods. These are several method draw bar charts for rental distribution based on different categories category, actor name, customer name, customer city, customer country customer district.

Each method sets the preferred size of the **JPanel** and crea **DefaultCategoryDataset** using the corresponding SQL query **Query_Rental** class. Then, it calls the **draw_barchart_with_dataset()** m with the dataset and the titles of the chart, x-axis, and y-axis to draw th chart.

Overall, these methods provide a way to visualize the rental distribution on different categories and can be useful for analyzing rental pattern trends.

```
    private static void
draw_bar_chart_rental_by_film_category(Charts_Rental3 frm, JPanel j
```

```
        jp.setPreferredSize(new Dimension(jp.getWidth(),
jp.getHeight()));

        DefaultCategoryDataset dataset =
create_bar_dataset(Query_Rental.get_sql_rental_category_dist(),
"number_of_films", "film_category");

        //Draws barchart rental distribution by film category
        draw_barchart_with_dataset(frm, jp, dataset, "THE TOP 10 RE
DISTRIBUTION BY FILM CATEGORY", "FILM CATEGORY", "NUMBER OF RENTED
FILMS");
    }

    private static void
draw_bar_chart_rental_by_actor_name(Charts_Rental3 frm, JPanel jp){
        jp.setPreferredSize(new Dimension(jp.getWidth(),
jp.getHeight()));

        DefaultCategoryDataset dataset =
create_bar_dataset(Query_Rental.get_sql_rental_actor_dist(),
"number_of_films", "actor_name");

        //Draws barchart rental distribution by actor name
        draw_barchart_with_dataset(frm, jp, dataset, "THE TOP 10 RE
DISTRIBUTION BY ACTOR NAME", "ACTOR NAME", "NUMBER OF RENTED FILMS"
    }

    private static void
draw_bar_chart_rental_by_customer_name(Charts_Rental3 frm, JPanel j
        jp.setPreferredSize(new Dimension(jp.getWidth(),
jp.getHeight()));

        DefaultCategoryDataset dataset =
create_bar_dataset(Query_Rental.get_sql_rental_customer_name_dist()
"number_of_films", "customer_name");

        //Draws barchart rental distribution by customer name
        draw_barchart_with_dataset(frm, jp, dataset, "THE TOP 10 RE
DISTRIBUTION BY CUSTOMER NAME", "CUSTOMER NAME", "NUMBER OF RENTED
FILMS");
    }

    private static void
draw_bar_chart_rental_by_customer_city(Charts_Rental3 frm, JPanel j
        jp.setPreferredSize(new Dimension(jp.getWidth(),
jp.getHeight()));

        DefaultCategoryDataset dataset =
create_bar_dataset(Query_Rental.get_sql_rental_customer_city_dist()
"number_of_films", "customer_city");

        //Draws barchart rental distribution by customer city


```

```
        draw_barchart_with_dataset(frm, jp, dataset, "THE TOP 10 RE
DISTRIBUTION BY CUSTOMER CITY", "CUSTOMER CITY", "NUMBER OF RENTED
FILMS");
    }

    private static void
draw_bar_chart_rental_by_customer_country(Charts_Rental3 frm, JPane
{
        jp.setPreferredSize(new Dimension(jp.getWidth(),
jp.getHeight()));

        DefaultCategoryDataset dataset =
create_bar_dataset(Query_Rental.get_sql_rental_customer_country_dis
"number_of_films", "customer_country");

        //Draws barchart rental distribution by customer country
        draw_barchart_with_dataset(frm, jp, dataset, "THE TOP 10 RE
DISTRIBUTION BY CUSTOMER COUNTRY", "CUSTOMER COUNTRY", "NUMBER OF R
FILMS");
    }

    private static void
draw_bar_chart_rental_by_customer_district(Charts_Rental3 frm, JPan
jp){
        jp.setPreferredSize(new Dimension(jp.getWidth(),
jp.getHeight()));

        DefaultCategoryDataset dataset =
create_bar_dataset(Query_Rental.get_sql_rental_customer_district_di
"number_of_films", "customer_district");

        //Draws barchart rental distribution by customer district
        draw_barchart_with_dataset(frm, jp, dataset, "THE TOP 10 RE
DISTRIBUTION BY CUSTOMER DISTRICT", "CUSTOMER DISTRICT", "NUMBER OF
RENTED FILMS");
    }
```

Step 16

In **Rental_Utils** class, define a new method named **jbchart_handler()**.

```
    public static void
jbchart_handler3(Charts_Rental3 frm){
        //Draws piechart rental distribution by
the film category

draw_bar_chart_rental_by_film_category(frm,
frm.getJPanel1());

        //Draws barchart rental distribution by
actor name

draw_bar_chart_rental_by_actor_name(frm,
```

```
frm.getJPanel2());

        //Draws barchart rental distribution by
customer name

draw_bar_chart_rental_by_customer_name(frm,
frm.getJPanel3());

        //Draws barchart rental distribution by
customer city

draw_bar_chart_rental_by_customer_city(frm,
frm.getJPanel4());

        //Draws barchart rental distribution by
customer country

draw_bar_chart_rental_by_customer_country(frm,
frm.getJPanel5());

        //Draws barchart rental distribution by
customer district

draw_bar_chart_rental_by_customer_district(frm,
frm.getJPanel6());
    }
```

Step 17

In **RentalForm**, double click on **jbChart** button to modify its event liste adding code in line 14 – 18:

```
    private void
jbChartActionPerformed(java.awt.event.ActionEvent
evt) {
        Charts_Rental frm1 = new Charts_Rental();
        frm1.setLocationRelativeTo(null);
        frm1.setTitle("SIX DISTRIBUTIONS IN
RENTAL TABLE");
        frm1.setVisible(true);
        Rental_Utils.jbchart_handler(frm1);

        Charts_Rental2 frm2 = new
Charts_Rental2();
        frm2.setLocationRelativeTo(null);
        frm2.setTitle("SIX DISTRIBUTIONS IN
RENTAL TABLE");
        frm2.setVisible(true);
        Rental_Utils.jbchart_handler2(frm2);

        Charts_Rental3 frm3 = new
Charts_Rental3();
        frm3.setLocationRelativeTo(null);
```

```
        frm3.setTitle("SIX DISTRIBUTIONS IN
RENTAL TABLE");
        frm3.setVisible(true);
        Rental_Utils.jbchart_handler3(frm3);
    }
```

Step 18

Run the project. Click on CHART button on the form. You will see the six displayed on the panels as shown in Figure 15.15.

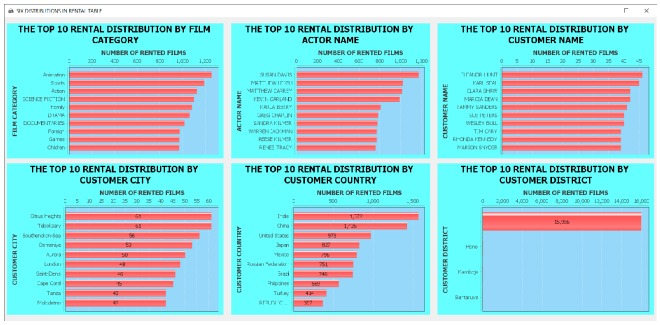


Figure 15.15 The top 10 rental distribution by film category, the top 10 r distribution by actor name, the top 10 rental distribution by customer nam top 10 rental distribution by customer city, the top 10 rental distributio customer country, and the top 10 rental distribution by customer distri

This is the full version of **Rental_Utils.java**:

```
package sakila;
import java.awt.Dimension;
import java.util.logging.Level;
import java.util.logging.Logger;
```

```java
import java.sql.*;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Calendar;
import java.util.HashMap;
import java.util.Objects;
import javax.swing.JComboBox;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.event.TableModelEvent;
import javax.swing.table.DefaultTableModel;
import javax.swing.table.TableModel;
import org.jfree.data.category.DefaultCategoryDataset;
import org.jfree.data.general.DefaultPieDataset;

public class Rental_Utils extends Utility{
    public static final int FIRST_INDEX = 0;
    public static final int INVALID_INDEX = -1;

    private static int currentIndex = FIRST_INDEX;
    private static final String SQL_ID = Query_Rental.get_sql_id();

    //Creates rental table
    public static void create_rental_table() {
        try (Connection conn = getConnection()) {
            Statement stmt = conn.createStatement();
            stmt.addBatch(Query_Rental.get_sql_rental());
            stmt.executeBatch();

            String message = String.format("Successfully creates rental table
            JOptionPane.showMessageDialog(null, message,
            "INFORMATION",JOptionPane.INFORMATION_MESSAGE);

        } catch (SQLException ex) {
            JOptionPane.showMessageDialog(null, ex.getMessage(),
                "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    //Populates rental table with some rows of data
    public static void populate_rental_table(){
        try(Connection conn = getConnection()){
            String sql = """
                INSERT INTO rental(rental_id, rental_date, inventory_id, cust
                    return_date, staff_id, last_update)
                VALUES(?, ?, ?, ?, ?, ?, ?)""";

            //Creates a new Rental class with default constructor
            PreparedStatement ps1 = conn.prepareStatement(sql);
            Rental obj1 = new Rental();
            ps1.setInt(1,obj1.getRentalID());
```

```
            ps1.setDate(2,obj1.getRentalDate());
            ps1.setInt(3,obj1.getInventoryID());
            ps1.setInt(4,obj1.getCustomerID());
            ps1.setDate(5,obj1.getReturnDate());
            ps1.setInt(6,obj1.getStaffID());
            ps1.setTimestamp(7,obj1.getLastUpdate());

            // Creates a new Rental class with nine-params constructor
            PreparedStatement ps2 = conn.prepareStatement(sql);
            SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd");
            String str_rental_date = "2023-05-03";
            java.util.Date rent_date = dateFormat.parse(str_rental_date);
            java.sql.Date rental_sql_date = new java.sql.Date(rent_date.getT

            String str_ret_date = "2023-05-19";
            java.util.Date ret_date = dateFormat.parse(str_ret_date);
            java.sql.Date return_sql_date = new java.sql.Date(ret_date.getTi

            Rental obj2 = new Rental(2, rental_sql_date, 2, 2, return_sql_da
Timestamp(System.currentTimeMillis()));
            ps2.setInt(1,obj2.getRentalID());
            ps2.setDate(2,obj2.getRentalDate());
            ps2.setInt(3,obj2.getInventoryID());
            ps2.setInt(4,obj2.getCustomerID());
            ps2.setDate(5,obj2.getReturnDate());
            ps2.setInt(6,obj2.getStaffID());
            ps2.setTimestamp(7,obj2.getLastUpdate());

            ps1.executeUpdate();
            ps2.executeUpdate();

        }catch(SQLException ex){
            JOptionPane.showMessageDialog(null, ex.getMessage(),
                "ERROR",JOptionPane.ERROR_MESSAGE);
        } catch (ParseException ex) {
            Logger.getLogger(Rental_Utils.class.getName()).log(Level.SEVERE,
ex);
        }
    }

    //Reads the content of rental table
    public static void read_rental_table(){
        try(Connection conn = getConnection()){
            Statement stmt = conn.createStatement();
            ResultSet rs = stmt.executeQuery("SELECT * FROM rental");

            while(rs.next()){
                int rental_id = rs.getInt("rental_id");
                Date rent_date = rs.getDate("rental_date");
                int inv_id = rs.getInt("inventory_id");
                int cust_id = rs.getInt("customer_id");
                Date ret_date = rs.getDate("return_date");
                int staff_id = rs.getInt("staff_id");
```

```
                Timestamp lu = rs.getTimestamp("last_update");

                //Creates a Rental object using seven-params constructor
                Rental obj = new Rental(rental_id, rent_date, inv_id, cust_id
ret_date, staff_id, lu);
                System.out.println(obj);
            }
            rs.close();
            stmt.close();

        }catch(SQLException ex){
            JOptionPane.showMessageDialog(null, ex.getMessage(),
                "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    private static ArrayList<Rental> get_rental_list(RentalForm frm, String s
String item){
        ArrayList<Rental> list = new ArrayList<>();

        try(Connection conn = getConnection();
            PreparedStatement ps = conn.prepareStatement(sql)){
            if (item.equalsIgnoreCase("none")==false) {
                 ps.setString(1,item);
            }
            ResultSet rs = ps.executeQuery();
            Rental obj;

            while(rs.next()){
                //Using twenty-eight-params constructor
                obj = new Rental(rs.getInt("rental_id"),
                        rs.getDate("rental_date"),
                        rs.getInt("inventory_id"),
                        rs.getInt("customer_id"),
                        rs.getDate("return_date"),
                        rs.getInt("staff_id"),
                        rs.getTimestamp("last_update"),
                        rs.getInt("film_id"),
                        rs.getString("title"),
                        rs.getString("description"),
                        rs.getInt("release_year"),
                        rs.getInt("rental_duration"),
                        rs.getDouble("rental_rate"),
                        rs.getInt("length"),
                        rs.getDouble("replacement_cost"),
                        rs.getString("rating"),
                        rs.getString("special_features"),
                        rs.getString("language_name"),
                        rs.getString("category_name"),
                        rs.getString("actor_name"),
                        rs.getString("customer_name"),
                        rs.getString("customer_email"),
                        rs.getString("staff_name"),
```

```
                        rs.getString("staff_email"),
                        rs.getString("store_city"),
                        rs.getString("store_country"),
                        rs.getString("store_district"),
                        rs.getString("store_postal_code"));

                list.add(obj);
            }
        }catch (SQLException ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
                "ERROR",JOptionPane.ERROR_MESSAGE);
        }
        return list;
    }

    private static void show_table_rental(RentalForm frm, ArrayList<Rental> l
throws SQLException{
        DefaultTableModel model = new DefaultTableModel(0,0);

        String header[] = {"Rental ID", "Rental Date", "Inventory ID", "Custo
"Return Date",
            "Staff ID", "Film ID", "Title", "Description", "Release Year",
            "Rental Duration", "Rental Rate", "Length", "Replacement Cost",
            "Rating", "Special Features", "Language", "Category", "Actor Name
            "Customer Name", "Customer Email", "Staff Name", "Staff Email",
            "Store City", "Store Country", "Store District", "Store Postal Co
            "Last Update"};

        model.setColumnIdentifiers(set_column_header(frm.getJTRental(), heade
        frm.getJTRental().setModel(model);

        Object[] row = new Object[28];

        for(int i=0; i<list.size(); i++){
            row[0] = list.get(i).getRentalID();
            row[1] = list.get(i).getRentalRate();
            row[2] = list.get(i).getInventoryID();
            row[3] = list.get(i).getCustomerID();
            row[4] = list.get(i).getReturnDate();
            row[5] = list.get(i).getStaffID();
            row[6] = list.get(i).getFilmID();
            row[7] = list.get(i).getTitle();
            row[8] = list.get(i).getDescription();
            row[9] = list.get(i).getReleaseYear();
            row[10] = list.get(i).getRentalDuration();
            row[11] = list.get(i).getRentalRate();
            row[12] = list.get(i).getLength();
            row[13] = list.get(i).getReplacementCost();
            row[14] = list.get(i).getRating();
            row[15] = list.get(i).getSpecialFeatures();
            row[16] = list.get(i).getLanguageName();
            row[17] = list.get(i).getCategory();
            row[18] = list.get(i).getActorName();
            row[19] = list.get(i).getCustomerName();
```

```
            row[20] = list.get(i).getCustomerEmail();
            row[21] = list.get(i).getStaffName();
            row[22] = list.get(i).getStaffEmail();
            row[23] = list.get(i).getStoreCity();
            row[24] = list.get(i).getStoreCountry();
            row[25] = list.get(i).getStoreDistrict();
            row[26] = list.get(i).getStorePostalCode();
            row[27] = list.get(i).getLastUpdate();

            model.addRow(row);
        }
    }

    public static void refresh_controls(RentalForm frm){
        frm.setLocationRelativeTo(null);
        frm.setTitle("RENTAL FORM");

        //Shows the content of rental table and populates combobox
        try{
            //Makes alternating color for table rows
            table_renderer(frm.getJTRental());

            //Populates table
            ArrayList<Rental> list = get_rental_list(frm,
Query_Rental.get_sql_rental_joint() + " ORDER BY rental_id", "none");
            show_table_rental(frm, list);

            //Populates jcbRentalID
            String sql_rent_id = "SELECT rental_id FROM rental ORDER BY renta
            populate_combobox(sql_rent_id, frm.getJCBRentalID(), frm);

            //Populates jcbInventoryID
            String sql_inv_id = "SELECT inventory_id FROM inventory ORDER BY
inventory_id";
            populate_combobox(sql_inv_id, frm.getJCBInventoryID(), frm);

            //Populates jcbCustomerID
            String sql_cust_id = "SELECT customer_id FROM customer ORDER BY
customer_id";
            populate_combobox(sql_cust_id, frm.getJCBCustomerID(), frm);

            //Populates jcbStaffID
            String sql_stf_id = "SELECT staff_id FROM staff ORDER BY staff_id
            populate_combobox(sql_stf_id, frm.getJCBStaffID(), frm);

        }catch (SQLException ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
            "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    private static void clear_staff_controls(RentalForm frm){
        frm.getJTFStaffID().setText("");
```

```
        frm.getJTFStaffName().setText("");
        frm.getJTFStaffEmail().setText("");
    }

   //Displays staff data result row by row
    private static <T> void display_staff_data(RentalForm frm, String sql, T
        try(Connection conn = getConnection()){
            PreparedStatement ps = conn.prepareStatement(sql);
            ps.setObject(1,item);
            ResultSet rs = ps.executeQuery();

            if (!rs.next()) {
                // no row found, clear the form fields
                clear_staff_controls(frm);
                return;
            }

            do{
                frm.getJTFStaffID().setText(String.valueOf(rs.getInt("staff_
                frm.getJTFStaffName().setText(rs.getString("first_name") + "
rs.getString("last_name"));
                frm.getJTFStaffEmail().setText(rs.getString("email"));

                // Determines item selected from jcbStaffID
                find_combo_value_selected(frm.getJCBStaffID(), rs.getInt("sta

            }while(rs.next());

            rs.close();
            ps.close();
        }catch(SQLException ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    public static void jcbStaff_handler(RentalForm frm) {
        Object item = frm.getJCBStaffID().getSelectedItem();
        display_staff_data(frm, Query_Staff.get_sql_id(), item);
    }

    private static void clear_customer_controls(RentalForm frm){
        frm.getJTFCustomerID().setText("");
        frm.getJTFCustomerName().setText("");
        frm.getJTFCustEmail().setText("");
    }

   //Displays customer data result row by row
    private static <T> void display_customer_data(RentalForm frm, String sql,
        try(Connection conn = getConnection()){
            PreparedStatement ps = conn.prepareStatement(sql);
            ps.setObject(1,item);
            ResultSet rs = ps.executeQuery();
```

```
            if (!rs.next()) {
                // no row found, clear the form fields
                clear_customer_controls(frm);
                return;
            }

            do{

frm.getJTFCustomerID().setText(String.valueOf(rs.getInt("customer_id")));
                frm.getJTFCustomerName().setText(rs.getString("first_name") +
rs.getString("last_name"));
                frm.getJTFCustEmail().setText(rs.getString("email"));

                // Determines item selected from jcbCustomerID
                find_combo_value_selected(frm.getJCBCustomerID(),
rs.getInt("customer_id"));

            }while(rs.next());

            rs.close();
            ps.close();
        }catch(SQLException ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    public static void jcbCustomer_handler(RentalForm frm) {
        Object item = frm.getJCBCustomerID().getSelectedItem();
        display_customer_data(frm, Query_Customer.get_sql_id(), item);
    }

    private static void clear_inventory_controls(RentalForm frm){
        frm.getJTFInventoryID().setText("");
        frm.getJTFFilmID().setText("");
        frm.getJTFStoreID().setText("");
        frm.getJTFAddressID().setText("");
        frm.getJTFAddress().setText("");
        frm.getJTFTitle().setText("");
        frm.getJTFDescription().setText("");
        frm.getJTFReleaseYear().setText("");
        frm.getJTFRentalDuration().setText("");
        frm.getJTFRentalRate().setText("");
        frm.getJTFLength().setText("");
        frm.getJTFReplacementCost().setText("");
        frm.getJTFSpecialFeatures().setText("");
        frm.getJTFCategory().setText("");
        frm.getJTFLanguageName().setText("");
        frm.getJTFActorName().setText("");
        frm.getJTFDistrict().setText("");
        frm.getJTFPhone().setText("");
        frm.getJTFPostalCode().setText("");
```

```
        frm.getJTFCity().setText("");
        frm.getJTFCountry().setText("");
    }
   //Displays inventory data result row by row
    private static <T> void display_inventory_data(RentalForm frm, String sql
{
        try(Connection conn = getConnection()){
            PreparedStatement ps = conn.prepareStatement(sql);
            ps.setObject(1,item);
            ResultSet rs = ps.executeQuery();

            if (!rs.next()) {
                // no row found, clear the form fields
                clear_inventory_controls(frm);
                return;
            }

            do{

frm.getJTFInventoryID().setText(String.valueOf(rs.getInt("inventory_id")));
                frm.getJTFFilmID().setText(String.valueOf(rs.getInt("film_id"
                frm.getJTFStoreID().setText(String.valueOf(rs.getInt("store_i

frm.getJTFAddressID().setText(String.valueOf(rs.getInt("address_id")));
                frm.getJTFAddress().setText(rs.getString("store_address"));
                frm.getJTFDistrict().setText(rs.getString("store_district"));
                frm.getJTFCity().setText(rs.getString("store_city"));
                frm.getJTFCountry().setText(rs.getString("store_country"));
                frm.getJTFPhone().setText(rs.getString("store_phone"));
                frm.getJTFPostalCode().setText(rs.getString("store_postal_cod
                frm.getJTFTitle().setText(rs.getString("title"));
                frm.getJTFDescription().setText(rs.getString("description"));

frm.getJTFReleaseYear().setText(String.valueOf(rs.getInt("release_year")));

frm.getJTFRentalDuration().setText(String.valueOf(rs.getInt("rental_duration"

frm.getJTFRentalRate().setText(String.valueOf(rs.getDouble("rental_rate")));
                frm.getJTFLength().setText(String.valueOf(rs.getInt("length")

frm.getJTFReplacementCost().setText(String.valueOf(rs.getDouble("replacement_
                frm.getJTFSpecialFeatures().setText(rs.getString("special_fea
                frm.getJTFRating().setText(rs.getString("rating"));
                frm.getJTFLanguageName().setText(rs.getString("language_name"
                frm.getJTFActorName().setText(rs.getString("actor_name"));
                frm.getJTFCategory().setText(rs.getString("category_name"));

                // Determines item selected from jcbInventoryID
                find_combo_value_selected(frm.getJCBInventoryID(),
rs.getInt("inventory_id"));

            }while(rs.next());
```

```
            rs.close();
            ps.close();
        }catch(SQLException ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    public static void jcbInventory_handler(RentalForm frm) {
        Object item = frm.getJCBInventoryID().getSelectedItem();
        display_inventory_data(frm, Query_Rental.get_sql_inventory_joint() +
inventory_id = ?", item);
    }

   //Displays rental data result row by row
    private static <T> void display_rental_data(RentalForm frm, String sql, T
        try(Connection conn = getConnection()){
            PreparedStatement ps = conn.prepareStatement(sql);
            ps.setObject(1,item);
            ResultSet rs = ps.executeQuery();

            if (!rs.next()) {
                // no row found, clear the form fields
                frm.getJTFRentalID().setText("");
                frm.getJTFLastUpdate().setText("");
                return;
            }

            do{
                frm.getJTFRentalID().setText(String.valueOf(rs.getInt("rental
                frm.getJTFLastUpdate().setText(rs.getString("last_update"));

                java.util.Date rentaldate = new
Date(rs.getTimestamp("rental_date").getTime());
                frm.getJCRentalDate().setDate(rentaldate);

                java.util.Date retdate = new
Date(rs.getTimestamp("return_date").getTime());
                frm.getJCReturnDate().setDate(retdate);

                // Determines item selected from jcbRentalID
                find_combo_value_selected(frm.getJCBRentalID(),
rs.getInt("rental_id"));

                // Determines item selected from jcbInventoryID
                find_combo_value_selected(frm.getJCBInventoryID(),
rs.getInt("inventory_id"));

                // Determines item selected from jcbStaffID
                find_combo_value_selected(frm.getJCBStaffID(), rs.getInt("sta

                // Determines item selected from jcbCustomerID
```

```
                find_combo_value_selected(frm.getJCBCustomerID(),
rs.getInt("customer_id"));

            }while(rs.next());

            rs.close();
            ps.close();
        }catch(SQLException ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    public static void jcbRental_handler(RentalForm frm) {
        Object item = frm.getJCBRentalID().getSelectedItem();
        display_rental_data(frm, Query_Rental.get_sql_id(), item);
    }

    public static void show_first_row(RentalForm frm){
        String item = String.valueOf(frm.getJCBRentalID().getItemAt(FIRST_INI
        display_rental_data(frm, SQL_ID, item);
        currentIndex = FIRST_INDEX;
    }

    public static void show_last_row(RentalForm frm){
        int endIndex = frm.getJCBRentalID().getItemCount() - 1;
        String item = String.valueOf(frm.getJCBRentalID().getItemAt(endIndex)
        display_rental_data(frm, SQL_ID, item);
        currentIndex = endIndex;
    }

    public static void show_prev_row(RentalForm frm){
        currentIndex--;
        if(currentIndex < FIRST_INDEX){
            currentIndex = FIRST_INDEX;
            return;
        }
        String item = String.valueOf(frm.getJCBRentalID().getItemAt(currentIr
        display_rental_data(frm, SQL_ID, item);
    }

    public static void show_next_row(RentalForm frm){
        int endIndex = frm.getJCBRentalID().getItemCount() - 1;
        currentIndex++;
        if(currentIndex > endIndex){
            currentIndex = endIndex;
            return;
        }
        String item = String.valueOf(frm.getJCBRentalID().getItemAt(currentIr
        display_rental_data(frm, SQL_ID, item);
    }

    public static void mouse_pressed_handler(RentalForm frm) {
```

```
        Objects.requireNonNull(frm, "frm must not be null");

        int selectedIndex = frm.getJTRental().getSelectedRow();
        if (selectedIndex == -1) {
            JOptionPane.showMessageDialog(frm, "Please select a row to view :
data.",
                "No row selected", JOptionPane.INFORMATION_MESSAGE);
            return;
        }

        try (Connection conn = getConnection()) {
            String id =
String.valueOf(frm.getJTRental().getModel().getValueAt(selectedIndex, 0));

            // Displays rental data
            display_rental_data(frm, SQL_ID, id);

        } catch (SQLException ex) {
            Logger.getLogger(RentalForm.class.getName()).log(Level.SEVERE, "E
displaying rental data", ex);
            String message = "Error displaying rental data: " + ex.getMessage
            String stackTrace = Arrays.toString(ex.getStackTrace());
            JOptionPane.showMessageDialog(frm, message + "\n\n" + stackTrace,
JOptionPane.ERROR_MESSAGE);
        }
    }

    //Updates row of data in rental tabel by rental_id
    public static void update_row_by_rental_id(RentalForm frm, int rental_id,
            Date rent_date, int inv_id, int cust_id, Date ret_date, int staff
throws SQLException{
        Connection conn = getConnection();
        ResultSet rs = null;
        String query_id = "SELECT rental_id FROM rental WHERE rental_id = ?";
        String update_query = """
            UPDATE rental SET rental_date = ?, inventory_id = ?, customer_id
            return_date = ?, staff_id = ? WHERE rental_id = ?""";
        try(PreparedStatement idPs = conn.prepareStatement(query_id,
                    ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_READ_ONL
        PreparedStatement updatePS = conn.prepareStatement(update_query,
                ResultSet.TYPE_SCROLL_SENSITIVE, ResultSet.CONCUR_UPDATABLE))
        {
            idPs.setInt(1,rental_id);
            if(!idPs.execute()){
                String message = "Can't find rental_id " + rental_id;

                JOptionPane.showMessageDialog(frm, message,
                    "ERROR",JOptionPane.ERROR_MESSAGE);
            } else{
                rs = idPs.getResultSet();
                rs.next();

                //Creates a Rental object using seven-params constructor
```

```
                Rental obj = new Rental(rental_id, rent_date, inv_id, cust_id
ret_date, staff_id, new Timestamp(System.currentTimeMillis()));
                updatePS.setDate(1, obj.getRentalDate());
                updatePS.setInt(2, obj.getInventoryID());
                updatePS.setInt(3, obj.getCustomerID());
                updatePS.setDate(4, obj.getReturnDate());
                updatePS.setInt(5, obj.getStaffID());
                updatePS.setInt(6, obj.getRentalID());

                updatePS.executeUpdate();
                rs.close();
                updatePS.close();
                idPs.close();
                conn.close();
            }
        }catch(SQLException ex){
            Logger.getLogger(RentalForm.class.getName()).log(Level.SEVERE, "E
updating rental data", ex);
            String message = "Error updating rental data: " + ex.getMessage()
            String stackTrace = Arrays.toString(ex.getStackTrace());
            JOptionPane.showMessageDialog(null, message + "\n\n" + stackTrace
"ERROR", JOptionPane.ERROR_MESSAGE);
        }catch(java.lang.NumberFormatException ex){
            Logger.getLogger(RentalForm.class.getName()).log(Level.SEVERE, "I
Input", ex);
            String message = "Invalid Input: " + ex.getMessage();
            String stackTrace = Arrays.toString(ex.getStackTrace());
            JOptionPane.showMessageDialog(null, message + "\n\n" + stackTrace
"ERROR", JOptionPane.ERROR_MESSAGE);
        }
    }

    private static HashMap<String, String> read_inputs(RentalForm frm) {
        HashMap<String, String> input_data = new HashMap<>();
        String cust_id = String.valueOf(frm.getJCBCustomerID().getSelectedIte
        String rental_id = String.valueOf(frm.getJCBRentalID().getSelectedIte
        String staff_id = String.valueOf(frm.getJCBStaffID().getSelectedItem(
        String inv_id = String.valueOf(frm.getJCBInventoryID().getSelectedIte

        //Gets the selected date from the getJCRentalDate()
        java.util.Date rental_date = frm.getJCRentalDate().getDate();
        //Create a SimpleDateFormat object with the desired date format
        SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");
        //Formats the date as a String using the SimpleDateFormat object
        String rent_date = sdf.format(rental_date);

        //Gets the selected date from the getJCRentalDate()
        java.util.Date return_date = frm.getJCReturnDate().getDate();
        //Formats the date as a String using the SimpleDateFormat object
        String ret_date = sdf.format(return_date);

        // Validate user input
        int rental_id_int = 0;
        try {
```

```
            rental_id_int = Integer.parseInt(rental_id);
            if (rental_id_int <= 0) {
                throw new IllegalArgumentException("Rental ID cannot be negat
zero");
            }
        } catch (NumberFormatException ex) {
            JOptionPane.showMessageDialog(frm, "Invalid Rental ID: " + rental
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        } catch (IllegalArgumentException ex) {
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        }

        int staff_id_int = 0;
        try {
            staff_id_int = Integer.parseInt(staff_id);
            if (staff_id_int <= 0) {
                throw new IllegalArgumentException("Staff ID cannot be negati
zero");
            }
        } catch (NumberFormatException ex) {
            JOptionPane.showMessageDialog(frm, "Invalid Staff ID: " + staff_i
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        } catch (IllegalArgumentException ex) {
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        }

        int cust_id_int = 0;
        try {
            cust_id_int = Integer.parseInt(cust_id);
            if (cust_id_int <= 0) {
                throw new IllegalArgumentException("Customer ID cannot be neg
zero");
            }
        } catch (NumberFormatException ex) {
            JOptionPane.showMessageDialog(frm, "Invalid Manager Customer ID:
cust_id,
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        } catch (IllegalArgumentException ex) {
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        }

        int inv_id_int = 0;
        try {
            inv_id_int = Integer.parseInt(inv_id);
```

```
            if (inv_id_int <= 0) {
                throw new IllegalArgumentException("Inventory ID cannot be ne
zero");
            }
        } catch (NumberFormatException ex) {
            JOptionPane.showMessageDialog(frm, "Invalid Manager Inventory ID
inv_id,
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        } catch (IllegalArgumentException ex) {
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        }

        input_data.put("cust_id", cust_id);
        input_data.put("inv_id", inv_id);
        input_data.put("staff_id", staff_id);
        input_data.put("rental_id", rental_id);
        input_data.put("rent_date", rent_date);
        input_data.put("ret_date", ret_date);

        return input_data;
    }

    private static void edit_actual(RentalForm frm) throws ParseException{
        try{
            HashMap<String, String> input_data = read_inputs(frm);
            int cust_id = Integer.parseInt(input_data.get("cust_id"));
            int inv_id = Integer.parseInt(input_data.get("inv_id"));
            int staff_id = Integer.parseInt(input_data.get("staff_id"));
            int rental_id = Integer.parseInt(input_data.get("rental_id"));
            String str_rental_date = input_data.get("rent_date");
            String str_ret_date = input_data.get("ret_date");

            SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd");
            java.util.Date rent_date = dateFormat.parse(str_rental_date);
            java.sql.Date rental_sql_date = new java.sql.Date(rent_date.getT

            java.util.Date ret_date = dateFormat.parse(str_ret_date);
            java.sql.Date return_sql_date = new java.sql.Date(ret_date.getTi

            update_row_by_rental_id(frm, rental_id, rental_sql_date, inv_id,
return_sql_date, staff_id);

            //Refreshes all objects on form
            refresh_controls(frm);

        }catch(SQLException ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }
```

```
    private static void enable_controls(boolean state, RentalForm frm){
        frm.getJBFirst().setEnabled(state);
        frm.getJBPrev().setEnabled(state);
        frm.getJBNext().setEnabled(state);
        frm.getJBLast().setEnabled(state);
        frm.getJBInsert().setEnabled(state);
        frm.getJBDelete().setEnabled(state);
        frm.getJTFAddressID().setEnabled(state);
        frm.getJTFStoreID().setEnabled(state);
        frm.getJTFAddress().setEnabled(state);
        frm.getJTFDistrict().setEnabled(state);
        frm.getJTFPhone().setEnabled(state);
        frm.getJTFPostalCode().setEnabled(state);
        frm.getJTFCity().setEnabled(state);
        frm.getJTFCountry().setEnabled(state);
        frm.getJTFStaffID().setEnabled(state);
        frm.getJTFStaffName().setEnabled(state);
        frm.getJTFStaffEmail().setEnabled(state);
        frm.getJTFLastUpdate().setEnabled(state);
        frm.getJTFFilmID().setEnabled(state);
        frm.getJTFCustEmail().setEnabled(state);
        frm.getJTFCustomerName().setEnabled(state);
        frm.getJTFTitle().setEnabled(state);
        frm.getJTFDescription().setEnabled(state);
        frm.getJTFReleaseYear().setEnabled(state);
        frm.getJTFRentalDuration().setEnabled(state);
        frm.getJTFRentalRate().setEnabled(state);
        frm.getJTFLength().setEnabled(state);
        frm.getJTFReplacementCost().setEnabled(state);
        frm.getJTFSpecialFeatures().setEnabled(state);
        frm.getJTFCategory().setEnabled(state);
        frm.getJTFLanguageName().setEnabled(state);
        frm.getJTFActorName().setEnabled(state);
    }

    public static void edit_handler(RentalForm frm){
        if(frm.getJBEdit().getText().equals("EDIT")){
            frm.getJBEdit().setText("CONFIRM");

            // Disables controls
            enable_controls(false, frm);
        }

        else {
            try {
                frm.getJBEdit().setText("EDIT");

                // Actual editing
                edit_actual(frm);

                //Enables controls
                enable_controls(true, frm);
```

```
            } catch (ParseException ex) {
                Logger.getLogger(Rental_Utils.class.getName()).log(Level.SEVE
ex);
            }
        }
    }

    //Inserts new row into rental table
    private static void insert_row(RentalForm frm) throws SQLException,
ParseException{
        HashMap<String, String> input_data = read_inputs(frm);
        int cust_id = Integer.parseInt(input_data.get("cust_id"));
        int inv_id = Integer.parseInt(input_data.get("inv_id"));
        int staff_id = Integer.parseInt(input_data.get("staff_id"));
        String str_rental_date = input_data.get("rent_date");
        String str_ret_date = input_data.get("ret_date");

        SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd");
        java.util.Date rent_date = dateFormat.parse(str_rental_date);
        java.sql.Date rental_sql_date = new java.sql.Date(rent_date.getTime()

        java.util.Date ret_date = dateFormat.parse(str_ret_date);
        java.sql.Date return_sql_date = new java.sql.Date(ret_date.getTime()

        // SQL insert statement
        String sql = """
            INSERT INTO rental(rental_date, inventory_id, customer_id,
                return_date, staff_id) VALUES(?, ?, ?, ?, ?)""";

        try(Connection conn = getConnection();
         PreparedStatement pstmt = conn.prepareStatement(sql)){

            //Creates a Rental object six-params constructor
            Rental obj = new Rental(rental_sql_date, inv_id, cust_id, return_
staff_id, new Timestamp(System.currentTimeMillis()));
            pstmt.setDate(1,obj.getRentalDate());
            pstmt.setInt(2,obj.getInventoryID());
            pstmt.setInt(3,obj.getCustomerID());
            pstmt.setDate(4,obj.getReturnDate());
            pstmt.setInt(5, obj.getStaffID());

            //Executes the sql insert statement
            pstmt.executeUpdate();
        } catch (SQLException ex) {
            Logger.getLogger(RentalForm.class.getName()).log(Level.SEVERE, "D
error", ex);
            JOptionPane.showMessageDialog(frm, "Error: Database error\n" +
ex.getMessage());
        }
    }

    private static void insert_actual(RentalForm frm) throws ParseException{
        try{
```

```
            insert_row(frm);

            //Refreshes table and comboboxes
            refresh_controls(frm);

        }catch(SQLException ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    public static void insert_handler(RentalForm frm){
        if(frm.getJBInsert().getText().equals("INSERT")){
            frm.getJBInsert().setText("CONFIRM");

            //Disables jbEdit
            frm.getJBEdit().setEnabled(false);

            // Disables controls
            enable_controls(false, frm);
            frm.getJCBRentalID().setEnabled(false);

            // Clears controls
            clear_customer_controls(frm);

            // Enables
            frm.getJBInsert().setEnabled(true);
        }

        else {
            frm.getJBInsert().setText("INSERT");

            try {
                // Actual insertion
                insert_actual(frm);
            } catch (ParseException ex) {
                Logger.getLogger(Rental_Utils.class.getName()).log(Level.SEVE
ex);
            }

            //Enables jbEdit
            frm.getJBEdit().setEnabled(true);

            //Enables controls
            enable_controls(true, frm);
            frm.getJCBRentalID().setEnabled(true);
        }
    }

    public static void delete_handler(RentalForm frm){
        int dialogButton = JOptionPane.YES_NO_OPTION;
        int rental_id =
Integer.parseInt(String.valueOf(frm.getJCBRentalID().getSelectedItem()));
```

```
        String message = String.format("Are you sure you want to delete the r
ID: %d)", rental_id);
        int answer = JOptionPane.showConfirmDialog(frm, message, "DELETING RO
DATA", dialogButton);

        if(answer == JOptionPane.YES_OPTION){
            String query = """
                DELETE FROM rental WHERE rental_id = ?""";
            try(Connection conn = getConnection();
                PreparedStatement ps = conn.prepareStatement(query)){
                // Use PreparedStatement to avoid SQL injection attacks
                ps.setInt(1, rental_id);
                ps.executeUpdate();

                // Refresh table and comboboxes
                refresh_controls(frm);

            } catch (SQLException ex){
                JOptionPane.showMessageDialog(frm, ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
            }
        }
    }

    private static void draw_pie_chart_rental_by_rental_year(Charts_Rental fr
jp){
        jp.setPreferredSize(new Dimension(jp.getWidth(), jp.getHeight()));
        DefaultPieDataset dataset =
create_pie_dataset(Query_Rental.get_sql_rental_year(), "number_of_films", "ye

        //Draws piechart rental distribution by the year of rental date
        draw_piechart_with_dataset(frm, jp, dataset, "THE RENTAL DISTRIBUTION
OF RENTAL DATE");
    }

    private static void draw_pie_chart_rental_by_rental_month(Charts_Rental f
JPanel jp){
        jp.setPreferredSize(new Dimension(jp.getWidth(), jp.getHeight()));
        DefaultPieDataset dataset =
create_pie_dataset(Query_Rental.get_sql_rental_month(), "number_of_films", "r

        //Draws piechart rental distribution by the month of rental date
        draw_piechart_with_dataset(frm, jp, dataset, "THE RENTAL DISTRIBUTION
OF RENTAL DATE");
    }

    private static void draw_pie_chart_rental_by_rental_week(Charts_Rental fr
jp){
        jp.setPreferredSize(new Dimension(jp.getWidth(), jp.getHeight()));
        DefaultPieDataset dataset =
create_pie_dataset(Query_Rental.get_sql_rental_week(), "number_of_films", "we
```

```
        //Draws piechart rental distribution by the week of rental date
        draw_piechart_with_dataset(frm, jp, dataset, "THE TOP 10 RENTAL DISTR
BY WEEK OF RENTAL DATE");
    }

    private static void draw_pie_chart_rental_by_rental_day(Charts_Rental frm
jp){
        jp.setPreferredSize(new Dimension(jp.getWidth(), jp.getHeight()));
        DefaultPieDataset dataset =
create_pie_dataset(Query_Rental.get_sql_rental_day(), "number_of_films", "day

        //Draws piechart rental distribution by the day of rental date
        draw_piechart_with_dataset(frm, jp, dataset, "THE RENTAL DISTRIBUTION
OF RENTAL DATE");
    }

    private static void draw_pie_chart_rental_by_rental_quarter(Charts_Rental
JPanel jp){
        jp.setPreferredSize(new Dimension(jp.getWidth(), jp.getHeight()));
        DefaultPieDataset dataset =
create_pie_dataset(Query_Rental.get_sql_rental_quarter(), "number_of_films",
"quarter");

        //Draws piechart rental distribution by the quarter of rental date
        draw_piechart_with_dataset(frm, jp, dataset, "THE RENTAL DISTRIBUTION
QUARTER OF RENTAL DATE");
    }

    private static void draw_bar_chart_rental_by_release_year(Charts_Rental f
JPanel jp){
        jp.setPreferredSize(new Dimension(jp.getWidth(), jp.getHeight()));

        DefaultCategoryDataset dataset =
create_bar_dataset(Query_Rental.get_sql_rental_film_year_dist(), "number_of_f
"film_year");

        //Draws barchart rental distribution by film release year
        draw_barchart_with_dataset(frm, jp, dataset, "THE RENTAL DISTRIBUTION
RELEASE YEAR", "YEAR", "NUMBER");
    }

    public static void jbchart_handler(Charts_Rental frm){
        //Draws piechart rental distribution by the year of rental date
        draw_pie_chart_rental_by_rental_year(frm, frm.getJPanel1());

        //Draws piechart rental distribution by the month of rental date
        draw_pie_chart_rental_by_rental_month(frm, frm.getJPanel2());

        //Draws piechart rental distribution by the week of rental date
        draw_pie_chart_rental_by_rental_week(frm, frm.getJPanel3());

        //Draws piechart rental distribution by the day of rental date
        draw_pie_chart_rental_by_rental_day(frm, frm.getJPanel4());
```

```
        //Draws piechart rental distribution by the quarter of rental date
        draw_pie_chart_rental_by_rental_quarter(frm, frm.getJPanel5());

        //Draws barchart rental distribution by film release year
        draw_bar_chart_rental_by_release_year(frm, frm.getJPanel6());
    }

    private static void draw_pie_chart_rental_by_rental_film_duration(Charts_
frm, JPanel jp){
        jp.setPreferredSize(new Dimension(jp.getWidth(), jp.getHeight()));
        DefaultPieDataset dataset =
create_pie_dataset(Query_Rental.get_sql_rental_film_duration_dist(),
"number_of_films", "film_duration");

        //Draws piechart rental distribution by the film duration
        draw_piechart_with_dataset(frm, jp, dataset, "THE RENTAL DISTRIBUTION
DURATION");
    }

    private static void draw_pie_chart_rental_by_rental_film_rating(Charts_Re
frm, JPanel jp){
        jp.setPreferredSize(new Dimension(jp.getWidth(), jp.getHeight()));
        DefaultPieDataset dataset =
create_pie_dataset(Query_Rental.get_sql_rental_film_rating_dist(), "number_o
"film_rating");

        //Draws piechart rental distribution by the film rating
        draw_piechart_with_dataset(frm, jp, dataset, "THE RENTAL DISTRIBUTION
RATING");
    }

    private static void draw_bar_chart_rental_by_staff_name(Charts_Rental2 fr
jp){
        jp.setPreferredSize(new Dimension(jp.getWidth(), jp.getHeight()));

        DefaultCategoryDataset dataset =
create_bar_dataset(Query_Rental.get_sql_rental_staff_name_dist(), "number_of_
"staff_name");

        //Draws barchart rental distribution by staff name
        draw_barchart_with_dataset(frm, jp, dataset, "THE TOP 10 RENTAL DISTR
BY STAFF NAME", "STAFF NAME", "NUMBER OF RENTED FILMS");
    }

    private static void draw_pie_chart_rental_by_language(Charts_Rental2 frm,
jp){
        jp.setPreferredSize(new Dimension(jp.getWidth(), jp.getHeight()));
        DefaultPieDataset dataset =
create_pie_dataset(Query_Rental.get_sql_rental_film_language_dist(),
"number_of_films", "film_language");

        //Draws piechart rental distribution by the film language
```

```
        draw_piechart_with_dataset(frm, jp, dataset, "THE RENTAL DISTRIBUTION
LANGUAGE");
    }

    private static void draw_bar_chart_rental_by_film_title(Charts_Rental2 fr
jp){
        jp.setPreferredSize(new Dimension(jp.getWidth(), jp.getHeight()));

        DefaultCategoryDataset dataset =
create_bar_dataset(Query_Rental.get_sql_rental_film_title_dist(), "number_of_
"film_title");

        //Draws barchart rental distribution by film title
        draw_barchart_with_dataset(frm, jp, dataset, "THE TOP 10 RENTAL DISTR
BY FILM TITLE", "FILM TITLE", "NUMBER OF RENTED FILMS");
    }

    private static void draw_bar_chart_rental_by_customer_active(Charts_Renta
JPanel jp){
        jp.setPreferredSize(new Dimension(jp.getWidth(), jp.getHeight()));

        DefaultCategoryDataset dataset =
create_bar_dataset(Query_Rental.get_sql_rental_customer_active_dist(),
"number_of_films", "customer_active");

        //Draws barchart rental distribution by customer active/not active
        draw_barchart_with_dataset(frm, jp, dataset, "THE RENTAL DISTRIBUTION
CUSTOMER ACTIVE", "CUSTOMER ACTIVE/NOT ACTIVE", "NUMBER OF RENTED FILMS");
    }

    public static void jbchart_handler2(Charts_Rental2 frm){
        //Draws piechart rental distribution by the film duration
        draw_pie_chart_rental_by_rental_film_duration(frm, frm.getJPanel1());

        //Draws piechart rental distribution by the film rating
        draw_pie_chart_rental_by_rental_film_rating(frm, frm.getJPanel2());

        //Draws barchart rental distribution by staff name
        draw_bar_chart_rental_by_staff_name(frm, frm.getJPanel3());

        //Draws piechart rental distribution by the film language
        draw_pie_chart_rental_by_language(frm, frm.getJPanel4());

        //Draws barchart rental distribution by film title
        draw_bar_chart_rental_by_film_title(frm, frm.getJPanel5());

        //Draws barchart rental distribution by customer active/not active
        draw_bar_chart_rental_by_customer_active(frm, frm.getJPanel6());

    }

    private static void draw_bar_chart_rental_by_film_category(Charts_Rental3
JPanel jp){
```

```
        jp.setPreferredSize(new Dimension(jp.getWidth(), jp.getHeight()));

        DefaultCategoryDataset dataset =
create_bar_dataset(Query_Rental.get_sql_rental_category_dist(), "number_of_f
"film_category");

        //Draws barchart rental distribution by film category
        draw_barchart_with_dataset(frm, jp, dataset, "THE TOP 10 RENTAL DISTR
BY FILM CATEGORY", "FILM CATEGORY", "NUMBER OF RENTED FILMS");
    }

    private static void draw_bar_chart_rental_by_actor_name(Charts_Rental3 fr
jp){
        jp.setPreferredSize(new Dimension(jp.getWidth(), jp.getHeight()));

        DefaultCategoryDataset dataset =
create_bar_dataset(Query_Rental.get_sql_rental_actor_dist(), "number_of_films
"actor_name");

        //Draws barchart rental distribution by actor name
        draw_barchart_with_dataset(frm, jp, dataset, "THE TOP 10 RENTAL DISTR
BY ACTOR NAME", "ACTOR NAME", "NUMBER OF RENTED FILMS");
    }
```

```
    private static void
draw_bar_chart_rental_by_customer_name(Charts_Rental3 frm, JPanel jp){
        jp.setPreferredSize(new Dimension(jp.getWidth(),
jp.getHeight()));

        DefaultCategoryDataset dataset =
create_bar_dataset(Query_Rental.get_sql_rental_customer_name_dist(),
"number_of_films", "customer_name");

        //Draws barchart rental distribution by customer name
        draw_barchart_with_dataset(frm, jp, dataset, "THE TOP 10 RENTAL
DISTRIBUTION BY CUSTOMER NAME", "CUSTOMER NAME", "NUMBER OF RENTED
FILMS");
    }

    private static void
draw_bar_chart_rental_by_customer_city(Charts_Rental3 frm, JPanel jp){
        jp.setPreferredSize(new Dimension(jp.getWidth(),
jp.getHeight()));

        DefaultCategoryDataset dataset =
create_bar_dataset(Query_Rental.get_sql_rental_customer_city_dist(),
"number_of_films", "customer_city");

        //Draws barchart rental distribution by customer city
        draw_barchart_with_dataset(frm, jp, dataset, "THE TOP 10 RENTAL
DISTRIBUTION BY CUSTOMER CITY", "CUSTOMER CITY", "NUMBER OF RENTED
FILMS");
    }

    private static void
draw_bar_chart_rental_by_customer_country(Charts_Rental3 frm, JPanel jp)
{
        jp.setPreferredSize(new Dimension(jp.getWidth(),
jp.getHeight()));

        DefaultCategoryDataset dataset =
create_bar_dataset(Query_Rental.get_sql_rental_customer_country_dist(),
"number_of_films", "customer_country");

        //Draws barchart rental distribution by customer country
        draw_barchart_with_dataset(frm, jp, dataset, "THE TOP 10 RENTAL
DISTRIBUTION BY CUSTOMER COUNTRY", "CUSTOMER COUNTRY", "NUMBER OF RENTED
FILMS");
    }

    private static void
draw_bar_chart_rental_by_customer_district(Charts_Rental3 frm, JPanel
jp){
        jp.setPreferredSize(new Dimension(jp.getWidth(),
jp.getHeight()));
```

```
        DefaultCategoryDataset dataset =
create_bar_dataset(Query_Rental.get_sql_rental_customer_district_dist(),
"number_of_films", "customer_district");

        //Draws barchart rental distribution by customer district
        draw_barchart_with_dataset(frm, jp, dataset, "THE TOP 10 RENTAL
DISTRIBUTION BY CUSTOMER DISTRICT", "CUSTOMER DISTRICT", "NUMBER OF
RENTED FILMS");
    }

    public static void jbchart_handler3(Charts_Rental3 frm){
        //Draws piechart rental distribution by the film category
        draw_bar_chart_rental_by_film_category(frm, frm.getJPanel1());

        //Draws barchart rental distribution by actor name
        draw_bar_chart_rental_by_actor_name(frm, frm.getJPanel2());

        //Draws barchart rental distribution by customer name
        draw_bar_chart_rental_by_customer_name(frm, frm.getJPanel3());

        //Draws barchart rental distribution by customer city
        draw_bar_chart_rental_by_customer_city(frm, frm.getJPanel4());

        //Draws barchart rental distribution by customer country
        draw_bar_chart_rental_by_customer_country(frm,
frm.getJPanel5());

        //Draws barchart rental distribution by customer district
        draw_bar_chart_rental_by_customer_district(frm,
frm.getJPanel6());
    }
}
```

This is the full version of **RentalForm.java**:

```
package sakila;

import com.toedter.calendar.JCalendar;
import java.awt.Toolkit;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JComboBox;
import javax.swing.JMenuItem;
import javax.swing.JOptionPane;
import javax.swing.JPopupMenu;
import javax.swing.JTable;
import javax.swing.JTextField;

public class RentalForm extends javax.swing.JFrame {
```

```
    public RentalForm() {
        initComponents();
        Utility.setLookAndFeel(this);
        Rental_Utils.refresh_controls(this);

 this.setIconImage(Toolkit.getDefaultToolkit().getImage(getClass().getResour
;
//        this.setDefaultCloseOperation(this.HIDE_ON_CLOSE);
    }

    //Getter method for jtfRentalID
    public JTextField getJTFRentalID(){
        return this.jtfRentalID;
    }

    //Getter method for jcbRentalID
    public JComboBox getJCBRentalID(){
        return this.jcbRentalID;
    }

    //Getter method for jcRentalDate
    public JCalendar getJCRentalDate(){
        return this.jcRentalDate;
    }

    //Getter method for jcReturnDate
    public JCalendar getJCReturnDate(){
        return this.jcReturnDate;
    }

    //Getter method for jtfLastUpdate
    public JTextField getJTFLastUpdate(){
        return this.jtfLastUpdate;
    }


    //Getter method for jtfInventoryID
    public JTextField getJTFInventoryID(){
        return this.jtfInventoryID;
    }

    //Getter method for jcbInventoryID
    public JComboBox getJCBInventoryID(){
        return this.jcbInventoryID;
    }

    //Getter method for jtfFilmID
    public JTextField getJTFFilmID(){
        return this.jtfFilmID;
    }

    //Getter method for jtfStoreID
    public JTextField getJTFStoreID(){
        return this.jtfStoreID;
```

```
    }

    //Getter method for jtfAddressID
    public JTextField getJTFAddressID(){
        return this.jtfAddressID;
    }

    //Getter method for jtfAddress
    public JTextField getJTFAddress(){
        return this.jtfAddress;
    }

    //Getter method for jtfDistrict
    public JTextField getJTFDistrict(){
        return this.jtfDistrict;
    }


    //Getter method for jtfPhone
    public JTextField getJTFPhone(){
        return this.jtfPhone;
    }

    //Getter method for jtfCity
    public JTextField getJTFCity(){
        return this.jtfCity;
    }

    //Getter method for jtfCountry
    public JTextField getJTFCountry(){
        return this.jtfCountry;
    }

    //Getter method for jtfPostalCode
    public JTextField getJTFPostalCode(){
        return this.jtfPostalCode;
    }

    //Getter method for jtfTitle
    public JTextField getJTFTitle(){
        return this.jtfTitle;
    }

    //Getter method for jtfDescription
    public JTextField getJTFDescription(){
        return this.jtfDescription;
    }

    //Getter method for jtfReleaseYear
    public JTextField getJTFReleaseYear(){
        return this.jtfReleaseYear;
    }
```

```
    //Getter method for jtfRentalDuration
    public JTextField getJTFRentalDuration(){
        return this.jtfRentalDuration;
    }

    //Getter method for jtfRentalRate
    public JTextField getJTFRentalRate(){
        return this.jtfRentalRate;
    }

    //Getter method for jtfLength
    public JTextField getJTFLength(){
        return this.jtfLength;
    }

    //Getter method for jtfReplacementCost
    public JTextField getJTFReplacementCost(){
        return this.jtfReplacementCost;
    }

    //Getter method for jtfSpecialFeatures
    public JTextField getJTFSpecialFeatures(){
        return this.jtfSpecialFeatures;
    }

    //Getter method for jtfRating
    public JTextField getJTFRating(){
        return this.jtfRating;
    }

    //Getter method for jtfLanguageID
    public JTextField getJTFLanguageID(){
        return this.jtfLanguageName;
    }

    //Getter method for jtfCustomerID
    public JTextField getJTFCustomerID(){
        return this.jtfCustomerID;
    }

    //Getter method for jcbCustomerID
    public JComboBox getJCBCustomerID(){
        return this.jcbCustomerID;
    }

    //Getter method for jtfCustFname
    public JTextField getJTFCustomerName(){
        return this.jtfCustomerName;
    }

    //Getter method for jtfCustEmail
    public JTextField getJTFCustEmail(){
        return this.jtfCustEmail;
```

```
    }

    //Getter method for jtfStaffID
    public JTextField getJTFStaffID(){
        return this.jtfStaffID;
    }

    //Getter method for jcbStaffID
    public JComboBox getJCBStaffID(){
        return this.jcbStaffID;
    }

    //Getter method for jtfStaffName
    public JTextField getJTFStaffName(){
        return this.jtfStaffName;
    }


    //Getter method for jtfStaffEmail
    public JTextField getJTFStaffEmail(){
        return this.jtfStaffEmail;
    }

    //Getter method for jtfCategory
    public JTextField getJTFCategory(){
        return this.jtfCategory;
    }

    //Getter method for jtfLanguageName
    public JTextField getJTFLanguageName(){
        return this.jtfLanguageName;
    }

    //Getter method for jtfActorName
    public JTextField getJTFActorName(){
        return this.jtfActorName;
    }

    //Getter method for jtRental
    public JTable getJTRental(){
        return this.jtRental;
    }

    //Getter method for jbEdit
    public JButton getJBEdit(){
        return this.jbEdit;
    }

    //Getter method for jbInsert
    public JButton getJBInsert(){
        return this.jbInsert;
    }
```

```
    //Getter method for jbDelete
    public JButton getJBDelete(){
        return this.jbDelete;
    }

    //Getter method for jbChart
    public JButton getJBChart(){
        return this.jbChart;
    }

    //Getter method for jbFirst
    public JButton getJBFirst(){
        return this.jbFirst;
    }

    //Getter method for jbPrev
    public JButton getJBPrev(){
        return this.jbPrev;
    }

    //Getter method for jbNext
    public JButton getJBNext(){
        return this.jbNext;
    }

    //Getter method for jbLast
    public JButton getJBLast(){
        return this.jbLast;
    }

        @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">
    private void initComponents() {
        //...
        pack();
    }// </editor-fold>

    private void jcbRentalIDActionPerformed(java.awt.event.ActionEvent evt) {
        Rental_Utils.jcbRental_handler(this);
    }

    private void jtRentalMousePressed(java.awt.event.MouseEvent evt) {
        Rental_Utils.mouse_pressed_handler(this);
    }

    private void jbFirstActionPerformed(java.awt.event.ActionEvent evt) {
        Rental_Utils.show_first_row(this);
    }

    private void jbPrevActionPerformed(java.awt.event.ActionEvent evt) {
        Rental_Utils.show_prev_row(this);
    }
```

```
    private void jbNextActionPerformed(java.awt.event.ActionEvent evt) {
        Rental_Utils.show_next_row(this);
    }

    private void jbLastActionPerformed(java.awt.event.ActionEvent evt) {
        Rental_Utils.show_last_row(this);
    }

    private void jbChartActionPerformed(java.awt.event.ActionEvent evt) {
        Charts_Rental frm1 = new Charts_Rental();
        frm1.setLocationRelativeTo(null);
        frm1.setTitle("SIX DISTRIBUTIONS IN RENTAL TABLE");
        frm1.setVisible(true);
        Rental_Utils.jbchart_handler(frm1);

        Charts_Rental2 frm2 = new Charts_Rental2();
        frm2.setLocationRelativeTo(null);
        frm2.setTitle("SIX DISTRIBUTIONS IN RENTAL TABLE");
        frm2.setVisible(true);
        Rental_Utils.jbchart_handler2(frm2);

        Charts_Rental3 frm3 = new Charts_Rental3();
        frm3.setLocationRelativeTo(null);
        frm3.setTitle("SIX DISTRIBUTIONS IN RENTAL TABLE");
        frm3.setVisible(true);
        Rental_Utils.jbchart_handler3(frm3);
    }

    private void jbEditActionPerformed(java.awt.event.ActionEvent evt) {
        Rental_Utils.edit_handler(this);
    }

    private void jbInsertActionPerformed(java.awt.event.ActionEvent evt) {
        Rental_Utils.insert_handler(this);
    }

    private void jbDeleteActionPerformed(java.awt.event.ActionEvent evt) {
        Rental_Utils.delete_handler(this);
    }

    private void jbCountryFormActionPerformed(java.awt.event.ActionEvent evt)
        CountryForm ct_form = new CountryForm();
        ct_form.setVisible(true);
    }

    private void jbAddressFormActionPerformed(java.awt.event.ActionEvent evt)
        AddressForm add_form = new AddressForm();
        add_form.setVisible(true);
    }

    private void jbStoreFormActionPerformed(java.awt.event.ActionEvent evt) {
        StoreForm str_form = new StoreForm();
```

```
        str_form.setVisible(true);
    }

    private void jbCityFormActionPerformed(java.awt.event.ActionEvent evt) {
        CityForm cty_form = new CityForm();
        cty_form.setVisible(true);
    }

    private void jcbInventoryIDActionPerformed(java.awt.event.ActionEvent evt
        Rental_Utils.jcbInventory_handler(this);
    }

    private void jcbCustomerIDActionPerformed(java.awt.event.ActionEvent evt)
        Rental_Utils.jcbCustomer_handler(this);
    }

    private void jcbStaffIDActionPerformed(java.awt.event.ActionEvent evt) {
        Rental_Utils.jcbStaff_handler(this);
    }

    private void jbCityForm1ActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
    }

    private void jbFilmFormActionPerformed(java.awt.event.ActionEvent evt) {
        FilmForm frm = new FilmForm();
        frm.setVisible(true);
    }

    private void jbCategoryFormActionPerformed(java.awt.event.ActionEvent evt
        CategoryForm frm = new CategoryForm();
        frm.setVisible(true);
    }

    private void jbActorFormActionPerformed(java.awt.event.ActionEvent evt) {
        ActorForm frm = new ActorForm();
        frm.setVisible(true);
    }

    public static void main(String args[]) {
        try {
            for (javax.swing.UIManager.LookAndFeelInfo info :
javax.swing.UIManager.getInstalledLookAndFeels()) {
                if ("Nimbus".equals(info.getName())) {
                    javax.swing.UIManager.setLookAndFeel(info.getClassName()
                    break;
                }
            }
        } catch (ClassNotFoundException ex) {

java.util.logging.Logger.getLogger(RentalForm.class.getName()).log(java.util.
null, ex);
        } catch (InstantiationException ex) {
```

```
java.util.logging.Logger.getLogger(RentalForm.class.getName()).log(java.util.
null, ex);
        } catch (IllegalAccessException ex) {

java.util.logging.Logger.getLogger(RentalForm.class.getName()).log(java.util.
null, ex);
        } catch (javax.swing.UnsupportedLookAndFeelException ex) {

java.util.logging.Logger.getLogger(RentalForm.class.getName()).log(java.util.
null, ex);
        }
        /* Create and display the form */
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new RentalForm().setVisible(true);
            }

        });
    }

    // Variables declaration - do not modify
    private javax.swing.JLabel jLabel10;
    private javax.swing.JLabel jLabel11;
    private javax.swing.JLabel jLabel12;
    private javax.swing.JLabel jLabel13;
    private javax.swing.JLabel jLabel14;
    private javax.swing.JLabel jLabel16;
    private javax.swing.JLabel jLabel17;
    private javax.swing.JLabel jLabel18;
    private javax.swing.JLabel jLabel19;
    private javax.swing.JLabel jLabel20;
    private javax.swing.JLabel jLabel21;
    private javax.swing.JLabel jLabel22;
    private javax.swing.JLabel jLabel23;
    private javax.swing.JLabel jLabel24;
    private javax.swing.JLabel jLabel25;
    private javax.swing.JLabel jLabel26;
    private javax.swing.JLabel jLabel27;
    private javax.swing.JLabel jLabel28;
    private javax.swing.JLabel jLabel29;
    private javax.swing.JLabel jLabel3;
    private javax.swing.JLabel jLabel30;
    private javax.swing.JLabel jLabel31;
    private javax.swing.JLabel jLabel32;
    private javax.swing.JLabel jLabel33;
    private javax.swing.JLabel jLabel34;
    private javax.swing.JLabel jLabel35;
    private javax.swing.JLabel jLabel4;
    private javax.swing.JLabel jLabel5;
    private javax.swing.JLabel jLabel6;
    private javax.swing.JLabel jLabel7;
    private javax.swing.JLabel jLabel8;
```

```
    private javax.swing.JLabel jLabel9;
    private javax.swing.JScrollPane jScrollPane1;
    private javax.swing.JButton jbActorForm;
    private javax.swing.JButton jbAddressForm;
    private javax.swing.JButton jbCategoryForm;
    private javax.swing.JButton jbChart;
    private javax.swing.JButton jbCityForm;
    private javax.swing.JButton jbCityForm1;
    private javax.swing.JButton jbCountryForm;
    private javax.swing.JButton jbDelete;
    private javax.swing.JButton jbEdit;
    private javax.swing.JButton jbFilmForm;
    private javax.swing.JButton jbFirst;
    private javax.swing.JButton jbInsert;
    private javax.swing.JButton jbLast;
    private javax.swing.JButton jbNext;
    private javax.swing.JButton jbPrev;
    private javax.swing.JButton jbStoreForm;
    private com.toedter.calendar.JCalendar jcRentalDate;
    private com.toedter.calendar.JCalendar jcReturnDate;
    private javax.swing.JComboBox<String> jcbCustomerID;
    private javax.swing.JComboBox<String> jcbInventoryID;
    private javax.swing.JComboBox<String> jcbRentalID;
    private javax.swing.JComboBox<String> jcbStaffID;
    private javax.swing.JTable jtRental;
    private javax.swing.JTextField jtfActorName;
    private javax.swing.JTextField jtfAddress;
    private javax.swing.JTextField jtfAddressID;
    private javax.swing.JTextField jtfCategory;
    private javax.swing.JTextField jtfCity;
    private javax.swing.JTextField jtfCountry;
    private javax.swing.JTextField jtfCustEmail;
    private javax.swing.JTextField jtfCustomerID;
    private javax.swing.JTextField jtfCustomerName;
    private javax.swing.JTextField jtfDescription;
    private javax.swing.JTextField jtfDistrict;
    private javax.swing.JTextField jtfFilmID;
    private javax.swing.JTextField jtfInventoryID;
    private javax.swing.JTextField jtfLanguageName;
    private javax.swing.JTextField jtfLastUpdate;
    private javax.swing.JTextField jtfLength;
    private javax.swing.JTextField jtfPhone;
    private javax.swing.JTextField jtfPostalCode;
    private javax.swing.JTextField jtfRating;
    private javax.swing.JTextField jtfReleaseYear;
    private javax.swing.JTextField jtfRentalDuration;
    private javax.swing.JTextField jtfRentalID;
    private javax.swing.JTextField jtfRentalRate;
    private javax.swing.JTextField jtfReplacementCost;
    private javax.swing.JTextField jtfSpecialFeatures;
    private javax.swing.JTextField jtfStaffEmail;
    private javax.swing.JTextField jtfStaffID;
    private javax.swing.JTextField jtfStaffName;
    private javax.swing.JTextField jtfStoreID;
```

```
    private javax.swing.JTextField jtfTitle;
    // End of variables declaration
}
```

PAYMENT FORM

PAYMENT FORM

CREATING AND POPULATING PAYMENT TABLE

CREATING AND POPULATING PAYMENT TABLE

Step 1	Create a new class named **Query_Payment**. These are a series of SQL queries to extract payment-related data from a database. Here's what each query does: 1. **sql_payment_quarter**: This query retrieves the total payment made for each quarter of the year, starting from January to March and going up to October to December. It uses a CASE statement to group the payment data by quarter. 2. **sql_payment_day**: This query retrieves the total payment made for each day of the week. It uses the DAYNAME function to get the name of the day from the payment date and groups the payment data by day. 3. **sql_payment_week**: This query retrieves the total payment made for each week of the year. It uses the WEEK function to get the week number from the payment date and groups the payment data by week. It also limits the output to the top 10 weeks. 4. **sql_payment_month**: This query retrieves the total payment made for each month of the year. It uses the MONTHNAME function to get the name of the month from the payment date and groups the payment data by month. 5. **sql_payment_year**: This query retrieves the total payment made for each year in the payment data. It groups the payment data by year and sorts the result in descending order by payment amount. 6. **sql_payment_film_year_dist**: This query retrieves the total payment made for each film release year in the payment data. It joins the payment, rental, inventory, and film tables to get the release year of each film and groups the payment data by the film release year. It then sorts the result

first by the number of films released in that year and then by payment amount.

7. **sql_payment_film_duration_dist**: This query retrieves the total payment made for films with different rental durations. It joins the payment, rental, inventory, and film tables to get the rental duration of each film and groups the payment data by film rental duration. It then sorts the result first by the number of rentals of films with that duration and then by payment amount.
8. **sql_payment_film_rating_dist**: This query retrieves the total payment made for films with different ratings. It joins the payment, rental, inventory, and film tables to get the rating of each film and groups the payment data by film rating. It then sorts the result in descending order by payment amount.
9. **sql_payment_staff_name_dist**: This query retrieves the total payment processed by each staff member. It joins the payment and staff tables to get the name of each staff member and groups the payment data by staff name. It then sorts the result in descending order by payment amount and limits the output to the top 10 staff members.
10. **sql_payment_film_language_dist**: This query retrieves the top 10 film languages by the total amount of payments received for the films in each language.
11. **sql_payment_film_title_dist**: This query retrieves the top 10 film titles by the total amount of payments received for each film.
12. **sql_payment_customer_active_dist**: This query retrieves the top 10 customer active statuses (i.e. whether they are active or inactive) by the total amount of payments made by each customer.
13. **sql_payment_customer_district_dist:** This query retrieves the top 10 customer districts by the total amount

of payments made by customers in each district.

14. **sql_payment_customer_country_dist**: This query retrieves the top 10 customer countries by the total amount of payments made by customers in each country.
15. **sql_payment_customer_city_dist**: This query retrieves the top 10 customer cities by the total amount of payments made by customers in each city.
16. **sql_payment_customer_name_dist**: This query retrieves the top 10 customer names by the total amount of payments made by each customer.
17. **sql_payment_actor_dist**: This query retrieves the top 10 actors by the total amount of payments received for the films they have acted in.
18. **sql_payment_category_dist**: This query retrieves the top 10 film categories by the total amount of payments received for the films in each category.
19. **sql_payment_joint** - This query retrieves the details of the top 10 payments made, including the payment ID, customer ID, staff ID, rental ID, payment amount, payment date, rental date, film ID, film title, film description, film release year, film rental duration, film rental rate, film length, film replacement cost, film rating, film special features, language name, category name, actor name, and customer name.

```
1       package sakila;
2
3       public class Query_Payment {
4           private static final String
5       sql_min = "SELECT MIN(payment_id)
6       FROM payment";
7           private static final String
8       sql_max = "SELECT MAX(payment_id)
9       FROM payment";
10          private static final String
11      sql_id = "SELECT * FROM payment
```

```
WHERE payment_id  = ?";


    private static final String
sql_payment_quarter = """
        SELECT
            CASE
                WHEN
MONTH(payment_date) BETWEEN 1 AND
3 THEN 'January to March'
                WHEN
MONTH(payment_date) BETWEEN 4 AND
6 THEN 'April to June'
                WHEN
MONTH(payment_date) BETWEEN 7 AND
9 THEN 'July to September'
                ELSE 'October to
December'
            END AS quarter,
        SUM(amount) AS payment
        FROM payment
        GROUP BY quarter
        ORDER BY Count(*)
DESC;""";

    private static final String
sql_payment_day = """
        SELECT
DAYNAME(payment_date) AS day,
SUM(amount) AS payment
        FROM payment
        GROUP BY day
        ORDER BY Count(*)
DESC;""";

    private static final String
sql_payment_week = """
        SELECT WEEK(payment_date)
AS week, SUM(amount) AS payment
        FROM payment
        GROUP BY week
        ORDER BY Count(*) DESC
        LIMIT 10;""";

    private static final String
sql_payment_month = """
        SELECT
MONTHNAME(payment_date) AS month,
SUM(amount) AS payment
        FROM payment
        GROUP BY month
        ORDER BY Count(*)
DESC;""";

```

```
66
67      private static final String
68  sql_payment_year = """
69          SELECT YEAR(payment_date)
70  AS year, SUM(p.amount) AS payment
71          FROM payment p
72          GROUP BY year
73          ORDER BY payment DESC;""";
74
75      private static final String
76  sql_payment_film_year_dist = """
77          SELECT
78  YEAR(f.release_year) AS film_year,
79  SUM(p.amount) AS payment
80          FROM payment p
81          JOIN rental r ON
82  r.rental_id = p.rental_id
83          JOIN inventory i ON
84  r.inventory_id = i.inventory_id
85          JOIN film f ON i.film_id =
86  f.film_id
87          GROUP BY film_year
88          ORDER BY Count(*) DESC,
89  payment DESC;""";
90
91      private static final String
92  sql_payment_film_duration_dist =
93  """
94          SELECT f.rental_duration
95  AS film_duration, SUM(p.amount) AS
96  payment
97          FROM payment p
98          JOIN rental r ON
99  r.rental_id = p.rental_id
100         JOIN inventory i ON
101 r.inventory_id = i.inventory_id
102         JOIN film f ON i.film_id =
103 f.film_id
104         GROUP BY film_duration
105         ORDER BY Count(*) DESC,
106 payment DESC;""";
107
108     private static final String
109 sql_payment_film_rating_dist = """
110         SELECT f.rating AS
111 film_rating, SUM(p.amount) AS
112 payment
113         FROM payment p
114         JOIN rental r ON
115 r.rental_id = p.rental_id
116         JOIN inventory i ON
117 r.inventory_id = i.inventory_id
118         JOIN film f ON i.film_id =
119 f.film_id
```

```
120         GROUP BY film_rating
121         ORDER BY payment DESC;""";
122
123     private static final String
124 sql_payment_staff_name_dist = """
125         SELECT
126127 CONCAT(st.first_name,'
128 ',st.last_name) AS staff_name,
129 SUM(p.amount) AS payment
130         FROM payment p
131         JOIN staff st ON
132 st.staff_id = p.staff_id
133         GROUP BY staff_name
134         ORDER BY payment DESC
135         LIMIT 10;""";
136
137     private static final String
138 sql_payment_film_language_dist =
139 """
140         SELECT l.name AS
141 film_language, SUM(p.amount) AS
142 payment
143         FROM payment p
144         JOIN rental r ON
145 r.rental_id = p.rental_id
146         JOIN inventory i ON
147 r.inventory_id = i.inventory_id
148         JOIN film f ON i.film_id =
149 f.film_id
150         JOIN language l ON
151 l.language_id = f.language_id
152         GROUP BY film_language
153         ORDER BY payment DESC
154         LIMIT 10;""";
155
156     private static final String
157 sql_payment_film_title_dist = """
158         SELECT f.title AS
159 film_title, SUM(p.amount) AS
160 payment
161         FROM payment p
162         JOIN rental r ON
163 r.rental_id = p.rental_id
164         JOIN inventory i ON
165 r.inventory_id = i.inventory_id
166         JOIN film f ON i.film_id =
167 f.film_id
168         GROUP BY film_title
169         ORDER BY payment DESC
170         LIMIT 10;""";
171
172     private static final String
173 sql_payment_customer_active_dist =
174 """
```

```
        SELECT cu.active AS
customer_active, SUM(p.amount) AS
payment
        FROM payment p
        JOIN customer cu ON
cu.customer_id = p.customer_id
        GROUP BY customer_active
        ORDER BY payment DESC
        LIMIT 10;""";

    private static final String
sql_payment_customer_district_dist
= """
        SELECT IFNULL(ad.district,
'UNKNOWN') AS customer_district,
SUM(p.amount) AS payment
        FROM payment p
        JOIN customer cu ON
cu.customer_id = p.customer_id
        JOIN address ad ON
ad.address_id = cu.address_id
        JOIN city ci ON ci.city_id
= ad.city_id
        JOIN country co ON
co.country_id = ci.country_id
        GROUP BY customer_district
```

	229 230 231 232 233 234 235 236 237 238 239 240 241 242 243 244 245 246247 248 249 250 251 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271 272 273 274 275 276 277 278 279 280 281 282 283		

```
        ORDER BY payment DESC
        LIMIT 10;""";

    private static final String
sql_payment_customer_country_dist = """
        SELECT co.country AS
customer_country, SUM(p.amount) AS
payment
        FROM payment p
        JOIN customer cu ON
cu.customer_id = p.customer_id
        JOIN address ad ON
ad.address_id = cu.address_id
        JOIN city ci ON ci.city_id =
ad.city_id
        JOIN country co ON
co.country_id = ci.country_id
        GROUP BY customer_country
        ORDER BY payment DESC
        LIMIT 10;""";

    private static final String
sql_payment_customer_city_dist = """
        SELECT ci.city AS
customer_city, SUM(p.amount) AS payment
        FROM payment p
        JOIN customer cu ON
cu.customer_id = p.customer_id
        JOIN address ad ON
ad.address_id = cu.address_id
        JOIN city ci ON ci.city_id =
ad.city_id
        JOIN country co ON
co.country_id = ci.country_id
        GROUP BY customer_city
        ORDER BY payment DESC
        LIMIT 10;""";

    private static final String
sql_payment_customer_name_dist = """
        SELECT CONCAT(cu.first_name,'
',cu.last_name) AS customer_name,
SUM(p.amount) AS payment
        FROM payment p
        JOIN customer cu ON
cu.customer_id = p.customer_id
        JOIN address ad ON
ad.address_id = cu.address_id
        JOIN city ci ON ci.city_id =
ad.city_id
        JOIN country co ON
co.country_id = ci.country_id
        GROUP BY customer_name
```

```
        ORDER BY payment DESC
        LIMIT 10;""";

    private static final String
sql_payment_actor_dist = """
        SELECT CONCAT(a.first_name,'
',a.last_name) AS actor_name,
SUM(p.amount) AS payment
        FROM payment p
        JOIN rental r ON r.rental_id =
p.rental_id
        JOIN inventory i ON
r.inventory_id = i.inventory_id
        JOIN film f ON i.film_id =
f.film_id
        JOIN language l ON
l.language_id = f.language_id
        JOIN film_actor fa ON f.film_id
= fa.actor_id
        JOIN actor a ON fa.actor_id =
a.actor_id
        GROUP BY actor_name
        ORDER BY payment DESC
        LIMIT 10;""";

    private static final String
sql_payment_category_dist = """
        SELECT c.name AS film_category,
SUM(p.amount) AS payment
        FROM payment p
        JOIN rental r ON r.rental_id =
p.rental_id
        JOIN inventory i ON
r.inventory_id = i.inventory_id
        JOIN film f ON i.film_id =
f.film_id
        JOIN language l ON
l.language_id = f.language_id
        JOIN film_category fc ON
f.film_id = fc.film_id
        JOIN category c ON
fc.category_id = c.category_id
        GROUP BY film_category
        ORDER BY payment DESC
        LIMIT 10;""";

    private static final String
sql_payment_joint = """
        SELECT p.payment_id,
p.customer_id, p.staff_id, p.rental_id,
            p.amount, p.payment_date,
p.last_update,
```

```
            r.rental_date,
r.inventory_id, r.return_date,
            f.film_id, f.title,
f.description, f.release_year,
            f.rental_duration,
f.rental_rate, f.length,
f.replacement_cost, f.rating,
            f.special_features, l.name
AS language_name, c.name AS
category_name,
            CONCAT(a.first_name,'
',a.last_name) AS actor_name,
            CONCAT(cu.first_name,'
',cu.last_name) AS customer_name,
cu.email AS customer_email,
            CONCAT(st.first_name,'
',st.last_name) AS staff_name, st.email
AS staff_email,
            ci.city AS store_city,
co.country AS store_country,
            ad.phone AS store_phone,
ad.district AS store_district,
            ad.postal_code AS
store_postal_code
        FROM payment p
        JOIN rental r ON r.rental_id =
p.rental_id
        JOIN inventory i ON
i.inventory_id = r.inventory_id
        JOIN store so ON so.store_id =
i.store_id
        JOIN address ad ON
ad.address_id = so.address_id
        JOIN city ci ON ci.city_id =
ad.city_id
        JOIN country co ON
co.country_id = ci.country_id
        JOIN staff st ON st.staff_id =
p.staff_id
        JOIN customer cu ON
cu.customer_id = p.customer_id
        JOIN film f ON i.film_id =
f.film_id
        JOIN language l ON
l.language_id = f.language_id
        JOIN film_category fc ON
f.film_id = fc.film_id
        JOIN category c ON
fc.category_id = c.category_id
        JOIN film_actor fa ON f.film_id
= fa.film_id
        JOIN actor a ON fa.actor_id =
a.actor_id""";
```

```
    private static final String
sql_payment = """
        CREATE TABLE payment (
          payment_id SMALLINT UNSIGNED
NOT NULL AUTO_INCREMENT,
          customer_id SMALLINT UNSIGNED
NOT NULL,
          staff_id TINYINT UNSIGNED NOT
NULL,
          rental_id INT DEFAULT NULL,
          amount DECIMAL(5,2) NOT NULL,
          payment_date DATETIME NOT
NULL,
          last_update TIMESTAMP DEFAULT
CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,
          PRIMARY KEY  (payment_id),
          KEY idx_fk_staff_id
(staff_id),
          KEY idx_fk_customer_id
(customer_id),
          CONSTRAINT fk_payment_rental
FOREIGN KEY (rental_id) REFERENCES
rental (rental_id) ON DELETE SET NULL
ON UPDATE CASCADE,
          CONSTRAINT
fk_payment_customer FOREIGN KEY
(customer_id) REFERENCES customer
(customer_id) ON DELETE RESTRICT ON
UPDATE CASCADE,
          CONSTRAINT fk_payment_staff
FOREIGN KEY (staff_id) REFERENCES staff
(staff_id) ON DELETE RESTRICT ON UPDATE
CASCADE
        ) ENGINE=InnoDB DEFAULT
CHARSET=utf8mb4;""";


    //Getter methods
    public static String get_sql_min()
{
        return sql_min;
    }

    public static String get_sql_max()
{
        return sql_max;
    }

    public static String get_sql_id() {
        return sql_id;
    }
```

```
    public static String
get_sql_payment() {
        return sql_payment;
    }

    public static String
get_sql_payment_joint() {
        return sql_payment_joint;
    }

    public static String
get_sql_payment_category_dist() {
        return
sql_payment_category_dist;
    }

    public static String
get_sql_payment_actor_dist() {
        return sql_payment_actor_dist;
    }

    public static String
get_sql_payment_customer_name_dist() {
        return
sql_payment_customer_name_dist;
    }

    public static String
get_sql_payment_customer_city_dist() {
        return
sql_payment_customer_city_dist;
    }

    public static String
get_sql_payment_customer_country_dist()
{
        return
sql_payment_customer_country_dist;
    }

    public static String
get_sql_payment_customer_active_dist()
{
        return
sql_payment_customer_active_dist;
    }

    public static String
get_sql_payment_film_title_dist() {
        return
sql_payment_film_title_dist;
    }
```

```
    public static String
get_sql_payment_film_language_dist() {
        return
sql_payment_film_language_dist;
    }

    public static String
get_sql_payment_staff_name_dist() {
        return
sql_payment_staff_name_dist;
    }

    public static String
get_sql_payment_film_rating_dist() {
        return
sql_payment_film_rating_dist;
    }

    public static String
get_sql_payment_film_duration_dist() {
        return
sql_payment_film_duration_dist;
    }

    public static String
get_sql_payment_film_year_dist() {
        return
sql_payment_film_year_dist;
    }

    public static String
get_sql_payment_year() {
        return sql_payment_year;
    }

    public static String
get_sql_payment_month() {
        return sql_payment_month;
    }

    public static String
get_sql_payment_week() {
        return sql_payment_week;
    }

    public static String
get_sql_payment_day() {
        return sql_payment_day;
    }

    public static String
get_sql_payment_quarter() {
```

```
        return sql_payment_quarter;
    }
}
```

Step 2

Then, create a public class name
instance variables and sever
getter/setter methods. The **Paym**
payment made by a customer for
store.

The 31 instance variables include t
payment date, rental ID, rental date
ID, return date, staff ID, last upd
title, description, release year, rent
length, replacement cost, rating, sp
name, category, actor name, cus
email, staff name, staff email, store
district, and store postal code.

The **Payment** class has three
constructor, a six-parameter co
parameter constructor. The six-par
the customer ID, staff ID, rental ID,
and last update timestamp.
constructor calls the six-parameter
payment ID. The 28-parameter con
parameter constructor and sets
variables.

The class also has getter and sette
instance variables. The setter met
throws an exception if the ID is less

```
1   package sakila;
2   import java.sql.Timestamp;
3   import java.sql.Date;
4   import java.util.Calendar;
5
6   public class Payment {
7       //31 instance variables
8       private int payment_id;
9       private double amount;
10      private Date payment_da
11      private int rental_id;
12      private Date rental_dat
13
14      private int inv_id;
15
```

```
16      private int cust_id;
17      private Date return_dat
18      private int staff_id;
19      private Timestamp last_
20
21      private int film_id;
22      private String title;
23      private String descript
24      private int release_yea
25      private int rental_dura
26
27      private double rental_r
28      private int length;
29      private double replacem
30      private String rating;
31      private String special_
32
33      private String language
34      private String category
35      private String actor_na
36      private String customer
37      private String customer
38
39      private String staff_na
40      private String staff_em
41      private String store_ci
42      private String store_co
43      private String store_di
44
45      private String store_po
46
47      //Default constructor
48      Payment(){
49          this(1, 1, 1, 1, 1.
50              new
51  Date(Calendar.getInstance()
52              new
53  Timestamp(System.currentTim
54      }
55
56      //Six-params constructo
57      Payment(int cust_id, in
58  rental_id, double amount, D
59  Timestamp lu){
60          setCustomerID(cust_
61          setStaffID(staff_id
62          setRentalID(rental_
63          setAmount(amount);
64          setPaymentDate(pay_
65          setLastUpdate(lu);
66      }
67
68      //Sevent-params constru
69
```

```
70      Payment(int pay_id, int
71  staff_id, int rental_id, do
72  pay_date, Timestamp lu){
73          this(cust_id, staf
74  amount, pay_date, lu);
75          setPaymentID(pay_ic
76      }
77
78      //28-params constructor
79      Payment(int pay_id, dou
80  pay_date, int rental_id, Da
81  inv_id, int cust_id, Date r
82  Timestamp lu,
83          int film_id, String
84  description, int year, int
85  int length, double cost, St
86  features, String lang_name,
87  String actor_name, String c
88  cust_email, String staff_na
89  staff_email, String store_c
90  store_country, String store
91  store_post){
92          this(pay_id, cust_i
93  rental_id, amount, pay_date
94          this.inv_id = inv_i
95          this.rental_date =
96          this.return_date =
97
98          this.film_id = film
99          this.title = title;
100         this.description =
101         this.release_year =
102         this.rental_duratio
103
104         this.rental_rate =
105         this.length = lengt
106         this.replacement_co
107         this.rating = ratin
108         this.special_featur
109
110         this.language_name
111         this.category = cat
112         this.actor_name = a
113         this.customer_name
114         this.customer_email
115
116         this.staff_name = s
117         this.staff_email =
118         this.store_city = s
119         this.store_country
120         this.store_district
121
122         this.store_postal_c
123     }
```

```
124
125         //Getter methods
126127      public int getPaymentID
128     payment_id;}
129       public double getAmount
130       public Date getPaymentD
131     this.payment_date;}
132       public int getRentalID(
133       public Date getRentalDa
134     this.rental_date;}
135       public int getInventory
136       public int getCustomerI
137       public Date getReturnDa
138     this.return_date;}
139       public int getStaffID()
140       public Timestamp getLas
141     last_update;}
142       public int getFilmID()
143       public String getTitle(
144       public String getDescri
145     description;}
146       public int getReleaseYe
147     release_year;}
148       public int getRentalDur
149     rental_duration;}
150       public double getRental
151     rental_rate;}
152       public int getLength()
153       public double getReplac
154     replacement_cost;}
155       public String getRating
156       public String getSpecia
157     special_features;}
158       public String getLangua
159     language_name;}
160
161       public String getCatego
162     category;}
163       public String getActorN
164     actor_name;}
165       public String getCustom
166     customer_name;}
167       public String getCustom
168     staff_name;}
169       public String getStaffN
170     actor_name;}
171       public String getStaffE
172     staff_email;}
173
174       public String getStore(
175     store_city;}
176       public String getStore(
177     store_country;}
178
```

```
179     public String getStoreD
180 store_district;}
181     public String getStoreP
182 store_postal_code;}
183
184     //Setter methods
185     public void setPaymentI
186         if (id <= 0) {
187             throw new Illegal
188 must be greater than zero.");
189         }
190         this.payment_id = i
191     }
192
193     public void setRentalID
194         if (id <= 0) {
195             throw new Illegal
196 must be greater than zero.");
197         }
198         this.rental_id = id
199     }
200
201     public void setCustomer
202         if (id <= 0) {
203             throw new Illegal
204 must be greater than zero.");
205         }
206         this.cust_id = id;
207     }
208
209     public void setStaffID(
210         if (id <= 0) {
211             throw new Illegal
212 must be greater than zero.");
213         }
214         this.staff_id = id;
215     }
216
217     public void setPaymentD
218         this.payment_date =
219     }
220
221     public void setAmount(d
222         if (_amount <= 0) {
223             this.amount = 1
224         }
225         this.amount = _amou
226     }
227
228     public void setLastUpda
229         if (date == null) {
```

```
            throw new IllegalAr
cannot be null");
            }
            this.last_update =
        }

        @Override
        public String toString(
            return "\nPayment I
getPaymentID() +
                    "\nRental ID
getRentalID() +
                    "\nCustomer
getCustomerID() +
                    "\nStaff ID
getStaffID() +
                    "\nAmount
getAmount() +
                    "\nPayment D
getPaymentDate() +
                    "\nLast Upda
getLastUpdate();
        }
}
```

Step 3

Create a new public class named **Pa**
defines a **Payment_Utils** class in a
contains utility methods for cre
reading the payment table in a data
Utility class and imports several cla
Java and other libraries.

The **Payment_Utils** class has a cor
stores the SQL ID of the payment
currentIndex that stores the curre
table. The class also has several stat

1. **create_payment_table()**: T
   payment table in the databa
   of SQL statements
   Query_Payment class. It sh
   success or error message, de
   batch execution was success
2. **populate_payment_table()**
   the **payment** table with
   creating **Payment** objects
   nine-params constructors a

the database using prepared
error message if any SQL or

3. **read_payment_table()**: Th
content of the payment table
statement and retrieving
payment_id, **rental_i**
customer_id, **staff_id**, **am**
columns from the **ResultSe**
object using the seven-pa
prints it to the console.

```
1  package sakila;
2  import java.awt.Dimension;
3  import java.util.logging.Level
4  import java.util.logging.Logge
5  import java.sql.*;
6  import java.text.ParseExceptio
7  import java.text.SimpleDateFor
8  import java.util.ArrayList;
9  import java.util.Arrays;
10 import java.util.HashMap;
11 import java.util.Objects;
12 import javax.swing.JOptionPane
13 import javax.swing.JPanel;
14 import javax.swing.table.Defau
15 import
16 org.jfree.data.category.Defaul
17 import org.jfree.data.general.
18
19 public class Payment_Utils ext
20     public static final int FI
21     public static final int IN
22
23     private static int current
24 FIRST_INDEX;
25     private static final Strin
26 Query_Payment.get_sql_id();
27
28     //Creates payment table
29     public static void create_
30         try (Connection conn =
31             Statement stmt =
32 conn.createStatement();
33
34 stmt.addBatch(Query_Payment.ge
35             stmt.executeBatch(
36
37             String message =
38 String.format("Successfully cr
39 table");
40
```

```
            JOptionPane.showMe
message,

 "INFORMATION",JOptionPane.INF

        } catch (SQLException
            JOptionPane.showMe
ex.getMessage(),

 "ERROR",JOptionPane.ERROR_MES
        }
    }

    //Populates payment table
data
    public static void populat
        try(Connection conn =
            String sql = """
                INSERT INTO pa
customer_id, staff_id, rental_
payment_date, last_update)
                VALUES(?, ?, ?

            //Creates a new Pa
default constructor
            PreparedStatement
conn.prepareStatement(sql);
            Payment obj1 = new
            ps1.setInt(1,obj1.
            ps1.setInt(2,obj1.
            ps1.setInt(3,obj1.
            ps1.setInt(4,obj1.
            ps1.setDouble(5,ob
            ps1.setDate(6,obj1

ps1.setTimestamp(7,obj1.getLas

            // Creates a new P
nine-params constructor
            PreparedStatement
conn.prepareStatement(sql);
            SimpleDateFormat d
SimpleDateFormat("yyyy-MM-dd")
            String str_pay_dat
            java.util.Date pay
dateFormat.parse(str_pay_date)
            java.sql.Date pay_
java.sql.Date(pay_date.getTime

            Payment obj2 = new
2, 10.75, pay_sql_date, new
Timestamp(System.currentTimeMi
            ps2.setInt(1,obj2.
```

```
95              ps2.setInt(2,obj2.
96              ps2.setInt(3,obj2.
97              ps2.setInt(4,obj2.
98              ps2.setDouble(5,ob
99              ps2.setDate(6,obj2
100
101 ps2.setTimestamp(7,obj2.getLas
102
103             ps1.executeUpdate(
104             ps2.executeUpdate(
105
106         }catch(SQLException ex
107             JOptionPane.showMe
108 ex.getMessage(),
109
110  "ERROR",JOptionPane.ERROR_MES
111         } catch (ParseExceptio
112
113 Logger.getLogger(Rental_Utils.class.g
114 null, ex);
115         }
116     }
117
    //Reads the content of pay
    public static void read_pa
        try(Connection conn =
            Statement stmt =
conn.createStatement();
            ResultSet rs =
stmt.executeQuery("SELECT * FR

            while(rs.next()){
                int pay_id =
rs.getInt("payment_id");
                int rental_id
rs.getInt("rental_id");
                Date pay_date
rs.getDate("payment_date");
                int cust_id =
rs.getInt("customer_id");
                int staff_id =
rs.getInt("staff_id");
                double amount
rs.getDouble("amount");
                Timestamp lu =
rs.getTimestamp("last_update")

                //Creates a Pa
seven-params constructor
                Payment obj =
cust_id, staff_id, rental_id,
lu);
                System.out.pri
```

```
            }
            rs.close();
            stmt.close();

        }catch(SQLException ex
            JOptionPane.showMe
ex.getMessage(),

 "ERROR",JOptionPane.ERROR_MES
        }
    }
}
```

Step 4

In the driver class, **create_payment_table()**, **populate read_payment_table()** as shown in

```
1  package sakila;
2
3  public class Sakila {
4      public static void main(Str
5  args) {
6          //...
7
8  //
9  Rental_Utils.create_rental_tabl
10 //
11 Rental_Utils.populate_rental_ta
12 //
13 Rental_Utils.read_rental_table(
14 //         RentalForm frm = new
15 RentalForm();
16 //         frm.setVisible(true);
17


   Payment_Utils.create_payment_ta

   Payment_Utils.populate_payment_

   Payment_Utils.read_payment_tabl
       }
   }
```

Run project to see the result in cons

```
Payment ID              : 1
Rental ID               : 1
Customer ID             : 1
Staff ID                : 1
Amount                  : 1.0
```

	Payment Date : 2023-05-04 Last Update : 2023-05-04 Payment ID : 2 Rental ID : 2 Customer ID : 2 Staff ID : 2 Amount : 10.75 Payment Date : 2023-05-03 Last Update : 2023-05-04

DESIGNING GUI
DESIGNING GUI

Step 1	In the project, create a new **JFrame Form** and name it as **PaymentForm.java**. In the Design tab, add thirty-five **JLabels** to the form and set their corresponding **text** properties as PAYMENT ID, AMOUNT, PAYMENT DATE, RENTAL ID, RENTAL DATE, RETURN DATE, INVENTORY ID, FILM ID, STORE ID, ADDRESS ID, ADDRESS, DISTRICT, CITY, COUNTRY, PHONE, POSTAL CODE, CUSTOMER ID, CUST. NAME, CUST.EMAIL, LAST UPDATE, STAFF ID, STAFF NAME, STAFF EMAIL, TITLE, DESCRIPTION, RELEASE YEAR, LENGTH, RATING, SPECIAL FEATURES, ACTOR NAME, RENTAL DURATION, RENTAL RATE, REPLACEMENT COST, and CATEGORY.
Step 2	Then, add thirty four **JTextField** to the form and set their corresponding **Variable Name** jtfPaymentID, jtfAmount, jtfRentalID, jtfReturnDate, jtfRentalDate, jtfInventoryID, jtfFilmID, jtfStoreID, jtfAddressID, jtfAddress, jtfDistrict, jtfCity, jtfCountry, jtfPhone, jtfPostalCode, jtfCustomerID, jtfCustomerName, jtfCustomerEmail, jtfLastUpdate, jtfStaffID, jtfStaffName, jtfStaffEmail, jtfTitle, jtfDescription, jtfReleaseYear, jtfLength, jtfRating, jtfSpecialFeatures, jtfActorName,

	jtfRentalDuration, jtfRentalRate, jtfReplacementCost, and jtfCategory.
Step 3	Then, add sixteen **JButton** to the form and set their corresponding **Variable Name** as **jbFirst**, **jbPrev**, **jbNext**, **jbLast**, **jbEdit**, **jbInsert**, **jbDelete**, **jbAddress**, **jbCountryForm**, **jbCityForm**, **jbAddressForm**, **jbStoreForm**, **jbFilmForm**, **jbActorForm**, **jbCategoryForm**, and **jbChart**. Set their corresponding **text** properties as FIRST, PREV, NEXT, LAST, EDIT, INSERT, DELETE, CHART, ADDRESS FORM, COUNTRY FORM, CITY FORM, STORE FORM, FILM FORM, CATEGORY FORM, ACTOR FORM, and CHART.
Step 4	Then, add four **JComboBoxes** to the form and set their corresponding **Variable Name** as **jcbPaymentID**, **jcbRentalID**, **jcbStaffID**, and **jcbCustomerID**.
Step 5	Add on **JCalendar** onto form and set its **Variable Name** as **jcPaymentDate**.
Step 6	Lastly, add a new **JTable** to the form set set its **Variable Name** as **jtPayment**. Then, right-click on it, then choose **Table Contents...** and set the number of columns to 31 and the number of rows to 100.
Step 7	In the driver class, **Sakila.java**, create a new object of **PaymentForm** class using its default constructor as shown in 16 – 17: 1 `package sakila;` 2 3 `public class Sakila {` 4 `public static void main(String[]` 5 `args) {` 6 `//...` 7 8 `//` 9 `Rental_Utils.create_rental_table();` 10 11

```
12  //
13  Rental_Utils.populate_rental_table();
14  //
15  Rental_Utils.read_rental_table();
16  //        RentalForm frm = new
17  RentalForm();
18  //        frm.setVisible(true);
19
    //
    Payment_Utils.create_payment_table();
    //
    Payment_Utils.populate_payment_table();
    //
    Payment_Utils.read_payment_table();
            PaymentForm frm = new
    PaymentForm();
            frm.setVisible(true);
        }
    }
```

Step 8

In **PaymentForm**'s constructor, invoke **setLookAndFeel()** to set the look and feel of the form as shown in line 17.

```
1   package sakila;
2
3   import java.awt.Toolkit;
4   import java.awt.event.ActionEvent;
5   import
6   java.awt.event.ActionListener;
7   import javax.swing.JButton;
8   import javax.swing.JComboBox;
9   import javax.swing.JMenuItem;
10  import javax.swing.JOptionPane;
11  import javax.swing.JPopupMenu;
12  import javax.swing.JTable;
13  import javax.swing.JTextField;
14
15  public class PaymentForm extends
16  javax.swing.JFrame {
17      public PaymentForm() {
18          initComponents();
19
20  Utility.setLookAndFeel(this);
        }
        //...
    }
```

Run the project to see the payment form as shown in Figure 16.1.

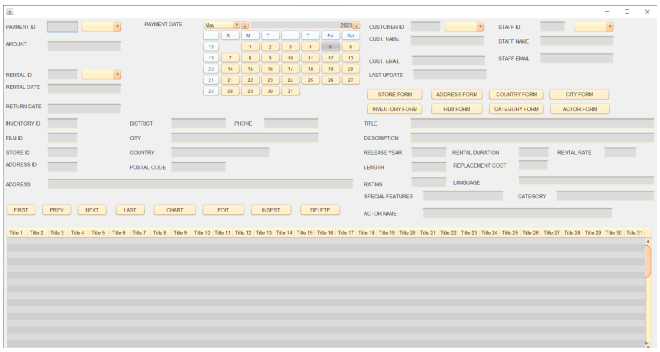


Figure 16.1 The layout of payment form

Step 9

In **PaymentForm.java**, define **getter()** method for every object in the form and place them outside and beneath its default constructor:

```
1       //Getter method for
2   jtfPaymentID
3       public JTextField
4   getJTFPaymentID(){
5           return
6   this.jtfPaymentID;
7       }
8
9       //Getter method for
10  jcbPaymentID
11      public JComboBox
12  getJCBPaymentID(){
13          return
14  this.jcbPaymentID;
15      }
16
17      //Getter method for
18  jtfAmount
19      public JTextField
20  getJTFAmount(){
21          return this.jtfAmount;
22      }
23
24      //Getter method for
25  jtfRentalID
26      public JTextField
27  getJTFRentalID(){
28          return this.jtfRentalID;
29      }
30
31      //Getter method for
32  jcbRentalID
33
```

```
34      public JComboBox
35  getJCBRentalID(){
36          return this.jcbRentalID;
37      }
38
39      //Getter method for
40  jcPaymentDate
41      public JCalendar
42  getJCPaymentDate(){
43          return
44  this.jcPaymentDate;
45      }
46
47      //Getter method for
48  jtfLastUpdate
49      public JTextField
50  getJTFLastUpdate(){
51          return
52  this.jtfLastUpdate;
53      }
54
55      //Getter method for
56  jtfInventoryID
57      public JTextField
58  getJTFInventoryID(){
59          return
60  this.jtfInventoryID;
61      }
62
63      //Getter method for
64  jtfFilmID
65      public JTextField
66  getJTFFilmID(){
67          return this.jtfFilmID;
68      }
69
70      //Getter method for
71  jtfRentalDate
72      public JTextField
73  getJTFRentalDate(){
74          return
75  this.jtfRentalDate;
76      }
77
78      //Getter method for
79  jtfReturnDate
80      public JTextField
81  getJTFReturnDate(){
82          return
83  this.jtfReturnDate;
84      }
85
86
87
```

```
88      //Getter method for
89  jtfStoreID
90      public JTextField
91  getJTFStoreID(){
92          return this.jtfStoreID;
93      }
94
95      //Getter method for
96  jtfAddressID
97      public JTextField
98  getJTFAddressID(){
99          return
100 this.jtfAddressID;
101     }
102
103     //Getter method for
104 jtfAddress
105     public JTextField
106 getJTFAddress(){
107         return this.jtfAddress;
108     }
109
110     //Getter method for
111 jtfDistrict
112     public JTextField
113 getJTFDistrict(){
114         return this.jtfDistrict;
115     }
116
117     //Getter method for jtfPhone
118     public JTextField
119 getJTFPhone(){
120         return this.jtfPhone;
121     }
122
123     //Getter method for jtfCity
124     public JTextField
125 getJTFCity(){
126         return this.jtfCity;
127     }
128
129     //Getter method for
130 jtfCountry
131     public JTextField
132 getJTFCountry(){
133         return this.jtfCountry;
134     }
135
136     //Getter method for
137 jtfPostalCode
138     public JTextField
139 getJTFPostalCode(){
140
141
```

```
142         return
143 this.jtfPostalCode;
144     }
145 
146     //Getter method for jtfTitle
147     public JTextField
148 getJTFTitle(){
149         return this.jtfTitle;
150     }
151 
152     //Getter method for
153 jtfDescription
154     public JTextField
155 getJTFDescription(){
156         return
157 this.jtfDescription;
158     }
159 
160     //Getter method for
161 jtfReleaseYear
162     public JTextField
163 getJTFReleaseYear(){
164         return
165 this.jtfReleaseYear;
166     }
167 
168     //Getter method for
169 jtfRentalDuration
170     public JTextField
171 getJTFRentalDuration(){
172         return
173 this.jtfRentalDuration;
174     }
175 
176     //Getter method for
177 jtfRentalRate
178     public JTextField
179 getJTFRentalRate(){
180         return
181 this.jtfRentalRate;
182     }
183 
184     //Getter method for
185 jtfLength
186     public JTextField
187 getJTFLength(){
188         return this.jtfLength;
189     }
190 
191     //Getter method for
192 jtfReplacementCost
193     public JTextField
194 getJTFReplacementCost(){
195 
```

```
196         return
197198 this.jtfReplacementCost;
199     }
200
201     //Getter method for
202 jtfSpecialFeatures
203     public JTextField
204 getJTFSpecialFeatures(){
205         return
206 this.jtfSpecialFeatures;
207     }
208
209     //Getter method for
210 jtfRating
211     public JTextField
212 getJTFRating(){
213         return this.jtfRating;
214     }
215
216     //Getter method for
217 jtfLanguageID
218     public JTextField
219 getJTFLanguageID(){
220         return
221 this.jtfLanguageName;
222     }
223
224     //Getter method for
225 jtfCustomerID
226     public JTextField
227 getJTFCustomerID(){
228         return
229 this.jtfCustomerID;
230     }
231
232     //Getter method for
233 jcbCustomerID
234     public JComboBox
235 getJCBCustomerID(){
236         return
237 this.jcbCustomerID;
238     }
239
240     //Getter method for
241 jtfCustFname
242     public JTextField
243 getJTFCustomerName(){
244         return
this.jtfCustomerName;
    }

    //Getter method for
jtfCustEmail
```

```
    public JTextField
getJTFCustEmail(){
        return
this.jtfCustEmail;
    }

    //Getter method for
jtfStaffID
    public JTextField
getJTFStaffID(){
        return this.jtfStaffID;
    }

    //Getter method for
jcbStaffID
    public JComboBox
getJCBStaffID(){
        return this.jcbStaffID;
    }

    //Getter method for
jtfStaffName
    public JTextField
getJTFStaffName(){
        return
this.jtfStaffName;
    }

    //Getter method for
jtfStaffEmail
    public JTextField
getJTFStaffEmail(){
        return
this.jtfStaffEmail;
    }

    //Getter method for
jtfCategory
    public JTextField
getJTFCategory(){
        return this.jtfCategory;
    }

    //Getter method for
jtfLanguageName
    public JTextField
getJTFLanguageName(){
        return
this.jtfLanguageName;
    }

    //Getter method for
jtfActorName
```

```
    public JTextField
getJTFActorName(){
        return
this.jtfActorName;
    }

    //Getter method for
jtPayment
    public JTable getJTPayment()
{
        return this.jtPayment;
    }

    //Getter method for jbEdit
    public JButton getJBEdit(){
        return this.jbEdit;
    }

    //Getter method for jbInsert
    public JButton getJBInsert()
{
        return this.jbInsert;
    }

    //Getter method for jbDelete
    public JButton getJBDelete()
{
        return this.jbDelete;
    }

    //Getter method for jbChart
    public JButton getJBChart(){
        return this.jbChart;
    }

    //Getter method for jbFirst
    public JButton getJBFirst(){
        return this.jbFirst;
    }

    //Getter method for jbPrev
    public JButton getJBPrev(){
        return this.jbPrev;
    }

    //Getter method for jbNext
    public JButton getJBNext(){
        return this.jbNext;
    }

    //Getter method for jbLast
    public JButton getJBLast(){
        return this.jbLast;
```

```
    }
```

POPULATING TABLE AND COMBOBOXES

POPULATING TABLE AND COMBOBOXES

Step 1

In **Payment_Utils.java**, add two new methods: **get_payment_list()** and **sh**

get_payment_list() takes three parameters: a **PaymentForm** object, an S establishes a database connection using **getConnection()**, creates a pre string, and sets a parameter value if the item string is not equal to "n iterates over the result set to create **Payment** objects, which it adds to an

show_table_payment() takes two parameters: a **PaymentForm** object an creates a **DefaultTableModel** with column headers, sets it as the model and iterates over the **Payment** objects in the **ArrayList** to add rows to t gets the values of its fields and populates an **Object** array with them, wh model.

```
1     private static ArrayList<Payment>
2  get_payment_list(PaymentForm frm, String sql, String item){
3          ArrayList<Payment> list = new ArrayList<>();
4
5          try(Connection conn = getConnection();
6              PreparedStatement ps = conn.prepareStatement(sql)){
7              if (item.equalsIgnoreCase("none")==false) {
8                   ps.setString(1,item);
9              }
10             ResultSet rs = ps.executeQuery();
11             Payment obj;
12
13             while(rs.next()){
14                 //Using thirty-one-params constructor
15                 obj = new Payment(rs.getInt("payment_id"),
16                    rs.getDouble("amount"),
17                    rs.getDate("payment_date"),
18                    rs.getInt("rental_id"),
19                    rs.getDate("rental_date"),
20                    rs.getInt("inventory_id"),
21                    rs.getInt("customer_id"),
22                    rs.getDate("return_date"),
23                    rs.getInt("staff_id"),
24                    rs.getTimestamp("last_update"),
25                    rs.getInt("film_id"),
26                    rs.getString("title"),
27                    rs.getString("description"),
28                    rs.getInt("release_year"),
```

```
                    rs.getInt("rental_duration"),
                    rs.getDouble("rental_rate"),
                    rs.getInt("length"),
                    rs.getDouble("replacement_cost"),
                    rs.getString("rating"),
                    rs.getString("special_features"),
                    rs.getString("language_name"),
                    rs.getString("category_name"),
                    rs.getString("actor_name"),
                    rs.getString("customer_name"),
                    rs.getString("customer_email"),
                    rs.getString("staff_name"),
                    rs.getString("staff_email"),
                    rs.getString("store_city"),
                    rs.getString("store_country"),
                    rs.getString("store_district"),
                    rs.getString("store_postal_code"));

                list.add(obj);
            }
        }catch (SQLException ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
                "ERROR",JOptionPane.ERROR_MESSAGE);
        }
        return list;
    }

    private static void show_table_payment(PaymentForm frm,
ArrayList<Payment> list) throws SQLException{
        DefaultTableModel model = new DefaultTableModel(0,0);

        String header[] = {"Payment ID", "Amount", "Payment
Date",
            "Rental ID", "Rental Date", "Inventory ID",
"Customer ID", "Return Date",
            "Staff ID", "Film ID", "Title", "Description",
"Release Year",
            "Rental Duration", "Rental Rate", "Length",
"Replacement Cost",
            "Rating", "Special Features", "Language",
"Category", "Actor Name",
            "Customer Name", "Customer Email", "Staff Name",
"Staff Email",
            "Store City", "Store Country", "Store District",
"Store Postal Code",
            "Last Update"};


model.setColumnIdentifiers(set_column_header(frm.getJTPayment(),
header));
        frm.getJTPayment().setModel(model);

        Object[] row = new Object[31];

```

```
        for(int i=0; i<list.size(); i++){
            row[0] = list.get(i).getPaymentID();
            row[1] = list.get(i).getAmount();
            row[2] = list.get(i).getPaymentDate();
            row[3] = list.get(i).getRentalID();
            row[4] = list.get(i).getRentalRate();
            row[5] = list.get(i).getInventoryID();
            row[6] = list.get(i).getCustomerID();
            row[7] = list.get(i).getReturnDate();
            row[8] = list.get(i).getStaffID();
            row[9] = list.get(i).getFilmID();
            row[10] = list.get(i).getTitle();
            row[11] = list.get(i).getDescription();
            row[12] = list.get(i).getReleaseYear();
            row[13] = list.get(i).getRentalDuration();
            row[14] = list.get(i).getRentalRate();
            row[15] = list.get(i).getLength();
            row[16] = list.get(i).getReplacementCost();
            row[17] = list.get(i).getRating();
            row[18] = list.get(i).getSpecialFeatures();
            row[19] = list.get(i).getLanguageName();
            row[20] = list.get(i).getCategory();
            row[21] = list.get(i).getActorName();
            row[22] = list.get(i).getCustomerName();
            row[23] = list.get(i).getCustomerEmail();
            row[24] = list.get(i).getStaffName();
            row[25] = list.get(i).getStaffEmail();
            row[26] = list.get(i).getStoreCity();
            row[27] = list.get(i).getStoreCountry();
            row[28] = list.get(i).getStoreDistrict();
            row[29] = list.get(i).getStorePostalCode();
            row[30] = list.get(i).getLastUpdate();

            model.addRow(row);
        }
    }
```

Step 2

In **Payment_Utils.java**, define **refresh_controls()** method. The purpose c on the **PaymentForm**. The method performs the following operations:

1. Centers the **PaymentForm** on the screen and sets its title to "PAY
2. Calls the **table_renderer()** method to apply alternating row colors
3. Calls the **get_payment_list()** and **show_table_payment()** meth **JTable** with data retrieved from the database.
4. Calls the **populate_combobox()** method four times to po **PaymentForm** with data retrieved from the database. Each **JCom** table (rental, payment, customer, staff) and displays the values of a
5. If any of the database operations in the method result in an displayed to the user via a **JOptionPane**.

```
    public static void
refresh_controls(PaymentForm frm){

frm.setLocationRelativeTo(null);
        frm.setTitle("PAYMENT FORM");

        //Shows the content of
payment table and populates combobox
        try{
            //Makes alternating color
for table rows

table_renderer(frm.getJTPayment());

            //Populates table
            ArrayList<Payment> list =
get_payment_list(frm,
Query_Payment.get_sql_payment_joint()
+ " ORDER BY payment_id", "none");
            show_table_payment(frm,
list);

            //Populates jcbRentalID
            String sql_rent_id =
 "SELECT rental_id FROM rental ORDER
BY rental_id";

populate_combobox(sql_rent_id,
frm.getJCBRentalID(), frm);

            //Populates jcbPaymentID
            String sql_pay_id =
 "SELECT payment_id FROM payment
ORDER BY payment_id";

populate_combobox(sql_pay_id,
frm.getJCBPaymentID(), frm);

            //Populates jcbCustomerID
            String sql_cust_id =
 "SELECT customer_id FROM customer
ORDER BY customer_id";

populate_combobox(sql_cust_id,
frm.getJCBCustomerID(), frm);

            //Populates jcbStaffID
            String sql_stf_id =
 "SELECT staff_id FROM staff ORDER BY
staff_id";

populate_combobox(sql_stf_id,
frm.getJCBStaffID(), frm);
```

```
        }catch (SQLException ex){

JOptionPane.showMessageDialog(frm,
ex.getMessage(),

 "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }
```

Step 3

In **PaymentForm**'s default constructor, the **Payment_Utils.refresh_c** populates the controls in the **PaymentForm** with data from a database from the **Payment_Utils** class.

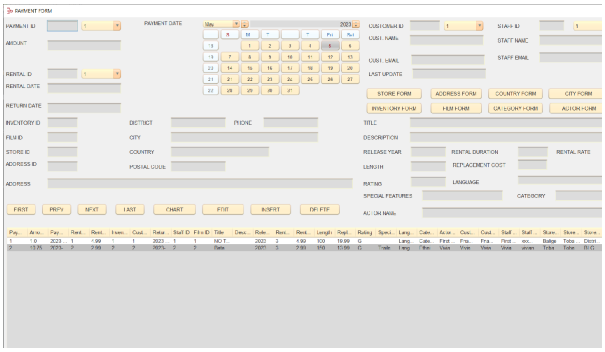


Figure 16.2 The content of **payment** table displaye

The **this.setIconImage()** method sets the icon of the **Pa** **this.setDefaultCloseOperation(this.HIDE_ON_CLOSE)** method se **PaymentForm** to hide the form instead of exiting the application when the

```
1  package sakila;
2  import java.awt.Toolkit;
3  import java.awt.event.ActionEvent;
4  import java.awt.event.ActionListener;
5  import java.io.IOException;
6  import java.text.ParseException;
7  import java.util.logging.Level;
8  import java.util.logging.Logger;
9  import javax.swing.JButton;
10 import javax.swing.JComboBox;
11 import javax.swing.JLabel;
12 import javax.swing.JTable;
13 import javax.swing.JTextField;
14
15 public class PaymentForm extends javax.swing.JFrame {
16     public PaymentForm() {
17         initComponents();
18         Utility.setLookAndFeel(this);
19
```

```
20          Payment_Utils.refresh_controls(this);
21
22   this.setIconImage(Toolkit.getDefaultToolkit().getImage(getClass().
23 ;
24          this.setDefaultCloseOperation(this.HIDE_ON_CLOSE);
25      }
26      //...
    }
```

Step 4

Run the project to see the content of **payment** table displayed in **jtPayme**

If you use the data from **Sakila** MySQL database available in the inte **payment** table displayed in **jtPayment** as shown in Figure 16.3.

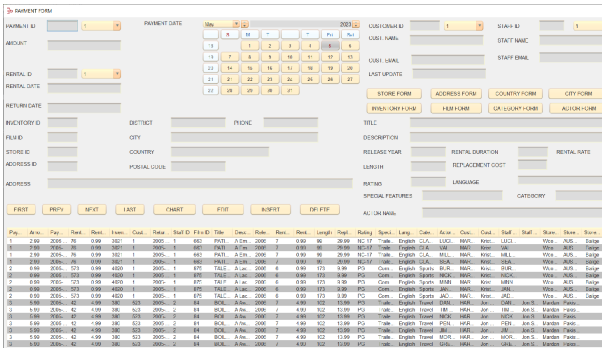


Figure 16.3 The the content of **payment** table in original **Sakila** databas **jtPayment**

DISPLAYING AND NAVIGATING DATA ROW BY ROW

DISPLAYING AND NAVIGATING DATA ROW BY ROW

Step 1

In **Payment_Utils**, define two new methods named **clear_staff_controls(** and **display_staff_data()**.

The **clear_staff_controls()** method is a private method that takes **PaymentForm** object frm as input and clears the text fields for staff ID staff name, and staff email. This method is called by th **display_staff_data()** method when no matching data is found in the resul set returned by the SQL query.

The **display_staff_data()** method is a private generic method that takes **PaymentForm** object frm, an SQL query string sql, and a generic iten item as input. This method is responsible for executing an SQL query wit a parameterized input value, populating the text fields in th

PaymentForm object with the data returned from the query, and selecting the corresponding value in the **JComboBox** for staff ID.

The try block in the **display_staff_data()** method sets up a connection to the database and prepares a **PreparedStatement** object with the provided SQL query string. The input parameter value is set to the generic item using the **setObject()** method of the **PreparedStatement** object. The query is executed using the **executeQuery()** method, which returns a **ResultSet** object containing the query results.

The if statement checks if the result set is empty, and if so, calls the **clear_staff_controls()** method to clear the text fields in the **PaymentForm** object and returns. If there is data in the result set, the do while loop iterates through each row of data and populates the text fields in the **PaymentForm** object with the corresponding values. The **find_combo_value_selected()** method is called to select the corresponding value in the **JComboBox** for staff ID.

Finally, the **ResultSet** object and the **PreparedStatement** object are closed using the **close()** method in the finally block. If an SQL exception is caught, an error message dialog box is displayed using the **JOptionPane** class.

```
1     private static void clear_staff_controls(PaymentForm frm){
2         frm.getJTFStaffID().setText("");
3         frm.getJTFStaffName().setText("");
4         frm.getJTFStaffEmail().setText("");
5     }
6
7    //Displays staff data result row by row
8     private static <T> void display_staff_data(PaymentForm frm,
9 String sql, T item){
10        try(Connection conn = getConnection()){
11            PreparedStatement ps = conn.prepareStatement(sql);
12            ps.setObject(1,item);
13            ResultSet rs = ps.executeQuery();
14
15            if (!rs.next()) {
16                // no row found, clear the form fields
17                clear_staff_controls(frm);
18                return;
19            }
20
21            do{
22
23 frm.getJTFStaffID().setText(String.valueOf(rs.getInt("staff_id")));
24
25 frm.getJTFStaffName().setText(rs.getString("first_name") + " " +
26 rs.getString("last_name"));
27
```

```
28
29 frm.getJTFStaffEmail().setText(rs.getString("email"));
30
31                 // Determines item selected from jcbStaffID
32                 find_combo_value_selected(frm.getJCBStaffID(),
33 rs.getInt("staff_id"));
34
35             }while(rs.next());
36
37             rs.close();
38             ps.close();
39         }catch(SQLException ex){
40             JOptionPane.showMessageDialog(frm, ex.getMessage(),
                       "ERROR",JOptionPane.ERROR_MESSAGE);
           }
    }
```

Step 2

Still in the same class, define another method named **jcbStaff_handler()**.

```
1     public static void
2 jcbStaff_handler(PaymentForm frm) {
3         Object item =
4 frm.getJCBStaffID().getSelectedItem();
          display_staff_data(frm,
Query_Staff.get_sql_id(), item);
      }
```

Step 3

In **PaymentForm**, double click on **jcbStaffID** combobox and define it corresponding event handler as follows:

```
1     private void
2 jcbStaffIDActionPerformed(java.awt.event.ActionEvent
3 evt) {
          Payment_Utils.jcbStaff_handler(this);
      }
```

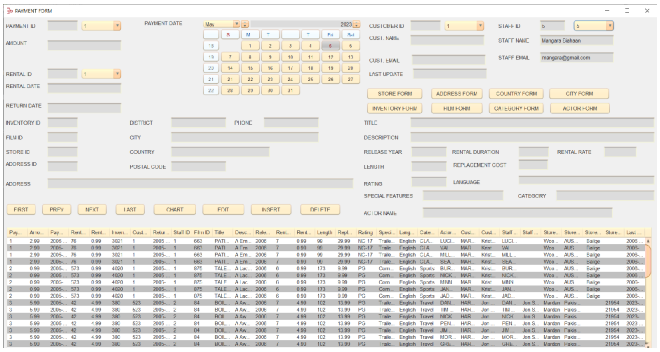


Figure 16.4 Displaying row by row the content of **staff** table

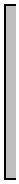
Step 4	Run the project. Choose one of items in **jcbStaffID** combobox to see rov by row the content of **staff** table as shown in Figure 16.4.
Step 5	In **Payment_Utils**, define two new methods name **clear_customer_controls()** and **display_customer_data()**. Th **clear_customer_controls(PaymentForm frm)** method clears th customer ID, customer name, and customer email fields on the rental form This method is called before displaying new customer data on the renta form. The **display_customer_data(PaymentForm frm, String sql, T item** method displays the customer data from the database on the rental form. I takes in three parameters - the rental form object, a SQL query to retriev the customer data, and the customer ID selected from the **JComboBox** The method retrieves the customer data for the selected customer ID an populates the customer ID, customer name, and customer email fields o the rental form. The method also sets the selected item in the **JComboBo** to the selected customer ID.

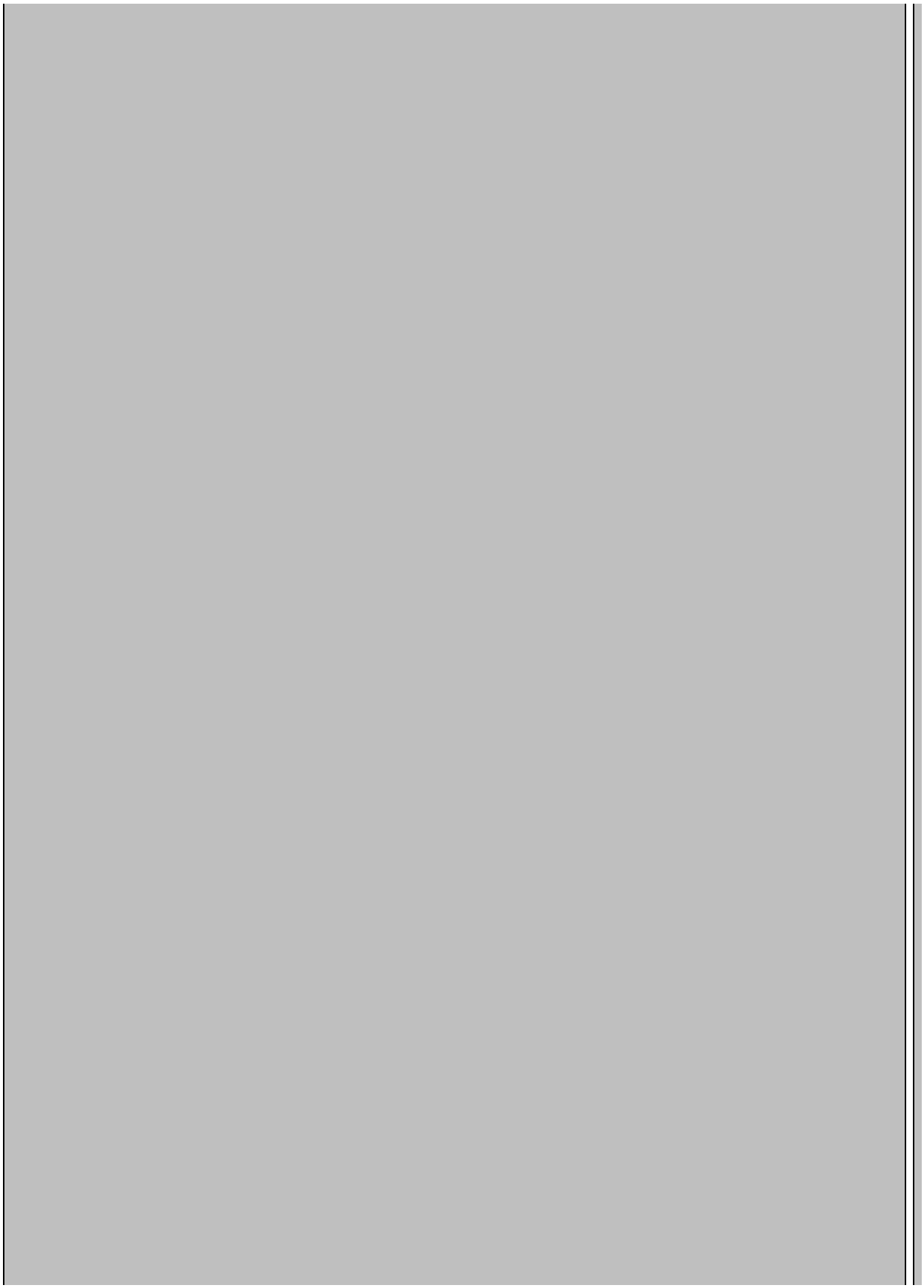
```
1     private static void
2  clear_customer_controls(PaymentForm
3  frm){
4
5  frm.getJTFCustomerID().setText("");
6
7  frm.getJTFCustomerName().setText("");
8
9  frm.getJTFCustEmail().setText("");
10     }
11
12    //Displays customer data result
13 row by row
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
```

```
31
32
33
34
35
36
37
38
39
40
```

```
    private static <T> void display_customer_data(PaymentForm frm, String
sql, T item){
        try(Connection conn = getConnection()){
            PreparedStatement ps = conn.prepareStatement(sql);
            ps.setObject(1,item);
            ResultSet rs = ps.executeQuery();

            if (!rs.next()) {
                // no row found, clear the form fields
                clear_customer_controls(frm);
                return;
            }

            do{

frm.getJTFCustomerID().setText(String.valueOf(rs.getInt("customer_id")));

frm.getJTFCustomerName().setText(rs.getString("first_name") + " " +
rs.getString("last_name"));
                frm.getJTFCustEmail().setText(rs.getString("email"));

                // Determines item selected from jcbCustomerID
                find_combo_value_selected(frm.getJCBCustomerID(),
rs.getInt("customer_id"));

            }while(rs.next());

            rs.close();
            ps.close();
        }catch(SQLException ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }
```

Step 6

Step 7	I e 
Step 8	F c
Step 9	I **c** T t t T c I e i r q c s I **J** 

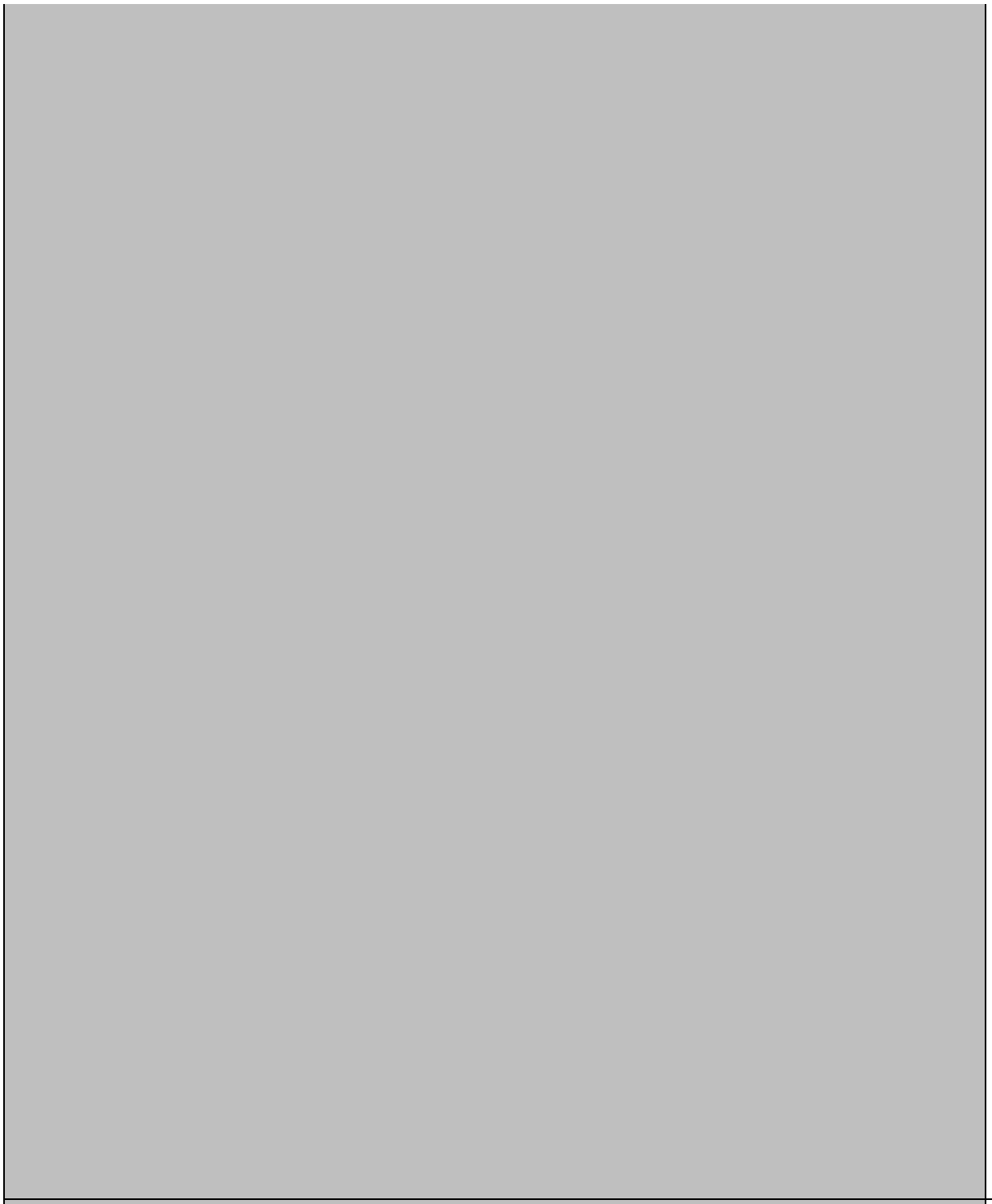


Step 10

Step 11	
Step 12	
Step 13	

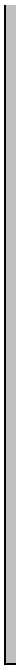
Step 14	S r f **J** c s T t u t e
Step 15	I e

Step 16	
Step 17	

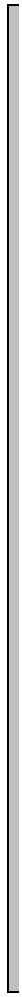


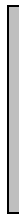
Step 18

T
e

	
Step 19	
Step 20	

Step 21





Step 22

UPDATING RECORD
UPDATING RECORD

Step 1

In **Payment_Utils** class, define a new method named **update_row_by_payment_id()**. This method updates a row of data in the **payment** table by **payment_id**. It takes in a **PaymentForm** object, and various parameters such as pay_id, cust_id, staff_id, rental_id, amount, and pay_date that are used to create a **Payment** object. It then uses a prepared statement to update the customer_id, staff_id, rental_id, amount, and payment_date fields in the payment table for the given payment_id.

If the **payment_id** is not found, it displays an error message using a **JOptionPane**. If there is an SQL or **NumberFormatException** exception, it logs the error using the **Logger** class and displays an error message using a **JOptionPane**.

```
1   //Updates row of data in payment tabel by payment_id
2   public static void update_row_by_payment_id(PaymentForm
3   frm, int pay_id, int cust_id,
4           int staff_id, int rental_id, double amount, Date
5   pay_date) throws SQLException{
6       Connection conn = getConnection();
7       ResultSet rs = null;
8       String query_id = "SELECT payment_id FROM payment WHERE
9   payment_id = ?";
10      String update_query = """
11          UPDATE payment SET customer_id = ?, staff_id = ?,
12  rental_id = ?,
13          amount = ?, payment_date = ?  WHERE payment_id =
14  ?""";
15      try(PreparedStatement idPs =
16  conn.prepareStatement(query_id,
17
18  ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_READ_ONLY);
19      PreparedStatement updatePS =
20  conn.prepareStatement(update_query,
21              ResultSet.TYPE_SCROLL_SENSITIVE,
22  ResultSet.CONCUR_UPDATABLE))
23      {
24          idPs.setInt(1,pay_id);
25          if(!idPs.execute()){
26              String message = "Can't find payment_id " +
27  pay_id;
28
29              JOptionPane.showMessageDialog(frm, message,
30                  "ERROR",JOptionPane.ERROR_MESSAGE);
```

```
            } else{
                rs = idPs.getResultSet();
                rs.next();

                //Creates a Payment object using seven-params
constructor
                Payment obj = new Payment(pay_id, cust_id,
staff_id, rental_id, amount, pay_date, new
Timestamp(System.currentTimeMillis()));
                updatePS.setInt(1, obj.getCustomerID());
                updatePS.setInt(2, obj.getStaffID());
                updatePS.setInt(3, obj.getRentalID());
                updatePS.setDouble(4, obj.getAmount());
                updatePS.setDate(5, obj.getPaymentDate());
                updatePS.setInt(6, obj.getPaymentID());

                updatePS.executeUpdate();
                rs.close();
                updatePS.close();
                idPs.close();
                conn.close();
            }
        }catch(SQLException ex){

Logger.getLogger(PaymentForm.class.getName()).log(Level.SEVERE,
"Error updating payment data", ex);
            String message = "Error updating payment data: " +
ex.getMessage();
            String stackTrace =
Arrays.toString(ex.getStackTrace());
            JOptionPane.showMessageDialog(null, message +
"\n\n" + stackTrace, "ERROR", JOptionPane.ERROR_MESSAGE);
        }catch(java.lang.NumberFormatException ex){

Logger.getLogger(PaymentForm.class.getName()).log(Level.SEVERE,
"Invalid Input", ex);
            String message = "Invalid Input: " +
ex.getMessage();
            String stackTrace =
Arrays.toString(ex.getStackTrace());
            JOptionPane.showMessageDialog(null, message +
"\n\n" + stackTrace, "ERROR", JOptionPane.ERROR_MESSAGE);
        }
    }
```

Step 2	Then in the same class, define a new method **read_inputs()**. It reads user input data from a **PaymentForm** object and validates the input. The method returns a **HashMap** object containing the validated input data.

The **PaymentForm** object is passed as a parameter to the method, and several fields of the **PaymentForm** object are accessed to retrieve user input data. These fields include **JComboBox** objects for customer ID, rental ID, staff ID, and payment ID, as well as a **JTextField** object for the payment amount, and a **JDateChooser** object for the payment date.

The method first retrieves the selected date from the **JDateChooser** object and formats it as a string using a **SimpleDateFormat** object. Then, it validates the user input for each ID field by parsing the input as an integer and checking that the resulting value is greater than zero. If the input is invalid, the method throws an exception and displays an error message to the user.

Finally, the validated input data is stored in a **HashMap** object with keys corresponding to the field names, and the **HashMap** is returned by the method.

```
1   private static HashMap<String, String>
2   read_inputs(PaymentForm frm) {
3           HashMap<String, String> input_data = new
4   HashMap<>();
5           String cust_id =
6   String.valueOf(frm.getJCBCustomerID().getSelectedItem());
7           String rental_id =
8   String.valueOf(frm.getJCBRentalID().getSelectedItem());
9           String staff_id =
10  String.valueOf(frm.getJCBStaffID().getSelectedItem());
11          String pay_id =
12  String.valueOf(frm.getJCBPaymentID().getSelectedItem());
13          String amount = frm.getJTFAmount().getText();
14
15          //Gets the selected date from the
16  getJCPaymentDate()
17          java.util.Date pay_date =
18  frm.getJCPaymentDate().getDate();
19          //Create a SimpleDateFormat object with the
20  desired date format
21          SimpleDateFormat sdf = new
22  SimpleDateFormat("yyyy-MM-dd");
23          //Formats the date as a String using the
24  SimpleDateFormat object
25          String str_pay_date = sdf.format(pay_date);
26
27          // Validate user input
28          int rental_id_int = 0;
29          try {
30              rental_id_int = Integer.parseInt(rental_id);
31              if (rental_id_int <= 0) {
32                  throw new
33  IllegalArgumentException("Rental ID cannot be negative or
34  zero");
```

```
            }
        } catch (NumberFormatException ex) {
            JOptionPane.showMessageDialog(frm, "Invalid
Rental ID: " + rental_id,
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        } catch (IllegalArgumentException ex) {
            JOptionPane.showMessageDialog(frm,
ex.getMessage(),
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        }

        int staff_id_int = 0;
        try {
            staff_id_int = Integer.parseInt(staff_id);
            if (staff_id_int <= 0) {
                throw new IllegalArgumentException("Staff
ID cannot be negative or zero");
            }
        } catch (NumberFormatException ex) {
            JOptionPane.showMessageDialog(frm, "Invalid
Staff ID: " + staff_id,
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        } catch (IllegalArgumentException ex) {
            JOptionPane.showMessageDialog(frm,
ex.getMessage(),
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        }

        int cust_id_int = 0;
        try {
            cust_id_int = Integer.parseInt(cust_id);
            if (cust_id_int <= 0) {
                throw new
IllegalArgumentException("Customer ID cannot be negative
or zero");
            }
        } catch (NumberFormatException ex) {
            JOptionPane.showMessageDialog(frm, "Invalid
Customer ID: " + cust_id,
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        } catch (IllegalArgumentException ex) {
            JOptionPane.showMessageDialog(frm,
ex.getMessage(),
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        }

        int pay_id_int = 0;

```

```
89          try {
90              pay_id_int = Integer.parseInt(pay_id);
91              if (pay_id_int <= 0) {
92                  throw new
93  IllegalArgumentException("Payment ID cannot be negative
    or zero");
                }
            } catch (NumberFormatException ex) {
                JOptionPane.showMessageDialog(frm, "Invalid
    Payment ID: " + pay_id,
                "Error", JOptionPane.ERROR_MESSAGE);
                throw ex;
            } catch (IllegalArgumentException ex) {
                JOptionPane.showMessageDialog(frm,
    ex.getMessage(),
                "Error", JOptionPane.ERROR_MESSAGE);
                throw ex;
            }

            input_data.put("cust_id", cust_id);
            input_data.put("pay_id", pay_id);
            input_data.put("staff_id", staff_id);
            input_data.put("rental_id", rental_id);
            input_data.put("amount", amount);
            input_data.put("str_pay_date", str_pay_date);

            return input_data;
        }
```

Step 3

Still in the same class, define another method named **edit_actual()**. It updatesa row in **payment** table based on the input data provided by the user in the **PaymentForm** object. It first reads the input data by calling the **read_inputs()** method and then extracts the required values from the **HashMap**. It then parses the date string into a java.sql.Date object and calls the **update_row_by_payment_id()** method to update the corresponding row in the table with the new values. Finally, it refreshes the controls on the form by calling the **refresh_controls()** method.

It catches any **SQLException** that may occur during the update process and displays an error message dialog box with the corresponding error message.

```
1       private static void
2   edit_actual(PaymentForm frm) throws
3   ParseException{
4           try{
5               HashMap<String, String> input_data
6   = read_inputs(frm);
7
```

```
            int cust_id =
Integer.parseInt(input_data.get("cust_id"));
            int pay_id =
Integer.parseInt(input_data.get("pay_id"));
            int staff_id =
Integer.parseInt(input_data.get("staff_id"));
            int rental_id =
Integer.parseInt(input_data.get("rental_id"));
            String str_pay_date =
input_data.get("str_pay_date");
            double amount =
Double.parseDouble(input_data.get("amount"));

            SimpleDateFormat dateFormat = new
SimpleDateFormat("yyyy-MM-dd");
            java.util.Date pay_date =
dateFormat.parse(str_pay_date);
            java.sql.Date pay_sql_date = new
java.sql.Date(pay_date.getTime());

            update_row_by_payment_id(frm,
pay_id, cust_id, staff_id, rental_id, amount,
pay_sql_date);

            //Refreshes all objects on form
            refresh_controls(frm);

        }catch(SQLException ex){
            JOptionPane.showMessageDialog(frm,
ex.getMessage(),

 "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }
```

Step 4

Lastly, define two new methods named **enable_controls()** and **edit_handler()**. The **enable_controls()** method takes in two parameters, a boolean state and an instance of **PaymentForm**. It sets the enabled state of various controls on the form based on the value of the state parameter.

The **edit_handler()** method handles the edit button click event on the form. If the edit button's text is "EDIT", it changes the text to "CONFIRM" and disables all controls on the form. If the text is "CONFIRM", it performs the actual editing by calling the **edit_actual()** method, changes the button's text back to "EDIT" and enables all controls on the form.

```
    private static void enable_controls(boolean state,
```

```
2   PaymentForm frm){
3           frm.getJBFirst().setEnabled(state);
4           frm.getJBPrev().setEnabled(state);
5           frm.getJBNext().setEnabled(state);
6           frm.getJBLast().setEnabled(state);
7           frm.getJBInsert().setEnabled(state);
8           frm.getJBDelete().setEnabled(state);
9           frm.getJTFAddressID().setEnabled(state);
10          frm.getJTFStoreID().setEnabled(state);
11          frm.getJTFAddress().setEnabled(state);
12          frm.getJTFDistrict().setEnabled(state);
13          frm.getJTFPhone().setEnabled(state);
14          frm.getJTFPostalCode().setEnabled(state);
15          frm.getJTFCity().setEnabled(state);
16          frm.getJTFCountry().setEnabled(state);
17          frm.getJTFStaffID().setEnabled(state);
18          frm.getJTFStaffName().setEnabled(state);
19          frm.getJTFStaffEmail().setEnabled(state);
20          frm.getJTFLastUpdate().setEnabled(state);
21          frm.getJTFFilmID().setEnabled(state);
22          frm.getJTFCustEmail().setEnabled(state);
23          frm.getJTFCustomerName().setEnabled(state);
24          frm.getJTFTitle().setEnabled(state);
25          frm.getJTFDescription().setEnabled(state);
26          frm.getJTFReleaseYear().setEnabled(state);
27          frm.getJTFRentalDuration().setEnabled(state);
28          frm.getJTFRentalRate().setEnabled(state);
29          frm.getJTFLength().setEnabled(state);
30          frm.getJTFReplacementCost().setEnabled(state);
31          frm.getJTFSpecialFeatures().setEnabled(state);
32          frm.getJTFCategory().setEnabled(state);
33          frm.getJTFLanguageName().setEnabled(state);
34          frm.getJTFActorName().setEnabled(state);
35          frm.getJTFInventoryID().setEnabled(state);
36          frm.getJTFReturnDate().setEnabled(state);
37          frm.getJTFRentalDate().setEnabled(state);
38      }
39
40      public static void edit_handler(PaymentForm frm){
41          if(frm.getJBEdit().getText().equals("EDIT")){
42              frm.getJBEdit().setText("CONFIRM");
43
44              // Disables controls
45              enable_controls(false, frm);
46          }
47
48          else {
49              try {
50                  frm.getJBEdit().setText("EDIT");
51
52                  // Actual editing
53                  edit_actual(frm);
54
55                  //Enables controls
```

```
                    enable_controls(true, frm);
                } catch (ParseException ex) {

Logger.getLogger(Payment_Utils.class.getName()).log(Level.SEVERE,
null, ex);
                }
            }
        }
```

Step 5

Then, double click on **jbEdit** button on **PaymentForm** to define its event listener:

```
    private void
jbEditActionPerformed(java.awt.event.ActionEvent
evt) {
        Payment_Utils.edit_handler(this);
    }
```

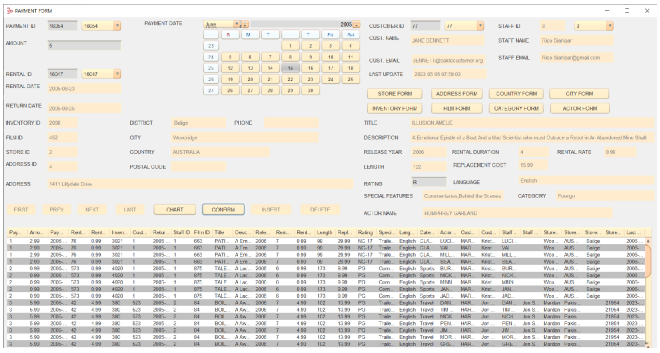


Figure 16.9 The payment form is in editing state

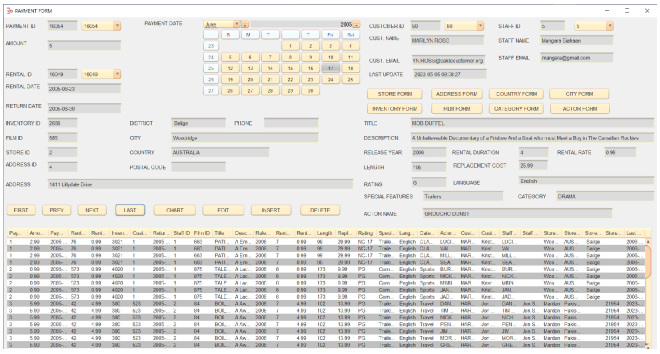


Figure 16.10 The edited row had been saved into database

Step 6

Run the project. Choose **payment_id** using **jcbPaymentID** combobox. Or, you can choose one of rows in **jtPayment** (in this case, **rental_id = 16054**). Then, click on EDIT button as shown in Figure 16.9.

	Edit any field you want (including choosing payment date **JCalendar**). Then, click on CONFIRM button. The edited row had been saved into **rental** table as shown in Figure 16.10.

INSERTING NEW RECORD
INSERTING NEW RECORD

Step 1

In **Payment_Utils** class, define a method named **insert_row()**. It inserts a new row into the **payment** table. It starts by calling the **read_inputs()** method to get the values entered by the user in the form. Then, it parses the necessary values (customer ID, staff ID, rental ID, payment amount, and payment date) and creates a new **Payment** object using these values.

After that, the SQL insert statement is created with placeholders for each parameter. The code then opens a connection to the database and prepares the statement using **PreparedStatement** to avoid SQL injection attacks. The parameter values are set using the methods provided by **PreparedStatement** object, and then the statement is executed using **executeUpdate()** method to insert the new row into the payment table.

If an exception occurs during the database operation, the code logs the error message and displays an error message to the user using **JOptionPane.showMessageDialog()**.

```
1    //Inserts new row into payment table
2    private static void insert_row(PaymentForm frm) throws
3  SQLException, ParseException{
4          HashMap<String, String> input_data = read_inputs(frm);
5          int cust_id =
6  Integer.parseInt(input_data.get("cust_id"));
7          int staff_id =
8  Integer.parseInt(input_data.get("staff_id"));
9          int rental_id =
10 Integer.parseInt(input_data.get("rental_id"));
11         String str_pay_date = input_data.get("str_pay_date");
12         double amount =
13 Double.parseDouble(input_data.get("amount"));
14
15         SimpleDateFormat dateFormat = new
16 SimpleDateFormat("yyyy-MM-dd");
17         java.util.Date pay_date =
18 dateFormat.parse(str_pay_date);
19         java.sql.Date pay_sql_date = new
20 java.sql.Date(pay_date.getTime());
```

```
21
22          // SQL insert statement
23          String sql = """
24                  INSERT INTO payment(customer_id, staff_id,
25  rental_id,
26                      amount, payment_date) VALUES(?, ?, ?, ?, ?)""";
27
28          try(Connection conn = getConnection();
29           PreparedStatement pstmt = conn.prepareStatement(sql)){
30
31              //Creates a Payment object six-params constructor
32              Payment obj = new Payment(cust_id, staff_id,
33  rental_id, amount, pay_sql_date, new
34  Timestamp(System.currentTimeMillis()));
35              pstmt.setInt(1,obj.getCustomerID());
36              pstmt.setInt(2,obj.getStaffID());
37              pstmt.setInt(3,obj.getRentalID());
38              pstmt.setDouble(4,obj.getAmount());
39              pstmt.setDate(5, obj.getPaymentDate());
40
                //Executes the sql insert statement
                pstmt.executeUpdate();
            } catch (SQLException ex) {

    Logger.getLogger(PaymentForm.class.getName()).log(Level.SEVERE,
    "Database error", ex);
                JOptionPane.showMessageDialog(frm, "Error: Database
    error\n" + ex.getMessage());
            }
        }
```

Step 2

Still in **Payment_Utils.java**, define **insert_actual()** and **insert_handler()** methods. The first method, **insert_actual()**, is a private static method that is called by the **insert_handler()** method. It takes a **PaymentForm** object as its parameter and is responsible for inserting a new payment record into the database. It does so by calling the **insert_row()** method, which inserts the new record into the payment table. If an exception is caught during the insertion process, it displays an error message in a pop-up window using the **JOptionPane** class.

The **insert_handler()** method is a public static method that is also part of the **Payment_Utils** class. It takes a **PaymentForm** object as its parameter and is responsible for handling the insertion of a new payment record. If the text of the **jbInsert** button on the form is "INSERT", it changes the text to "CONFIRM" and disables the **jbEdit** button and several other controls on the form using the **enable_controls()** method. It also clears the values of some text fields on the form.

If the text of the **jbInsert** button is "CONFIRM", the method calls the **insert_actual()** method to actually insert the new record into the database. If an exception occurs during the insertion, it catches the exception and displays an error message using the **JOptionPane** class. Finally, the method re-enables the **jbEdit** button and the disabled controls on the form using the **enable_controls()** method.

```
1   private static void insert_actual(PaymentForm frm) throws
2  ParseException{
3          try{
4              insert_row(frm);
5
6              //Refreshes table and comboboxes
7              refresh_controls(frm);
8
9          }catch(SQLException ex){
10             JOptionPane.showMessageDialog(frm, ex.getMessage(),
11                     "ERROR",JOptionPane.ERROR_MESSAGE);
12         }
13     }
14
15     public static void insert_handler(PaymentForm frm){
16         if(frm.getJBInsert().getText().equals("INSERT")){
17             frm.getJBInsert().setText("CONFIRM");
18
19             //Disables jbEdit
20             frm.getJBEdit().setEnabled(false);
21
22             // Disables controls
23             enable_controls(false, frm);
24             frm.getJCBPaymentID().setEnabled(false);
25
26             // Clears controls
27             frm.getJTFAmount().setText("");
28             frm.getJTFLastUpdate().setText("");
29
30             // Enables
31             frm.getJBInsert().setEnabled(true);
32         }
33
34         else {
35             frm.getJBInsert().setText("INSERT");
36
37             try {
38                 // Actual insertion
39                 insert_actual(frm);
40             } catch (ParseException ex) {
41
42 Logger.getLogger(Payment_Utils.class.getName()).log(Level.SEVERE,
43 null, ex);
44             }
45
```

```
46                 //Enables jbEdit
47                 frm.getJBEdit().setEnabled(true);
48
49                 //Enables controls
50                 enable_controls(true, frm);
51                 frm.getJCBPaymentID().setEnabled(true);
             }
         }
```

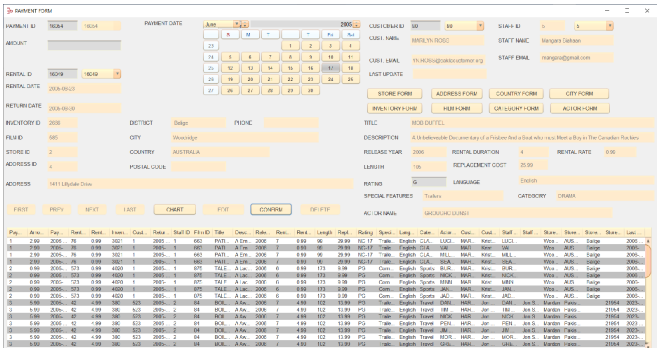


Figure 16.11 When user clicks on INSERT button, the payment form will be in state of insertion

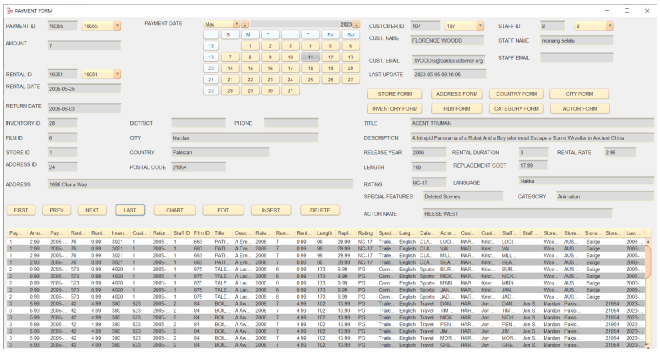


Figure 16.12 The new data had been saved into **payment** table

Step 3 — In **PaymentForm.java**, double click on INSERT button to create its event listener:

```
1     private void
2 jbInsertActionPerformed(java.awt.event.ActionEvent
3 evt) {
          Payment_Utils.insert_handler(this);
      }
```

Step 4 — Run the project. Click on INSERT button. You will see the state of payment form when insertion is in progress as shown in Figure 16.11.

	Then, fill in all fields. Then, click CONFIRM button to save the new record into **payment** table as shown in Figure 16.12.

DELETING RECORD
DELETING RECORD

Step 1	Then in **Payment_Utils** class, define **delete_handler()** metho **delete_handler()** method is a static method that takes a **PaymentForm** o its parameter. This method is responsible for handling the deletion of a data from the database table. The method first displays a confirmation dialog box to the user, asking if sure they want to delete the selected row of data. If the user clicks tl button, the method proceeds with the deletion of the row. The **pay_id** variable is assigned the value of the selected item fr **jcbPaymentID** combobox in the **PaymentForm** object. The method then creates a SQL query string to delete the row of data f **payment** table where the **payment_id** column matches the **pay_id** variabl The **getConnection()** method is called to establish a connection to the d and a **PreparedStatement** object is created to execute the query with the variable as a parameter to avoid SQL injection attacks. If the deletion is successful, the **refresh_controls()** method is called to the table and comboboxes with the updated data. If an exception occurs during the deletion process, the method displays message dialog box to the user.

```
1     public static void delete_handler(PaymentForm frm){
2         int dialogButton = JOptionPane.YES_NO_OPTION;
3         int pay_id =
4 Integer.parseInt(String.valueOf(frm.getJCBPaymentID().getSelectedIt
5
6         String message = String.format("Are you sure you want to de
7 the row Payment ID: %d)", pay_id);
8         int answer = JOptionPane.showConfirmDialog(frm, message, "D
9 ROW OF DATA", dialogButton);
10
11        if(answer == JOptionPane.YES_OPTION){
12            String query = """
13                DELETE FROM payment WHERE payment_id = ?""";
14            try(Connection conn = getConnection();
```

```
                PreparedStatement ps = conn.prepareStatement(query)
                // Use PreparedStatement to avoid SQL injection att
                ps.setInt(1, pay_id);
                ps.executeUpdate();

                // Refresh table and comboboxes
                refresh_controls(frm);

            } catch (SQLException ex){
                JOptionPane.showMessageDialog(frm, ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
            }
        }
    }
```

Step 2

In **PaymentForm.java**, double click on DELETE button to generate it listener:

```
    private void
jbDeleteActionPerformed(java.awt.event.ActionEvent
evt) {
        Payment_Utils.delete_handler(this);
    }
```

Step 3

Run the project. Choose **payment_id** using **jcbPaymentID** combobox. The on DELETE button. The corresponding row of data had been delet database.

PLOTTING CHART

PLOTTING CHART

Step 1

Create a new **JFrame** and save it as **Charts_Payment.java**.

Step 2

In **Charts_Rental.java**, add six **JPanels** and set their corresponding **Va Name** as **jPanel1**, **jPanel2**, **jPanel3**, **jPanel4**, **jPanel5**, and **jPanel6**. add getter method for each object as follows:

```
    //Getter method for jPanel1
    public JPanel getJPanel1(){
        return this.jPanel1;
    }
```

```
    //Getter method for jPanel2
    public JPanel getJPanel2(){
        return this.jPanel2;
    }

    //Getter method for jPanel3
    public JPanel getJPanel3(){
        return this.jPanel3;
    }

    //Getter method for jPanel4
    public JPanel getJPanel4(){
        return this.jPanel4;
    }

    //Getter method for jPanel5
    public JPanel getJPanel5(){
        return this.jPanel5;
    }

    //Getter method for jPanel6
    public JPanel getJPanel6(){
        return this.jPanel6;
    }
```

Step 3

In **Rental_Utils** class, define six new methods. These are methods that ar to draw different types of charts in the Charts_Payment form.

1. **draw_pie_chart_by_payment_year()**: This method draws a pie ch payment distribution by year of payment date. It crea **DefaultPieDataset** by calling **create_pie_dataset()** method wi SQL query that retrieves payment data grouped by payment year it calls **draw_piechart_with_dataset()** method to draw the pie with the given dataset and chart title.
2. **draw_pie_chart_by_payment_month()**: This method draws a pie of payment distribution by month of payment date. It crea **DefaultPieDataset** by calling **create_pie_dataset()** method wi SQL query that retrieves payment data grouped by payment m Then it calls **draw_piechart_with_dataset()** method to draw th chart with the given dataset and chart title.
3. **draw_pie_chart_by_payment_week()**: This method draws a pie of top 10 payment distribution by week of payment date. It cre **DefaultPieDataset** by calling **create_pie_dataset()** method wi SQL query that retrieves payment data grouped by payment week it calls **draw_piechart_with_dataset()** method to draw the pie with the given dataset and chart title.
4. **draw_pie_chart_by_payment_day()**: This method draws a pie ch payment distribution by day of payment date. It creat

DefaultPieDataset by calling **create_pie_dataset()** method wi SQL query that retrieves payment data grouped by payment day. T calls **draw_piechart_with_dataset()** method to draw the pie char the given dataset and chart title.

5. **draw_pie_chart_by_payment_quarter()**: This method draws chart of payment distribution by quarter of payment date. It cre **DefaultPieDataset** by calling **create_pie_dataset()** method wi SQL query that retrieves payment data grouped by payment qu Then it calls **draw_piechart_with_dataset()** method to draw tl chart with the given dataset and chart title.
6. **draw_bar_chart_by_release_year()**: This method draws a bar ch payment distribution by film release year. It creat **DefaultCategoryDataset** by calling **create_bar_dataset()** metho the SQL query that retrieves payment data grouped by film release Then it calls **draw_barchart_with_dataset()** method to draw th chart with the given dataset, chart title, X-axis label, and Y-axis lab

```
    private static void
draw_pie_chart_by_payment_year(Charts_Payment frm, JPanel jp){
        jp.setPreferredSize(new Dimension(jp.getWidth(),
jp.getHeight()));
        DefaultPieDataset dataset =
create_pie_dataset(Query_Payment.get_sql_payment_year(),
"payment", "year");

        //Draws piechart payment distribution by the year of
payment date
        draw_piechart_with_dataset(frm, jp, dataset, "THE PAYMENT
DISTRIBUTION BY YEAR OF PAYMENT DATE");
    }

    private static void
draw_pie_chart_by_payment_month(Charts_Payment frm, JPanel jp){
        jp.setPreferredSize(new Dimension(jp.getWidth(),
jp.getHeight()));
        DefaultPieDataset dataset =
create_pie_dataset(Query_Payment.get_sql_payment_month(),
"payment", "month");

        //Draws piechart payment distribution by the month of
payment date
        draw_piechart_with_dataset(frm, jp, dataset, "THE PAYMENT
DISTRIBUTION BY MONTH OF PAYMENT DATE");
    }

    private static void
draw_pie_chart_by_payment_week(Charts_Payment frm, JPanel jp){
        jp.setPreferredSize(new Dimension(jp.getWidth(),
jp.getHeight()));
        DefaultPieDataset dataset =
create_pie_dataset(Query_Payment.get_sql_payment_week(),
```

```
"payment", "week");

        //Draws piechart payment distribution by the week of
payment date
        draw_piechart_with_dataset(frm, jp, dataset, "THE TOP 10
PAYMENT DISTRIBUTION BY WEEK OF PAYMENT DATE");
    }

    private static void
draw_pie_chart_by_payment_day(Charts_Payment frm, JPanel jp){
        jp.setPreferredSize(new Dimension(jp.getWidth(),
jp.getHeight()));
        DefaultPieDataset dataset =
create_pie_dataset(Query_Payment.get_sql_payment_day(), "payment",
"day");

        //Draws piechart payment distribution by the day of
payment date
        draw_piechart_with_dataset(frm, jp, dataset, "THE PAYMENT
DISTRIBUTION BY DAY OF PAYMENT DATE");
    }

    private static void
draw_pie_chart_by_payment_quarter(Charts_Payment frm, JPanel jp){
        jp.setPreferredSize(new Dimension(jp.getWidth(),
jp.getHeight()));
        DefaultPieDataset dataset =
create_pie_dataset(Query_Payment.get_sql_payment_quarter(),
"payment", "quarter");

        //Draws piechart payment distribution by the quarter of
payment date
        draw_piechart_with_dataset(frm, jp, dataset, "THE PAYMENT
DISTRIBUTION BY QUARTER OF PAYMENT DATE");
    }

    private static void
draw_bar_chart_by_release_year(Charts_Payment frm, JPanel jp){
        jp.setPreferredSize(new Dimension(jp.getWidth(),
jp.getHeight()));

        DefaultCategoryDataset dataset =
create_bar_dataset(Query_Payment.get_sql_payment_film_year_dist(),
"payment", "film_year");

        //Draws barchart payment distribution by film release year
        draw_barchart_with_dataset(frm, jp, dataset, "THE PAYMENT
DISTRIBUTION BY FILM RELEASE YEAR", "YEAR", "NUMBER");
    }
```

Step	In **Payment_Utils** class, define a new method named **jbchart_handler()**.

4

```
    public static void
jbchart_handler(Charts_Payment frm){
        //Draws piechart payment
distribution by the year of payment
date

draw_pie_chart_by_payment_year(frm,
frm.getJPanel1());

        //Draws piechart payment
distribution by the month of payment
date

draw_pie_chart_by_payment_month(frm,
frm.getJPanel2());

        //Draws piechart payment
distribution by the week of payment
date

draw_pie_chart_by_payment_week(frm,
frm.getJPanel3());

        //Draws piechart payment
distribution by the day of payment
date

draw_pie_chart_by_payment_day(frm,
frm.getJPanel4());

        //Draws piechart payment
distribution by the quarter of payment
date

draw_pie_chart_by_payment_quarter(frm,
frm.getJPanel5());

        //Draws barchart payment
distribution by film release year

draw_bar_chart_by_release_year(frm,
frm.getJPanel6());
    }
```

Step 5

In **PaymentForm**, double click on **jbChart** button to define its event lister

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

```
        Charts_Payment frm1 = new
Charts_Payment();
        frm1.setLocationRelativeTo(null);
        frm1.setTitle("SIX DISTRIBUTIONS IN
PAYMENT TABLE");
        frm1.setVisible(true);
        Payment_Utils.jbchart_handler(frm1);
    }
```

Step 6

Run the project. Click on CHART button on the form. You will see the six displayed on the panels as shown in Figure 16.13.

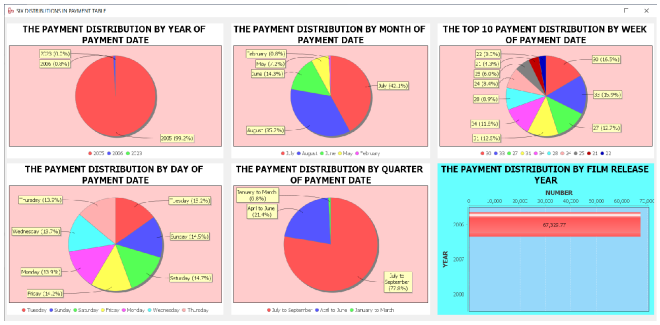


Figure 16.13 The payment distribution by year of payment date, the pay distribution by month of payment date, the top 10 payment distribution by of payment date, the payment distribution by day of payment date, the pay distribution by quarter of payment date, and the payment distribution by release year

Step 7

Create a new **JFrame** and save it as **Charts_Payment2.java**.

Step 8

In **Charts_Payment2.java**, add six **JPanels** and set their correspo **Variable Name** as **jPanel1**, **jPanel2**, **jPanel3**, **jPanel4**, **jPanel5**, and **jPa** Then, add getter method for each object as follows:

```
    //Getter method for jPanel1
    public JPanel getJPanel1(){
        return this.jPanel1;
    }

    //Getter method for jPanel2
    public JPanel getJPanel2(){
        return this.jPanel2;
    }

    //Getter method for jPanel3
    public JPanel getJPanel3(){
        return this.jPanel3;
```

```
    }

    //Getter method for jPanel4
    public JPanel getJPanel4(){
        return this.jPanel4;
    }

    //Getter method for jPanel5
    public JPanel getJPanel5(){
        return this.jPanel5;
    }

    //Getter method for jPanel6
    public JPanel getJPanel6(){
        return this.jPanel6;
    }
```

Step 9

In **Payment_Utils** class, define six new methods. These are six private methods that draw different types of charts based on data obtained fror queries. Here's a brief summary of each method and what it does:

1. **draw_pie_chart_payment_film_duration()**: draws a pie chart sh the distribution of payments by the duration of the rented film.
2. **draw_pie_chart_payment_film_rating()**: draws a pie chart sh the distribution of payments by the rating of the rented film.
3. **draw_bar_chart_payment_by_staff_name()**: draws a bar showing the top 10 staff members based on the number of rented and the corresponding number of payments made.
4. **draw_pie_chart_payment_by_language()**: draws a pie chart sh the distribution of payments by the language of the rented film.
5. **draw_bar_chart_payment_by_film_title()**: draws a bar chart sh the top 10 rented films based on the number of rentals, an corresponding number of payments made.
6. **draw_bar_chart_payment_by_customer_active()**: draws a bar showing the distribution of payments by whether the customer is or not.

```
    private static void
draw_pie_chart_payment_film_duration(Charts_Payment2 frm, JPanel jp
        jp.setPreferredSize(new Dimension(jp.getWidth(),
jp.getHeight()));
        DefaultPieDataset dataset =
create_pie_dataset(Query_Payment.get_sql_payment_film_duration_dist
"payment", "film_duration");

        //Draws piechart payment distribution by the film duration
        draw_piechart_with_dataset(frm, jp, dataset, "THE PAYMENT
DISTRIBUTION BY FILM DURATION");
```

```
    }

    private static void
draw_pie_chart_payment_film_rating(Charts_Payment2 frm, JPanel jp){
        jp.setPreferredSize(new Dimension(jp.getWidth(),
jp.getHeight()));
        DefaultPieDataset dataset =
create_pie_dataset(Query_Payment.get_sql_payment_film_rating_dist()
"payment", "film_rating");

        //Draws piechart payment distribution by the film rating
        draw_piechart_with_dataset(frm, jp, dataset, "THE PAYMENT
DISTRIBUTION BY FILM RATING");
    }

    private static void
draw_bar_chart_payment_by_staff_name(Charts_Payment2 frm, JPanel jp
        jp.setPreferredSize(new Dimension(jp.getWidth(),
jp.getHeight()));

        DefaultCategoryDataset dataset =
create_bar_dataset(Query_Payment.get_sql_payment_staff_name_dist(),
"payment", "staff_name");

        //Draws barchart payment distribution by staff name
        draw_barchart_with_dataset(frm, jp, dataset, "THE TOP 10 PA
DISTRIBUTION BY STAFF NAME", "STAFF NAME", "NUMBER OF RENTED FILMS"
    }

    private static void
draw_pie_chart_payment_by_language(Charts_Payment2 frm, JPanel jp){
        jp.setPreferredSize(new Dimension(jp.getWidth(),
jp.getHeight()));
        DefaultPieDataset dataset =
create_pie_dataset(Query_Payment.get_sql_payment_film_language_dist
"payment", "film_language");

        //Draws piechart payment distribution by the film language
        draw_piechart_with_dataset(frm, jp, dataset, "THE PAYMENT
DISTRIBUTION BY FILM LANGUAGE");
    }

    private static void
draw_bar_chart_payment_by_film_title(Charts_Payment2 frm, JPanel jp
        jp.setPreferredSize(new Dimension(jp.getWidth(),
jp.getHeight()));

        DefaultCategoryDataset dataset =
create_bar_dataset(Query_Payment.get_sql_payment_film_title_dist(),
"payment", "film_title");

        //Draws barchart payment distribution by film title

```

```
66          draw_barchart_with_dataset(frm, jp, dataset, "THE TOP 10 PA
67  DISTRIBUTION BY FILM TITLE", "FILM TITLE", "NUMBER OF RENTED FILMS"
68      }
69
70      private static void
71  draw_bar_chart_payment_by_customer_active(Charts_Payment2 frm, JPan
72  jp){
73          jp.setPreferredSize(new Dimension(jp.getWidth(),
74  jp.getHeight()));

            DefaultCategoryDataset dataset =
    create_bar_dataset(Query_Payment.get_sql_payment_customer_active_di
    "payment", "customer_active");

            //Draws barchart payment distribution by customer active/no
    active
            draw_barchart_with_dataset(frm, jp, dataset, "THE PAYMENT
    DISTRIBUTION BY CUSTOMER ACTIVE", "CUSTOMER ACTIVE/NOT ACTIVE", "NU
    OF RENTED FILMS");
        }
```

Step 10

In **Payment_Utils** class, define a new method named **jbchart_handler2()**

```
1       public static void
2   jbchart_handler2(Charts_Payment2 frm){
3           //Draws piechart payment distribution
4   by the film duration
5
6   draw_pie_chart_payment_film_duration(frm,
7   frm.getJPanel1());
8
9           //Draws piechart payment distribution
10  by the film rating
11
12  draw_pie_chart_payment_film_rating(frm,
13  frm.getJPanel2());
14
15          //Draws barchart payment distribution
16  by staff name
17
18  draw_bar_chart_payment_by_staff_name(frm,
19  frm.getJPanel3());
20

            //Draws piechart payment distribution
    by the film language

    draw_pie_chart_payment_by_language(frm,
    frm.getJPanel4());

            //Draws barchart payment distribution
    by film title
```

```
draw_bar_chart_payment_by_film_title(frm,
frm.getJPanel5());

        //Draws barchart payment distribution
by customer active/not active

draw_bar_chart_payment_by_customer_active(frm,
frm.getJPanel6());

    }
```

Step 11

In **PaymentForm**, double click on **jbChart** button to modify its event liste adding code in line 8 – 12:

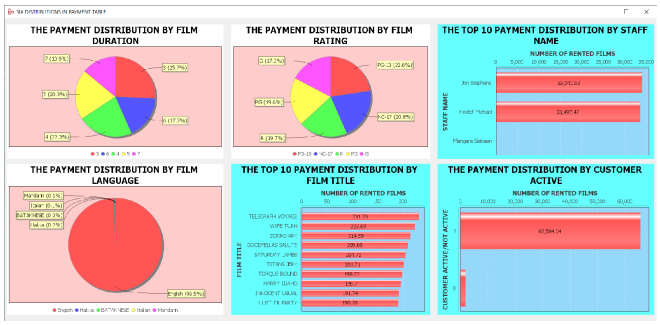


Figure 16.14 The payment distribution by film duration, the paymen distribution by film rating, the top 10 payment distribution by staff name payment distribution by film language, the top 10 payment distribution by title, and the payment distribution by customer active

```
1   private void
2   jbChartActionPerformed(java.awt.event.ActionEvent
3   evt) {
4           Charts_Payment frm1 = new
5   Charts_Payment();
6           frm1.setLocationRelativeTo(null);
7           frm1.setTitle("SIX DISTRIBUTIONS IN
8   PAYMENT TABLE");
9           frm1.setVisible(true);
10          Payment_Utils.jbchart_handler(frm1);
11
12          Charts_Payment2 frm2 = new
13  Charts_Payment2();
14          frm2.setLocationRelativeTo(null);
            frm2.setTitle("SIX DISTRIBUTIONS IN
    PAYMENT TABLE");
            frm2.setVisible(true);
            Payment_Utils.jbchart_handler2(frm2);


    }
```

Step 12	Run the project. Click on CHART button on the form. You will see the six displayed on the panels as shown in Figure 16.14.
Step 13	Create a new **JFrame** and save it as **Charts_Payment3.java**.

Step 14	In **Charts_Payment3.java**, add six **JPanels** and set their correspo **Variable Name** as **jPanel1**, **jPanel2**, **jPanel3**, **jPanel4**, **jPanel5**, and **jPa** Then, add getter method for each object as follows:
	
	`1 //Getter method for jPanel1`
	`2 public JPanel getJPanel1(){`
	`3 return this.jPanel1;`
	`4 }`
	`5`
	`6 //Getter method for jPanel2`
	`7 public JPanel getJPanel2(){`
	`8 return this.jPanel2;`
	`9 }`
	`10`
	`11 //Getter method for jPanel3`
	`12 public JPanel getJPanel3(){`
	`13 return this.jPanel3;`
	`14 }`
	`15`
	`16 //Getter method for jPanel4`
	`17 public JPanel getJPanel4(){`
	`18 return this.jPanel4;`
	`19 }`
	`20`
	`21 //Getter method for jPanel5`
	`22 public JPanel getJPanel5(){`
	`23 return this.jPanel5;`
	`24 }`
	`25`
	`26 //Getter method for jPanel6`
	`27 public JPanel getJPanel6(){`
	`28 return this.jPanel6;`
	`29 }`

Step 15	In **Payment_Utils** class, define six new methods. These method responsible for drawing bar charts of payment distributions based on v criteria such as film category, actor name, customer name, customer customer country, and customer district.

Each method first sets the preferred size of the provided **JPanel**, cre
dataset using a corresponding SQL query that is defined in the **Query_Pa**
class, and then calls a method named **draw_barchart_with_dataset()** to
the bar chart with the provided dataset, chart title, x-axis label, and y-axis

The **draw_barchart_with_dataset()** method is likely defined elsewhere
program and takes care of creating a **JFreeChart** object with the pr
dataset and labels, and then adding it to the provided **JFrame**.

```
1
2
3
4
5
6
7
8
9
10
11
12
```

```
13      private static void
14  draw_bar_chart_payment_by_film_category(Charts_Payment3 frm, JPanel jp){
15          jp.setPreferredSize(new Dimension(jp.getWidth(), jp.getHeight()))
16
17          DefaultCategoryDataset dataset =
18  create_bar_dataset(Query_Payment.get_sql_payment_category_dist(),
19  "payment", "film_category");
20
21          //Draws barchart payment distribution by film category
22          draw_barchart_with_dataset(frm, jp, dataset, "THE TOP 10 PAYMENT
23  DISTRIBUTION BY FILM CATEGORY", "FILM CATEGORY", "NUMBER OF RENTED
24  FILMS");
25      }
26
27      private static void
28  draw_bar_chart_payment_by_actor_name(Charts_Payment3 frm, JPanel jp){
29          jp.setPreferredSize(new Dimension(jp.getWidth(), jp.getHeight()))
30
31          DefaultCategoryDataset dataset =
32  create_bar_dataset(Query_Payment.get_sql_payment_actor_dist(), "payment",
33  "actor_name");
34
35          //Draws barchart payment distribution by actor name
36          draw_barchart_with_dataset(frm, jp, dataset, "THE TOP 10 PAYMENT
37  DISTRIBUTION BY ACTOR NAME", "ACTOR NAME", "NUMBER OF RENTED FILMS");
38      }
39
40      private static void
41  draw_bar_chart_payment_by_customer_name(Charts_Payment3 frm, JPanel jp){
42          jp.setPreferredSize(new Dimension(jp.getWidth(), jp.getHeight()))
43
44          DefaultCategoryDataset dataset =
45  create_bar_dataset(Query_Payment.get_sql_payment_customer_name_dist(),
46  "payment", "customer_name");
47
48          //Draws barchart payment distribution by customer name
49          draw_barchart_with_dataset(frm, jp, dataset, "THE TOP 10 PAYMENT
50  DISTRIBUTION BY CUSTOMER NAME", "CUSTOMER NAME", "NUMBER OF RENTED
51  FILMS");
52      }
53
54      private static void
55  draw_bar_chart_payment_by_customer_city(Charts_Payment3 frm, JPanel jp){
56          jp.setPreferredSize(new Dimension(jp.getWidth(), jp.getHeight()))
57
58          DefaultCategoryDataset dataset =
59  create_bar_dataset(Query_Payment.get_sql_payment_customer_city_dist(),
60  "payment", "customer_city");
61
62          //Draws barchart payment distribution by customer city
63          draw_barchart_with_dataset(frm, jp, dataset, "THE TOP 10 PAYMENT
64  DISTRIBUTION BY CUSTOMER CITY", "CUSTOMER CITY", "NUMBER OF RENTED
65  FILMS");
```

```
66      }
67
68      private static void
69  draw_bar_chart_payment_by_customer_country(Charts_Payment3 frm, JPanel jp
70  {
71          jp.setPreferredSize(new Dimension(jp.getWidth(), jp.getHeight()))
72
73          DefaultCategoryDataset dataset =
74  create_bar_dataset(Query_Payment.get_sql_payment_customer_country_dist(),
75  "payment", "customer_country");
76
77          //Draws barchart payment distribution by customer country
78          draw_barchart_with_dataset(frm, jp, dataset, "THE TOP 10 PAYMENT
    DISTRIBUTION BY CUSTOMER COUNTRY", "CUSTOMER COUNTRY", "NUMBER OF RENTED
    FILMS");
      }

      private static void
    draw_bar_chart_payment_by_customer_district(Charts_Payment3 frm, JPanel
    jp){
          jp.setPreferredSize(new Dimension(jp.getWidth(), jp.getHeight()))

          DefaultCategoryDataset dataset =
    create_bar_dataset(Query_Payment.get_sql_payment_customer_district_dist()
    "payment", "customer_district");
          //Draws barchart payment distribution by customer district
          draw_barchart_with_dataset(frm, jp, dataset, "THE TOP 10 PAYMENT
    DISTRIBUTION BY CUSTOMER DISTRICT", "CUSTOMER DISTRICT", "NUMBER OF RENTE
    FILMS");
      }
```

Step 16

Step 17

Step 18

This is the full version of **Payment_Utils.java**:

```
package sakila;
import java.awt.Dimension;
import java.util.logging.Level;
import java.util.logging.Logger;
import java.sql.*;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashMap;
import java.util.Objects;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.table.DefaultTableModel;
import org.jfree.data.category.DefaultCategoryDataset;
import org.jfree.data.general.DefaultPieDataset;

public class Payment_Utils extends Utility{
    public static final int FIRST_INDEX = 0;
    public static final int INVALID_INDEX = -1;
```

```
    private static int currentIndex = FIRST_INDEX;
    private static final String SQL_ID = Query_Payment.get_sql_id();

    //Creates payment table
    public static void create_payment_table() {
        try (Connection conn = getConnection()) {
            Statement stmt = conn.createStatement();
            stmt.addBatch(Query_Payment.get_sql_payment());
            stmt.executeBatch();

            String message = String.format("Successfully creates payment tabl
            JOptionPane.showMessageDialog(null, message,
            "INFORMATION",JOptionPane.INFORMATION_MESSAGE);

        } catch (SQLException ex) {
            JOptionPane.showMessageDialog(null, ex.getMessage(),
                "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    //Populates payment table with some rows of data
    public static void populate_payment_table(){
        try(Connection conn = getConnection()){
            String sql = """
                INSERT INTO payment(payment_id, customer_id, staff_id, rental
amount, payment_date, last_update)
                VALUES(?, ?, ?, ?, ?, ?, ?)""";

            //Creates a new Payment class with default constructor
            PreparedStatement ps1 = conn.prepareStatement(sql);
            Payment obj1 = new Payment();
            ps1.setInt(1,obj1.getPaymentID());
            ps1.setInt(2,obj1.getCustomerID());
            ps1.setInt(3,obj1.getStaffID());
            ps1.setInt(4,obj1.getRentalID());
            ps1.setDouble(5,obj1.getAmount());
            ps1.setDate(6,obj1.getPaymentDate());
            ps1.setTimestamp(7,obj1.getLastUpdate());

            // Creates a new Payment class with nine-params constructor
            PreparedStatement ps2 = conn.prepareStatement(sql);
            SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd");
            String str_pay_date = "2023-05-03";
            java.util.Date pay_date = dateFormat.parse(str_pay_date);
            java.sql.Date pay_sql_date = new java.sql.Date(pay_date.getTime()

            Payment obj2 = new Payment(2, 2, 2, 2, 10.75, pay_sql_date, new
Timestamp(System.currentTimeMillis()));
            ps2.setInt(1,obj2.getPaymentID());
            ps2.setInt(2,obj2.getCustomerID());
            ps2.setInt(3,obj2.getStaffID());
            ps2.setInt(4,obj2.getRentalID());
```

```
            ps2.setDouble(5,obj2.getAmount());
            ps2.setDate(6,obj2.getPaymentDate());
            ps2.setTimestamp(7,obj2.getLastUpdate());

            ps1.executeUpdate();
            ps2.executeUpdate();

        }catch(SQLException ex){
            JOptionPane.showMessageDialog(null, ex.getMessage(),
                "ERROR",JOptionPane.ERROR_MESSAGE);
        } catch (ParseException ex) {
            Logger.getLogger(Rental_Utils.class.getName()).log(Level.SEVERE,
ex);
        }
    }

    //Reads the content of payment table
    public static void read_payment_table(){
        try(Connection conn = getConnection()){
            Statement stmt = conn.createStatement();
            ResultSet rs = stmt.executeQuery("SELECT * FROM payment");

            while(rs.next()){
                int pay_id = rs.getInt("payment_id");
                int rental_id = rs.getInt("rental_id");
                Date pay_date = rs.getDate("payment_date");
                int cust_id = rs.getInt("customer_id");
                int staff_id = rs.getInt("staff_id");
                double amount = rs.getDouble("amount");
                Timestamp lu = rs.getTimestamp("last_update");

                //Creates a Payment object using seven-params constructor
                Payment obj = new Payment(pay_id, cust_id, staff_id, rental_i
amount, pay_date, lu);
                System.out.println(obj);
            }
            rs.close();
            stmt.close();

        }catch(SQLException ex){
            JOptionPane.showMessageDialog(null, ex.getMessage(),
                "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    private static ArrayList<Payment> get_payment_list(PaymentForm frm, Strin
String item){
        ArrayList<Payment> list = new ArrayList<>();

        try(Connection conn = getConnection();
            PreparedStatement ps = conn.prepareStatement(sql)){
            if (item.equalsIgnoreCase("none")==false) {
                 ps.setString(1,item);
```

```
            }
            ResultSet rs = ps.executeQuery();
            Payment obj;

            while(rs.next()){
                //Using thirty-one-params constructor
                obj = new Payment(rs.getInt("payment_id"),
                   rs.getDouble("amount"),
                   rs.getDate("payment_date"),
                   rs.getInt("rental_id"),
                   rs.getDate("rental_date"),
                   rs.getInt("inventory_id"),
                   rs.getInt("customer_id"),
                   rs.getDate("return_date"),
                   rs.getInt("staff_id"),
                   rs.getTimestamp("last_update"),
                   rs.getInt("film_id"),
                   rs.getString("title"),
                   rs.getString("description"),
                   rs.getInt("release_year"),
                   rs.getInt("rental_duration"),
                   rs.getDouble("rental_rate"),
                   rs.getInt("length"),
                   rs.getDouble("replacement_cost"),
                   rs.getString("rating"),
                   rs.getString("special_features"),
                   rs.getString("language_name"),
                   rs.getString("category_name"),
                   rs.getString("actor_name"),
                   rs.getString("customer_name"),
                   rs.getString("customer_email"),
                   rs.getString("staff_name"),
                   rs.getString("staff_email"),
                   rs.getString("store_city"),
                   rs.getString("store_country"),
                   rs.getString("store_district"),
                   rs.getString("store_postal_code"));

                list.add(obj);
            }
        }catch (SQLException ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
                "ERROR",JOptionPane.ERROR_MESSAGE);
        }
        return list;
    }

    private static void show_table_payment(PaymentForm frm, ArrayList<Payment
throws SQLException{
        DefaultTableModel model = new DefaultTableModel(0,0);

        String header[] = {"Payment ID", "Amount", "Payment Date",
            "Rental ID", "Rental Date", "Inventory ID", "Customer ID", "Retu
            "Staff ID", "Film ID", "Title", "Description", "Release Year",
```

```
            "Rental Duration", "Rental Rate", "Length", "Replacement Cost",
            "Rating", "Special Features", "Language", "Category", "Actor Name
            "Customer Name", "Customer Email", "Staff Name", "Staff Email",
            "Store City", "Store Country", "Store District", "Store Postal Co
            "Last Update"};

        model.setColumnIdentifiers(set_column_header(frm.getJTPayment(), head
        frm.getJTPayment().setModel(model);

        Object[] row = new Object[31];

        for(int i=0; i<list.size(); i++){
            row[0] = list.get(i).getPaymentID();
            row[1] = list.get(i).getAmount();
            row[2] = list.get(i).getPaymentDate();
            row[3] = list.get(i).getRentalID();
            row[4] = list.get(i).getRentalRate();
            row[5] = list.get(i).getInventoryID();
            row[6] = list.get(i).getCustomerID();
            row[7] = list.get(i).getReturnDate();
            row[8] = list.get(i).getStaffID();
            row[9] = list.get(i).getFilmID();
            row[10] = list.get(i).getTitle();
            row[11] = list.get(i).getDescription();
            row[12] = list.get(i).getReleaseYear();
            row[13] = list.get(i).getRentalDuration();
            row[14] = list.get(i).getRentalRate();
            row[15] = list.get(i).getLength();
            row[16] = list.get(i).getReplacementCost();
            row[17] = list.get(i).getRating();
            row[18] = list.get(i).getSpecialFeatures();
            row[19] = list.get(i).getLanguageName();
            row[20] = list.get(i).getCategory();
            row[21] = list.get(i).getActorName();
            row[22] = list.get(i).getCustomerName();
            row[23] = list.get(i).getCustomerEmail();
            row[24] = list.get(i).getStaffName();
            row[25] = list.get(i).getStaffEmail();
            row[26] = list.get(i).getStoreCity();
            row[27] = list.get(i).getStoreCountry();
            row[28] = list.get(i).getStoreDistrict();
            row[29] = list.get(i).getStorePostalCode();
            row[30] = list.get(i).getLastUpdate();

            model.addRow(row);
        }
    }

    public static void refresh_controls(PaymentForm frm){
        frm.setLocationRelativeTo(null);
        frm.setTitle("PAYMENT FORM");

        //Shows the content of payment table and populates combobox
        try{
```

```
            //Makes alternating color for table rows
            table_renderer(frm.getJTPayment());

            //Populates table
            ArrayList<Payment> list = get_payment_list(frm,
Query_Payment.get_sql_payment_joint() + " ORDER BY payment_id", "none");
            show_table_payment(frm, list);

            //Populates jcbRentalID
            String sql_rent_id = "SELECT rental_id FROM rental ORDER BY renta
            populate_combobox(sql_rent_id, frm.getJCBRentalID(), frm);

            //Populates jcbPaymentID
            String sql_pay_id = "SELECT payment_id FROM payment ORDER BY payr
            populate_combobox(sql_pay_id, frm.getJCBPaymentID(), frm);

            //Populates jcbCustomerID
            String sql_cust_id = "SELECT customer_id FROM customer ORDER BY
customer_id";
            populate_combobox(sql_cust_id, frm.getJCBCustomerID(), frm);

            //Populates jcbStaffID
            String sql_stf_id = "SELECT staff_id FROM staff ORDER BY staff_ic
            populate_combobox(sql_stf_id, frm.getJCBStaffID(), frm);

        }catch (SQLException ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
            "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    private static void clear_staff_controls(PaymentForm frm){
        frm.getJTFStaffID().setText("");
        frm.getJTFStaffName().setText("");
        frm.getJTFStaffEmail().setText("");
    }

   //Displays staff data result row by row
    private static <T> void display_staff_data(PaymentForm frm, String sql, T
        try(Connection conn = getConnection()){
            PreparedStatement ps = conn.prepareStatement(sql);
            ps.setObject(1,item);
            ResultSet rs = ps.executeQuery();

            if (!rs.next()) {
                // no row found, clear the form fields
                clear_staff_controls(frm);
                return;
            }

            do{
                frm.getJTFStaffID().setText(String.valueOf(rs.getInt("staff_i
```

```
                frm.getJTFStaffName().setText(rs.getString("first_name") + "
rs.getString("last_name"));
                frm.getJTFStaffEmail().setText(rs.getString("email"));

                // Determines item selected from jcbStaffID
                find_combo_value_selected(frm.getJCBStaffID(), rs.getInt("sta

            }while(rs.next());

            rs.close();
            ps.close();
        }catch(SQLException ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    public static void jcbStaff_handler(PaymentForm frm) {
        Object item = frm.getJCBStaffID().getSelectedItem();
        display_staff_data(frm, Query_Staff.get_sql_id(), item);
    }

    private static void clear_customer_controls(PaymentForm frm){
        frm.getJTFCustomerID().setText("");
        frm.getJTFCustomerName().setText("");
        frm.getJTFCustEmail().setText("");
    }

   //Displays customer data result row by row
    private static <T> void display_customer_data(PaymentForm frm, String sql
{
        try(Connection conn = getConnection()){
            PreparedStatement ps = conn.prepareStatement(sql);
            ps.setObject(1,item);
            ResultSet rs = ps.executeQuery();

            if (!rs.next()) {
                // no row found, clear the form fields
                clear_customer_controls(frm);
                return;
            }

            do{

frm.getJTFCustomerID().setText(String.valueOf(rs.getInt("customer_id")));
                frm.getJTFCustomerName().setText(rs.getString("first_name") +
rs.getString("last_name"));
                frm.getJTFCustEmail().setText(rs.getString("email"));

                // Determines item selected from jcbCustomerID
                find_combo_value_selected(frm.getJCBCustomerID(),
rs.getInt("customer_id"));
```

```
            }while(rs.next());

            rs.close();
            ps.close();
        }catch(SQLException ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    public static void jcbCustomer_handler(PaymentForm frm) {
        Object item = frm.getJCBCustomerID().getSelectedItem();
        display_customer_data(frm, Query_Customer.get_sql_id(), item);
    }

    private static void clear_rental_controls(PaymentForm frm){
        frm.getJTFRentalID().setText("");
        frm.getJTFRentalDate().setText("");
        frm.getJTFReturnDate().setText("");
        frm.getJTFInventoryID().setText("");
        frm.getJTFFilmID().setText("");
        frm.getJTFStoreID().setText("");
        frm.getJTFAddressID().setText("");
        frm.getJTFAddress().setText("");
        frm.getJTFTitle().setText("");
        frm.getJTFDescription().setText("");
        frm.getJTFReleaseYear().setText("");
        frm.getJTFRentalDuration().setText("");
        frm.getJTFRentalRate().setText("");
        frm.getJTFLength().setText("");
        frm.getJTFReplacementCost().setText("");
        frm.getJTFSpecialFeatures().setText("");
        frm.getJTFCategory().setText("");
        frm.getJTFLanguageName().setText("");
        frm.getJTFActorName().setText("");
        frm.getJTFDistrict().setText("");
        frm.getJTFPhone().setText("");
        frm.getJTFPostalCode().setText("");
        frm.getJTFCity().setText("");
        frm.getJTFCountry().setText("");
    }
   //Displays rental data result row by row
    private static <T> void display_rental_data(PaymentForm frm, String sql,
        try(Connection conn = getConnection()){
            PreparedStatement ps = conn.prepareStatement(sql);
            ps.setObject(1,item);
            ResultSet rs = ps.executeQuery();

            if (!rs.next()) {
                // no row found, clear the form fields
                clear_rental_controls(frm);
                return;
            }
```

```
            do{
                frm.getJTFRentalID().setText(String.valueOf(rs.getInt("rental
frm.getJTFRentalDate().setText(String.valueOf(rs.getDate("rental_date")));

frm.getJTFReturnDate().setText(String.valueOf(rs.getDate("return_date")));

frm.getJTFInventoryID().setText(String.valueOf(rs.getInt("inventory_id")));
                frm.getJTFFilmID().setText(String.valueOf(rs.getInt("film_id"
                frm.getJTFStoreID().setText(String.valueOf(rs.getInt("store_i
frm.getJTFAddressID().setText(String.valueOf(rs.getInt("address_id")));
                frm.getJTFAddress().setText(rs.getString("store_address"));
                frm.getJTFDistrict().setText(rs.getString("store_district"));
                frm.getJTFCity().setText(rs.getString("store_city"));
                frm.getJTFCountry().setText(rs.getString("store_country"));
                frm.getJTFPhone().setText(rs.getString("store_phone"));
                frm.getJTFPostalCode().setText(rs.getString("store_postal_cod
                frm.getJTFTitle().setText(rs.getString("title"));
                frm.getJTFDescription().setText(rs.getString("description"));
frm.getJTFReleaseYear().setText(String.valueOf(rs.getInt("release_year")));

frm.getJTFRentalDuration().setText(String.valueOf(rs.getInt("rental_duration"
frm.getJTFRentalRate().setText(String.valueOf(rs.getDouble("rental_rate")));
                frm.getJTFLength().setText(String.valueOf(rs.getInt("length")
frm.getJTFReplacementCost().setText(String.valueOf(rs.getDouble("replacement_
                frm.getJTFSpecialFeatures().setText(rs.getString("special_fea
                frm.getJTFRating().setText(rs.getString("rating"));
                frm.getJTFLanguageName().setText(rs.getString("language_name"
                frm.getJTFActorName().setText(rs.getString("actor_name"));
                frm.getJTFCategory().setText(rs.getString("category_name"));

                // Determines item selected from jcbRentalID
                find_combo_value_selected(frm.getJCBRentalID(),
rs.getInt("rental_id"));

            }while(rs.next());

            rs.close();
            ps.close();
        }catch(SQLException ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    public static void jcbRental_handler(PaymentForm frm) {
        Object item = frm.getJCBRentalID().getSelectedItem();
        display_rental_data(frm, Query_Rental.get_sql_rental_joint() + " WHER
rental_id = ?", item);
```

```
    }

   //Displays payment data result row by row
    private static <T> void display_payment_data(PaymentForm frm, String sql,
        try(Connection conn = getConnection()){
            PreparedStatement ps = conn.prepareStatement(sql);
            ps.setObject(1,item);
            ResultSet rs = ps.executeQuery();

            if (!rs.next()) {
                // no row found, clear the form fields
                frm.getJTFPaymentID().setText("");
                frm.getJTFAmount().setText("");
                frm.getJTFLastUpdate().setText("");
                return;
            }

            do{

frm.getJTFPaymentID().setText(String.valueOf(rs.getInt("payment_id")));
                frm.getJTFAmount().setText(String.valueOf(rs.getInt("amount")
                frm.getJTFLastUpdate().setText(rs.getString("last_update"));

                java.util.Date paydate = new
Date(rs.getTimestamp("payment_date").getTime());
                frm.getJCPaymentDate().setDate(paydate);

                // Determines item selected from jcbRentalID
                find_combo_value_selected(frm.getJCBRentalID(),
rs.getInt("rental_id"));

                // Determines item selected from jcbPaymentID
                find_combo_value_selected(frm.getJCBPaymentID(),
rs.getInt("payment_id"));

                // Determines item selected from jcbStaffID
                find_combo_value_selected(frm.getJCBStaffID(), rs.getInt("sta

                // Determines item selected from jcbCustomerID
                find_combo_value_selected(frm.getJCBCustomerID(),
rs.getInt("customer_id"));

            }while(rs.next());

            rs.close();
            ps.close();
        }catch(SQLException ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    public static void jcbPayment_handler(PaymentForm frm) {
```

```
        Object item = frm.getJCBPaymentID().getSelectedItem();
        display_payment_data(frm, Query_Payment.get_sql_id(), item);
    }

    public static void show_first_row(PaymentForm frm){
        String item = String.valueOf(frm.getJCBPaymentID().getItemAt(FIRST_IN
        display_payment_data(frm, SQL_ID, item);
        currentIndex = FIRST_INDEX;
    }

    public static void show_last_row(PaymentForm frm){
        int endIndex = frm.getJCBPaymentID().getItemCount() - 1;
        String item = String.valueOf(frm.getJCBPaymentID().getItemAt(endIndex
        display_payment_data(frm, SQL_ID, item);
        currentIndex = endIndex;
    }

    public static void show_prev_row(PaymentForm frm){
        currentIndex--;
        if(currentIndex < FIRST_INDEX){
            currentIndex = FIRST_INDEX;
            return;
        }
        String item = String.valueOf(frm.getJCBPaymentID().getItemAt(currentI
        display_payment_data(frm, SQL_ID, item);
    }

    public static void show_next_row(PaymentForm frm){
        int endIndex = frm.getJCBPaymentID().getItemCount() - 1;
        currentIndex++;
        if(currentIndex > endIndex){
            currentIndex = endIndex;
            return;
        }
        String item = String.valueOf(frm.getJCBPaymentID().getItemAt(currentI
        display_payment_data(frm, SQL_ID, item);
    }

    public static void mouse_pressed_handler(PaymentForm frm) {
        Objects.requireNonNull(frm, "frm must not be null");

        int selectedIndex = frm.getJTPayment().getSelectedRow();
        if (selectedIndex == -1) {
            JOptionPane.showMessageDialog(frm, "Please select a row to view i
data.",
                "No row selected", JOptionPane.INFORMATION_MESSAGE);
            return;
        }

        try (Connection conn = getConnection()) {
            String id =
String.valueOf(frm.getJTPayment().getModel().getValueAt(selectedIndex, 0));
```

```
            // Displays payment data
            display_payment_data(frm, SQL_ID, id);

        } catch (SQLException ex) {
            Logger.getLogger(PaymentForm.class.getName()).log(Level.SEVERE, '
displaying payment data", ex);
            String message = "Error displaying payment data: " + ex.getMessag
            String stackTrace = Arrays.toString(ex.getStackTrace());
            JOptionPane.showMessageDialog(frm, message + "\n\n" + stackTrace,
JOptionPane.ERROR_MESSAGE);
        }
    }

    //Updates row of data in payment tabel by payment_id
    public static void update_row_by_payment_id(PaymentForm frm, int pay_id,
cust_id,
            int staff_id, int rental_id, double amount, Date pay_date) throws
SQLException{
        Connection conn = getConnection();
        ResultSet rs = null;
        String query_id = "SELECT payment_id FROM payment WHERE payment_id =
        String update_query = """
            UPDATE payment SET customer_id = ?, staff_id = ?, rental_id = ?,
            amount = ?, payment_date = ?  WHERE payment_id = ?""";
        try(PreparedStatement idPs = conn.prepareStatement(query_id,
                    ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_READ_ONL
        PreparedStatement updatePS = conn.prepareStatement(update_query,
                ResultSet.TYPE_SCROLL_SENSITIVE, ResultSet.CONCUR_UPDATABLE))
        {
            idPs.setInt(1,pay_id);
            if(!idPs.execute()){
                String message = "Can't find payment_id " + pay_id;

                JOptionPane.showMessageDialog(frm, message,
                    "ERROR",JOptionPane.ERROR_MESSAGE);
            } else{
                rs = idPs.getResultSet();
                rs.next();

                //Creates a Payment object using seven-params constructor
                Payment obj = new Payment(pay_id, cust_id, staff_id, rental_i
amount, pay_date, new Timestamp(System.currentTimeMillis()));
                updatePS.setInt(1, obj.getCustomerID());
                updatePS.setInt(2, obj.getStaffID());
                updatePS.setInt(3, obj.getRentalID());
                updatePS.setDouble(4, obj.getAmount());
                updatePS.setDate(5, obj.getPaymentDate());
                updatePS.setInt(6, obj.getPaymentID());

                updatePS.executeUpdate();
                rs.close();
                updatePS.close();
                idPs.close();
```

```
                conn.close();
            }
        }catch(SQLException ex){
            Logger.getLogger(PaymentForm.class.getName()).log(Level.SEVERE, '
updating payment data", ex);
            String message = "Error updating payment data: " + ex.getMessage(
            String stackTrace = Arrays.toString(ex.getStackTrace());
            JOptionPane.showMessageDialog(null, message + "\n\n" + stackTrace
"ERROR", JOptionPane.ERROR_MESSAGE);
        }catch(java.lang.NumberFormatException ex){
            Logger.getLogger(PaymentForm.class.getName()).log(Level.SEVERE, '
Input", ex);
            String message = "Invalid Input: " + ex.getMessage();
            String stackTrace = Arrays.toString(ex.getStackTrace());
            JOptionPane.showMessageDialog(null, message + "\n\n" + stackTrace
"ERROR", JOptionPane.ERROR_MESSAGE);
        }
    }

    private static HashMap<String, String> read_inputs(PaymentForm frm) {
        HashMap<String, String> input_data = new HashMap<>();
        String cust_id = String.valueOf(frm.getJCBCustomerID().getSelectedIte
        String rental_id = String.valueOf(frm.getJCBRentalID().getSelectedIte
        String staff_id = String.valueOf(frm.getJCBStaffID().getSelectedItem(
        String pay_id = String.valueOf(frm.getJCBPaymentID().getSelectedItem(
        String amount = frm.getJTFAmount().getText();

        //Gets the selected date from the getJCPaymentDate()
        java.util.Date pay_date = frm.getJCPaymentDate().getDate();
        //Create a SimpleDateFormat object with the desired date format
        SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");
        //Formats the date as a String using the SimpleDateFormat object
        String str_pay_date = sdf.format(pay_date);

        // Validate user input
        int rental_id_int = 0;
        try {
            rental_id_int = Integer.parseInt(rental_id);
            if (rental_id_int <= 0) {
                throw new IllegalArgumentException("Rental ID cannot be negat
zero");
            }
        } catch (NumberFormatException ex) {
            JOptionPane.showMessageDialog(frm, "Invalid Rental ID: " + rental
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        } catch (IllegalArgumentException ex) {
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        }

        int staff_id_int = 0;
```

```
        try {
            staff_id_int = Integer.parseInt(staff_id);
            if (staff_id_int <= 0) {
                throw new IllegalArgumentException("Staff ID cannot be negat
zero");
            }
        } catch (NumberFormatException ex) {
            JOptionPane.showMessageDialog(frm, "Invalid Staff ID: " + staff_
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        } catch (IllegalArgumentException ex) {
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        }

        int cust_id_int = 0;
        try {
            cust_id_int = Integer.parseInt(cust_id);
            if (cust_id_int <= 0) {
                throw new IllegalArgumentException("Customer ID cannot be neg
zero");
            }
        } catch (NumberFormatException ex) {
            JOptionPane.showMessageDialog(frm, "Invalid Customer ID: " + cust
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        } catch (IllegalArgumentException ex) {
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        }

        int pay_id_int = 0;
        try {
            pay_id_int = Integer.parseInt(pay_id);
            if (pay_id_int <= 0) {
                throw new IllegalArgumentException("Payment ID cannot be nega
zero");
            }
        } catch (NumberFormatException ex) {
            JOptionPane.showMessageDialog(frm, "Invalid Payment ID: " + pay_
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        } catch (IllegalArgumentException ex) {
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
            "Error", JOptionPane.ERROR_MESSAGE);
            throw ex;
        }

        input_data.put("cust_id", cust_id);
        input_data.put("pay_id", pay_id);
        input_data.put("staff_id", staff_id);
```

```
        input_data.put("rental_id", rental_id);
        input_data.put("amount", amount);
        input_data.put("str_pay_date", str_pay_date);

        return input_data;
    }

    private static void edit_actual(PaymentForm frm) throws ParseException{
        try{
            HashMap<String, String> input_data = read_inputs(frm);
            int cust_id = Integer.parseInt(input_data.get("cust_id"));
            int pay_id = Integer.parseInt(input_data.get("pay_id"));
            int staff_id = Integer.parseInt(input_data.get("staff_id"));
            int rental_id = Integer.parseInt(input_data.get("rental_id"));
            String str_pay_date = input_data.get("str_pay_date");
            double amount = Double.parseDouble(input_data.get("amount"));

            SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd");
            java.util.Date pay_date = dateFormat.parse(str_pay_date);
            java.sql.Date pay_sql_date = new java.sql.Date(pay_date.getTime()

            update_row_by_payment_id(frm, pay_id, cust_id, staff_id, rental_i
amount, pay_sql_date);

            //Refreshes all objects on form
            refresh_controls(frm);

        }catch(SQLException ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    private static void enable_controls(boolean state, PaymentForm frm){
        frm.getJBFirst().setEnabled(state);
        frm.getJBPrev().setEnabled(state);
        frm.getJBNext().setEnabled(state);
        frm.getJBLast().setEnabled(state);
        frm.getJBInsert().setEnabled(state);
        frm.getJBDelete().setEnabled(state);
        frm.getJTFAddressID().setEnabled(state);
        frm.getJTFStoreID().setEnabled(state);
        frm.getJTFAddress().setEnabled(state);
        frm.getJTFDistrict().setEnabled(state);
        frm.getJTFPhone().setEnabled(state);
        frm.getJTFPostalCode().setEnabled(state);
        frm.getJTFCity().setEnabled(state);
        frm.getJTFCountry().setEnabled(state);
        frm.getJTFStaffID().setEnabled(state);
        frm.getJTFStaffName().setEnabled(state);
        frm.getJTFStaffEmail().setEnabled(state);
        frm.getJTFLastUpdate().setEnabled(state);
        frm.getJTFFilmID().setEnabled(state);
```

```
        frm.getJTFCustEmail().setEnabled(state);
        frm.getJTFCustomerName().setEnabled(state);
        frm.getJTFTitle().setEnabled(state);
        frm.getJTFDescription().setEnabled(state);
        frm.getJTFReleaseYear().setEnabled(state);
        frm.getJTFRentalDuration().setEnabled(state);
        frm.getJTFRentalRate().setEnabled(state);
        frm.getJTFLength().setEnabled(state);
        frm.getJTFReplacementCost().setEnabled(state);
        frm.getJTFSpecialFeatures().setEnabled(state);
        frm.getJTFCategory().setEnabled(state);
        frm.getJTFLanguageName().setEnabled(state);
        frm.getJTFActorName().setEnabled(state);
        frm.getJTFInventoryID().setEnabled(state);
        frm.getJTFReturnDate().setEnabled(state);
        frm.getJTFRentalDate().setEnabled(state);
    }

    public static void edit_handler(PaymentForm frm){
        if(frm.getJBEdit().getText().equals("EDIT")){
            frm.getJBEdit().setText("CONFIRM");

            // Disables controls
            enable_controls(false, frm);
        }

        else {
            try {
                frm.getJBEdit().setText("EDIT");

                // Actual editing
                edit_actual(frm);

                //Enables controls
                enable_controls(true, frm);
            } catch (ParseException ex) {
                Logger.getLogger(Payment_Utils.class.getName()).log(Level.SE\
null, ex);
            }
        }
    }

    //Inserts new row into payment table
    private static void insert_row(PaymentForm frm) throws SQLException,
ParseException{
        HashMap<String, String> input_data = read_inputs(frm);
        int cust_id = Integer.parseInt(input_data.get("cust_id"));
        int staff_id = Integer.parseInt(input_data.get("staff_id"));
        int rental_id = Integer.parseInt(input_data.get("rental_id"));
        String str_pay_date = input_data.get("str_pay_date");
        double amount = Double.parseDouble(input_data.get("amount"));

        SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd");
```

```
        java.util.Date pay_date = dateFormat.parse(str_pay_date);
        java.sql.Date pay_sql_date = new java.sql.Date(pay_date.getTime());

        // SQL insert statement
        String sql = """
            INSERT INTO payment(customer_id, staff_id, rental_id,
                amount, payment_date) VALUES(?, ?, ?, ?, ?)""";

        try(Connection conn = getConnection();
         PreparedStatement pstmt = conn.prepareStatement(sql)){

            //Creates a Payment object six-params constructor
            Payment obj = new Payment(cust_id, staff_id, rental_id, amount,
pay_sql_date, new Timestamp(System.currentTimeMillis()));
            pstmt.setInt(1,obj.getCustomerID());
            pstmt.setInt(2,obj.getStaffID());
            pstmt.setInt(3,obj.getRentalID());
            pstmt.setDouble(4,obj.getAmount());
            pstmt.setDate(5, obj.getPaymentDate());

            //Executes the sql insert statement
            pstmt.executeUpdate();
        } catch (SQLException ex) {
            Logger.getLogger(PaymentForm.class.getName()).log(Level.SEVERE, "
error", ex);
            JOptionPane.showMessageDialog(frm, "Error: Database error\n" +
ex.getMessage());
        }
    }

    private static void insert_actual(PaymentForm frm) throws ParseException{
        try{
            insert_row(frm);

            //Refreshes table and comboboxes
            refresh_controls(frm);

        }catch(SQLException ex){
            JOptionPane.showMessageDialog(frm, ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
        }
    }

    public static void insert_handler(PaymentForm frm){
        if(frm.getJBInsert().getText().equals("INSERT")){
            frm.getJBInsert().setText("CONFIRM");

            //Disables jbEdit
            frm.getJBEdit().setEnabled(false);
```

```
            // Disables controls
            enable_controls(false, frm);
            frm.getJCBPaymentID().setEnabled(false);

            // Clears controls
            frm.getJTFAmount().setText("");
            frm.getJTFLastUpdate().setText("");

            // Enables
            frm.getJBInsert().setEnabled(true);
        }

        else {
            frm.getJBInsert().setText("INSERT");

            try {
                // Actual insertion
                insert_actual(frm);
            } catch (ParseException ex) {

Logger.getLogger(Payment_Utils.class.getName()).log(Level.SEVERE, null,
ex);
            }

            //Enables jbEdit
            frm.getJBEdit().setEnabled(true);

            //Enables controls
            enable_controls(true, frm);
            frm.getJCBPaymentID().setEnabled(true);
        }
    }

    public static void delete_handler(PaymentForm frm){
        int dialogButton = JOptionPane.YES_NO_OPTION;
        int pay_id =
Integer.parseInt(String.valueOf(frm.getJCBPaymentID().getSelectedItem()));

        String message = String.format("Are you sure you want to delete
the row Payment ID: %d)", pay_id);
        int answer = JOptionPane.showConfirmDialog(frm, message, "DELETING
ROW OF DATA", dialogButton);

        if(answer == JOptionPane.YES_OPTION){
            String query = """
                DELETE FROM payment WHERE payment_id = ?""";
            try(Connection conn = getConnection();
                PreparedStatement ps = conn.prepareStatement(query)){
                // Use PreparedStatement to avoid SQL injection attacks
                ps.setInt(1, pay_id);
                ps.executeUpdate();
```

```
                // Refresh table and comboboxes
                refresh_controls(frm);

            } catch (SQLException ex){
                JOptionPane.showMessageDialog(frm, ex.getMessage(),
                    "ERROR",JOptionPane.ERROR_MESSAGE);
            }
        }
    }

    private static void draw_pie_chart_by_payment_year(Charts_Payment frm,
JPanel jp){
        jp.setPreferredSize(new Dimension(jp.getWidth(), jp.getHeight()));
        DefaultPieDataset dataset =
create_pie_dataset(Query_Payment.get_sql_payment_year(), "payment",
"year");

        //Draws piechart payment distribution by the year of payment date
        draw_piechart_with_dataset(frm, jp, dataset, "THE PAYMENT
DISTRIBUTION BY YEAR OF PAYMENT DATE");
    }

    private static void draw_pie_chart_by_payment_month(Charts_Payment
frm, JPanel jp){
        jp.setPreferredSize(new Dimension(jp.getWidth(), jp.getHeight()));
        DefaultPieDataset dataset =
create_pie_dataset(Query_Payment.get_sql_payment_month(), "payment",
"month");

        //Draws piechart payment distribution by the month of payment date
        draw_piechart_with_dataset(frm, jp, dataset, "THE PAYMENT
DISTRIBUTION BY MONTH OF PAYMENT DATE");
    }

    private static void draw_pie_chart_by_payment_week(Charts_Payment frm,
JPanel jp){
        jp.setPreferredSize(new Dimension(jp.getWidth(), jp.getHeight()));
        DefaultPieDataset dataset =
create_pie_dataset(Query_Payment.get_sql_payment_week(), "payment",
"week");

        //Draws piechart payment distribution by the week of payment date
        draw_piechart_with_dataset(frm, jp, dataset, "THE TOP 10 PAYMENT
DISTRIBUTION BY WEEK OF PAYMENT DATE");
    }

    private static void draw_pie_chart_by_payment_day(Charts_Payment frm,
JPanel jp){
        jp.setPreferredSize(new Dimension(jp.getWidth(), jp.getHeight()));
        DefaultPieDataset dataset =
create_pie_dataset(Query_Payment.get_sql_payment_day(), "payment", "day");

        //Draws piechart payment distribution by the day of payment date
```

```
        draw_piechart_with_dataset(frm, jp, dataset, "THE PAYMENT
DISTRIBUTION BY DAY OF PAYMENT DATE");
    }

    private static void draw_pie_chart_by_payment_quarter(Charts_Payment
frm, JPanel jp){
        jp.setPreferredSize(new Dimension(jp.getWidth(), jp.getHeight()));
        DefaultPieDataset dataset =
create_pie_dataset(Query_Payment.get_sql_payment_quarter(), "payment",
"quarter");

        //Draws piechart payment distribution by the quarter of payment
date
        draw_piechart_with_dataset(frm, jp, dataset, "THE PAYMENT
DISTRIBUTION BY QUARTER OF PAYMENT DATE");
    }

    private static void draw_bar_chart_by_release_year(Charts_Payment frm,
JPanel jp){
        jp.setPreferredSize(new Dimension(jp.getWidth(), jp.getHeight()));

        DefaultCategoryDataset dataset =
create_bar_dataset(Query_Payment.get_sql_payment_film_year_dist(),
"payment", "film_year");

        //Draws barchart payment distribution by film release year
        draw_barchart_with_dataset(frm, jp, dataset, "THE PAYMENT
DISTRIBUTION BY FILM RELEASE YEAR", "YEAR", "NUMBER");
    }

    public static void jbchart_handler(Charts_Payment frm){
        //Draws piechart payment distribution by the year of payment date
        draw_pie_chart_by_payment_year(frm, frm.getJPanel1());

        //Draws piechart payment distribution by the month of payment date
        draw_pie_chart_by_payment_month(frm, frm.getJPanel2());

        //Draws piechart payment distribution by the week of payment date
        draw_pie_chart_by_payment_week(frm, frm.getJPanel3());

        //Draws piechart payment distribution by the day of payment date
        draw_pie_chart_by_payment_day(frm, frm.getJPanel4());

        //Draws piechart payment distribution by the quarter of payment
date
        draw_pie_chart_by_payment_quarter(frm, frm.getJPanel5());

        //Draws barchart payment distribution by film release year
        draw_bar_chart_by_release_year(frm, frm.getJPanel6());
    }

    private static void
draw_pie_chart_payment_film_duration(Charts_Payment2 frm, JPanel jp){
```

```
        jp.setPreferredSize(new Dimension(jp.getWidth(), jp.getHeight()));
        DefaultPieDataset dataset =
create_pie_dataset(Query_Payment.get_sql_payment_film_duration_dist(),
"payment", "film_duration");

        //Draws piechart payment distribution by the film duration
        draw_piechart_with_dataset(frm, jp, dataset, "THE PAYMENT
DISTRIBUTION BY FILM DURATION");
    }

    private static void draw_pie_chart_payment_film_rating(Charts_Payment2
frm, JPanel jp){
        jp.setPreferredSize(new Dimension(jp.getWidth(), jp.getHeight()));
        DefaultPieDataset dataset =
create_pie_dataset(Query_Payment.get_sql_payment_film_rating_dist(),
"payment", "film_rating");

        //Draws piechart payment distribution by the film rating
        draw_piechart_with_dataset(frm, jp, dataset, "THE PAYMENT
DISTRIBUTION BY FILM RATING");
    }

    private static void
draw_bar_chart_payment_by_staff_name(Charts_Payment2 frm, JPanel jp){
        jp.setPreferredSize(new Dimension(jp.getWidth(), jp.getHeight()));

        DefaultCategoryDataset dataset =
create_bar_dataset(Query_Payment.get_sql_payment_staff_name_dist(),
"payment", "staff_name");

        //Draws barchart payment distribution by staff name
        draw_barchart_with_dataset(frm, jp, dataset, "THE TOP 10 PAYMENT
DISTRIBUTION BY STAFF NAME", "STAFF NAME", "NUMBER OF RENTED FILMS");
    }

    private static void draw_pie_chart_payment_by_language(Charts_Payment2
frm, JPanel jp){
        jp.setPreferredSize(new Dimension(jp.getWidth(), jp.getHeight()));
        DefaultPieDataset dataset =
create_pie_dataset(Query_Payment.get_sql_payment_film_language_dist(),
"payment", "film_language");

        //Draws piechart payment distribution by the film language
        draw_piechart_with_dataset(frm, jp, dataset, "THE PAYMENT
DISTRIBUTION BY FILM LANGUAGE");
    }

    private static void
draw_bar_chart_payment_by_film_title(Charts_Payment2 frm, JPanel jp){
        jp.setPreferredSize(new Dimension(jp.getWidth(), jp.getHeight()));

        DefaultCategoryDataset dataset =
create_bar_dataset(Query_Payment.get_sql_payment_film_title_dist(),
```

```
"payment", "film_title");

        //Draws barchart payment distribution by film title
        draw_barchart_with_dataset(frm, jp, dataset, "THE TOP 10 PAYMENT
DISTRIBUTION BY FILM TITLE", "FILM TITLE", "NUMBER OF RENTED FILMS");
    }

    private static void
draw_bar_chart_payment_by_customer_active(Charts_Payment2 frm, JPanel jp){
        jp.setPreferredSize(new Dimension(jp.getWidth(), jp.getHeight()));

        DefaultCategoryDataset dataset =
create_bar_dataset(Query_Payment.get_sql_payment_customer_active_dist(),
"payment", "customer_active");

        //Draws barchart payment distribution by customer active/not
active
        draw_barchart_with_dataset(frm, jp, dataset, "THE PAYMENT
DISTRIBUTION BY CUSTOMER ACTIVE", "CUSTOMER ACTIVE/NOT ACTIVE", "NUMBER OF
RENTED FILMS");
    }

    public static void jbchart_handler2(Charts_Payment2 frm){
        //Draws piechart payment distribution by the film duration
        draw_pie_chart_payment_film_duration(frm, frm.getJPanel1());

        //Draws piechart payment distribution by the film rating
        draw_pie_chart_payment_film_rating(frm, frm.getJPanel2());

        //Draws barchart payment distribution by staff name
        draw_bar_chart_payment_by_staff_name(frm, frm.getJPanel3());

        //Draws piechart payment distribution by the film language
        draw_pie_chart_payment_by_language(frm, frm.getJPanel4());

        //Draws barchart payment distribution by film title
        draw_bar_chart_payment_by_film_title(frm, frm.getJPanel5());

        //Draws barchart payment distribution by customer active/not
active
        draw_bar_chart_payment_by_customer_active(frm, frm.getJPanel6());

    }

    private static void
draw_bar_chart_payment_by_film_category(Charts_Payment3 frm, JPanel jp){
        jp.setPreferredSize(new Dimension(jp.getWidth(), jp.getHeight()));

        DefaultCategoryDataset dataset =
create_bar_dataset(Query_Payment.get_sql_payment_category_dist(),
"payment", "film_category");

        //Draws barchart payment distribution by film category
```

```
        draw_barchart_with_dataset(frm, jp, dataset, "THE TOP 10 PAYMENT
DISTRIBUTION BY FILM CATEGORY", "FILM CATEGORY", "NUMBER OF RENTED
FILMS");
    }

    private static void
draw_bar_chart_payment_by_actor_name(Charts_Payment3 frm, JPanel jp){
        jp.setPreferredSize(new Dimension(jp.getWidth(), jp.getHeight()));

        DefaultCategoryDataset dataset =
create_bar_dataset(Query_Payment.get_sql_payment_actor_dist(), "payment",
"actor_name");

        //Draws barchart payment distribution by actor name
        draw_barchart_with_dataset(frm, jp, dataset, "THE TOP 10 PAYMENT
DISTRIBUTION BY ACTOR NAME", "ACTOR NAME", "NUMBER OF RENTED FILMS");
    }

    private static void
draw_bar_chart_payment_by_customer_name(Charts_Payment3 frm, JPanel jp){
        jp.setPreferredSize(new Dimension(jp.getWidth(), jp.getHeight()));

        DefaultCategoryDataset dataset =
create_bar_dataset(Query_Payment.get_sql_payment_customer_name_dist(),
"payment", "customer_name");

        //Draws barchart payment distribution by customer name
        draw_barchart_with_dataset(frm, jp, dataset, "THE TOP 10 PAYMENT
DISTRIBUTION BY CUSTOMER NAME", "CUSTOMER NAME", "NUMBER OF RENTED
FILMS");
    }

    private static void
draw_bar_chart_payment_by_customer_city(Charts_Payment3 frm, JPanel jp){
        jp.setPreferredSize(new Dimension(jp.getWidth(), jp.getHeight()));

        DefaultCategoryDataset dataset =
create_bar_dataset(Query_Payment.get_sql_payment_customer_city_dist(),
"payment", "customer_city");

        //Draws barchart payment distribution by customer city
        draw_barchart_with_dataset(frm, jp, dataset, "THE TOP 10 PAYMENT
DISTRIBUTION BY CUSTOMER CITY", "CUSTOMER CITY", "NUMBER OF RENTED
FILMS");
    }

    private static void
draw_bar_chart_payment_by_customer_country(Charts_Payment3 frm, JPanel jp)
{
        jp.setPreferredSize(new Dimension(jp.getWidth(), jp.getHeight()));

        DefaultCategoryDataset dataset =
create_bar_dataset(Query_Payment.get_sql_payment_customer_country_dist(),
```

```
"payment", "customer_country");

        //Draws barchart payment distribution by customer country
        draw_barchart_with_dataset(frm, jp, dataset, "THE TOP 10 PAYMENT
DISTRIBUTION BY CUSTOMER COUNTRY", "CUSTOMER COUNTRY", "NUMBER OF RENTED
FILMS");
    }

    private static void
draw_bar_chart_payment_by_customer_district(Charts_Payment3 frm, JPanel
jp){
        jp.setPreferredSize(new Dimension(jp.getWidth(), jp.getHeight()));

        DefaultCategoryDataset dataset =
create_bar_dataset(Query_Payment.get_sql_payment_customer_district_dist(),
"payment", "customer_district");

        //Draws barchart payment distribution by customer district
        draw_barchart_with_dataset(frm, jp, dataset, "THE TOP 10 PAYMENT
DISTRIBUTION BY CUSTOMER DISTRICT", "CUSTOMER DISTRICT", "NUMBER OF RENTED
FILMS");
    }

    public static void jbchart_handler3(Charts_Payment3 frm){
        //Draws piechart payment distribution by the film category
        draw_bar_chart_payment_by_film_category(frm, frm.getJPanel1());

        //Draws barchart payment distribution by actor name
        draw_bar_chart_payment_by_actor_name(frm, frm.getJPanel2());

        //Draws barchart payment distribution by customer name
        draw_bar_chart_payment_by_customer_name(frm, frm.getJPanel3());

        //Draws barchart payment distribution by customer city
        draw_bar_chart_payment_by_customer_city(frm, frm.getJPanel4());

        //Draws barchart payment distribution by customer country
        draw_bar_chart_payment_by_customer_country(frm, frm.getJPanel5());

        //Draws barchart payment distribution by customer district
        draw_bar_chart_payment_by_customer_district(frm,
frm.getJPanel6());
    }
}
```

This is the full version of **PaymentForm.java**:

```
package sakila;

import com.toedter.calendar.JCalendar;
import java.awt.Toolkit;
```

```java
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JComboBox;
import javax.swing.JMenuItem;
import javax.swing.JOptionPane;
import javax.swing.JPopupMenu;
import javax.swing.JTable;
import javax.swing.JTextField;

public class PaymentForm extends javax.swing.JFrame {

    public PaymentForm() {
        initComponents();
        Utility.setLookAndFeel(this);
        Payment_Utils.refresh_controls(this);

 this.setIconImage(Toolkit.getDefaultToolkit().getImage(getClass().getResour
;
        this.setDefaultCloseOperation(this.HIDE_ON_CLOSE);
    }

    //Getter method for jtfPaymentID
    public JTextField getJTFPaymentID(){
        return this.jtfPaymentID;
    }

    //Getter method for jcbPaymentID
    public JComboBox getJCBPaymentID(){
        return this.jcbPaymentID;
    }

    //Getter method for jtfAmount
    public JTextField getJTFAmount(){
        return this.jtfAmount;
    }

    //Getter method for jtfRentalID
    public JTextField getJTFRentalID(){
        return this.jtfRentalID;
    }

    //Getter method for jcbRentalID
    public JComboBox getJCBRentalID(){
        return this.jcbRentalID;
    }

    //Getter method for jcPaymentDate
    public JCalendar getJCPaymentDate(){
        return this.jcPaymentDate;
    }

    //Getter method for jtfLastUpdate
```

```
    public JTextField getJTFLastUpdate(){
        return this.jtfLastUpdate;
    }

    //Getter method for jtfInventoryID
    public JTextField getJTFInventoryID(){
        return this.jtfInventoryID;
    }

    //Getter method for jtfFilmID
    public JTextField getJTFFilmID(){
        return this.jtfFilmID;
    }

    //Getter method for jtfRentalDate
    public JTextField getJTFRentalDate(){
        return this.jtfRentalDate;
    }

    //Getter method for jtfReturnDate
    public JTextField getJTFReturnDate(){
        return this.jtfReturnDate;
    }

    //Getter method for jtfStoreID
    public JTextField getJTFStoreID(){
        return this.jtfStoreID;
    }

    //Getter method for jtfAddressID
    public JTextField getJTFAddressID(){
        return this.jtfAddressID;
    }

    //Getter method for jtfAddress
    public JTextField getJTFAddress(){
        return this.jtfAddress;
    }

    //Getter method for jtfDistrict
    public JTextField getJTFDistrict(){
        return this.jtfDistrict;
    }

    //Getter method for jtfPhone
    public JTextField getJTFPhone(){
        return this.jtfPhone;
    }

    //Getter method for jtfCity
    public JTextField getJTFCity(){
        return this.jtfCity;
    }
```

```
    //Getter method for jtfCountry
    public JTextField getJTFCountry(){
        return this.jtfCountry;
    }

    //Getter method for jtfPostalCode
    public JTextField getJTFPostalCode(){
        return this.jtfPostalCode;
    }

    //Getter method for jtfTitle
    public JTextField getJTFTitle(){
        return this.jtfTitle;
    }

    //Getter method for jtfDescription
    public JTextField getJTFDescription(){
        return this.jtfDescription;
    }

    //Getter method for jtfReleaseYear
    public JTextField getJTFReleaseYear(){
        return this.jtfReleaseYear;
    }

    //Getter method for jtfRentalDuration
    public JTextField getJTFRentalDuration(){
        return this.jtfRentalDuration;
    }

    //Getter method for jtfRentalRate
    public JTextField getJTFRentalRate(){
        return this.jtfRentalRate;
    }

    //Getter method for jtfLength
    public JTextField getJTFLength(){
        return this.jtfLength;
    }

    //Getter method for jtfReplacementCost
    public JTextField getJTFReplacementCost(){
        return this.jtfReplacementCost;
    }

    //Getter method for jtfSpecialFeatures
    public JTextField getJTFSpecialFeatures(){
        return this.jtfSpecialFeatures;
    }

    //Getter method for jtfRating
```

```
    public JTextField getJTFRating(){
        return this.jtfRating;
    }

    //Getter method for jtfLanguageID
    public JTextField getJTFLanguageID(){
        return this.jtfLanguageName;
    }

    //Getter method for jtfCustomerID
    public JTextField getJTFCustomerID(){
        return this.jtfCustomerID;
    }

    //Getter method for jcbCustomerID
    public JComboBox getJCBCustomerID(){
        return this.jcbCustomerID;
    }

    //Getter method for jtfCustFname
    public JTextField getJTFCustomerName(){
        return this.jtfCustomerName;
    }

    //Getter method for jtfCustEmail
    public JTextField getJTFCustEmail(){
        return this.jtfCustEmail;
    }

    //Getter method for jtfStaffID
    public JTextField getJTFStaffID(){
        return this.jtfStaffID;
    }

    //Getter method for jcbStaffID
    public JComboBox getJCBStaffID(){
        return this.jcbStaffID;
    }

    //Getter method for jtfStaffName
    public JTextField getJTFStaffName(){
        return this.jtfStaffName;
    }

    //Getter method for jtfStaffEmail
    public JTextField getJTFStaffEmail(){
        return this.jtfStaffEmail;
    }

    //Getter method for jtfCategory
    public JTextField getJTFCategory(){
        return this.jtfCategory;
    }
```

```
    //Getter method for jtfLanguageName
    public JTextField getJTFLanguageName(){
        return this.jtfLanguageName;
    }

    //Getter method for jtfActorName
    public JTextField getJTFActorName(){
        return this.jtfActorName;
    }

    //Getter method for jtPayment
    public JTable getJTPayment(){
        return this.jtPayment;
    }

    //Getter method for jbEdit
    public JButton getJBEdit(){
        return this.jbEdit;
    }

    //Getter method for jbInsert
    public JButton getJBInsert(){
        return this.jbInsert;
    }

    //Getter method for jbDelete
    public JButton getJBDelete(){
        return this.jbDelete;
    }

    //Getter method for jbChart
    public JButton getJBChart(){
        return this.jbChart;
    }

    //Getter method for jbFirst
    public JButton getJBFirst(){
        return this.jbFirst;
    }

    //Getter method for jbPrev
    public JButton getJBPrev(){
        return this.jbPrev;
    }

    //Getter method for jbNext
    public JButton getJBNext(){
        return this.jbNext;
    }

    //Getter method for jbLast
```

```
    public JButton getJBLast(){
        return this.jbLast;
    }

    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">
    private void initComponents() {
        //...
        pack();
    }// </editor-fold>

    private void jcbRentalIDActionPerformed(java.awt.event.ActionEvent evt) {
        Payment_Utils.jcbRental_handler(this);
    }

    private void jtPaymentMousePressed(java.awt.event.MouseEvent evt) {
        Payment_Utils.mouse_pressed_handler(this);
    }

    private void jbFirstActionPerformed(java.awt.event.ActionEvent evt) {
        Payment_Utils.show_first_row(this);
    }

    private void jbPrevActionPerformed(java.awt.event.ActionEvent evt) {
        Payment_Utils.show_prev_row(this);
    }

    private void jbNextActionPerformed(java.awt.event.ActionEvent evt) {
        Payment_Utils.show_next_row(this);
    }

    private void jbLastActionPerformed(java.awt.event.ActionEvent evt) {
        Payment_Utils.show_last_row(this);
    }

    private void jbChartActionPerformed(java.awt.event.ActionEvent evt) {
        Charts_Payment frm1 = new Charts_Payment();
        frm1.setLocationRelativeTo(null);
        frm1.setTitle("SIX DISTRIBUTIONS IN PAYMENT TABLE");
        frm1.setVisible(true);
        Payment_Utils.jbchart_handler(frm1);

        Charts_Payment2 frm2 = new Charts_Payment2();
        frm2.setLocationRelativeTo(null);
        frm2.setTitle("SIX DISTRIBUTIONS IN PAYMENT TABLE");
        frm2.setVisible(true);
        Payment_Utils.jbchart_handler2(frm2);

        Charts_Payment3 frm3 = new Charts_Payment3();
        frm3.setLocationRelativeTo(null);
        frm3.setTitle("SIX DISTRIBUTIONS IN PAYMENT TABLE");
        frm3.setVisible(true);
        Payment_Utils.jbchart_handler3(frm3);
```

```
    }

    private void jbEditActionPerformed(java.awt.event.ActionEvent evt) {
        Payment_Utils.edit_handler(this);
    }

    private void jbInsertActionPerformed(java.awt.event.ActionEvent evt) {
        Payment_Utils.insert_handler(this);
    }

    private void jbDeleteActionPerformed(java.awt.event.ActionEvent evt) {
        Payment_Utils.delete_handler(this);
    }

    private void jbCountryFormActionPerformed(java.awt.event.ActionEvent evt)
        CountryForm ct_form = new CountryForm();
        ct_form.setVisible(true);
    }

    private void jbAddressFormActionPerformed(java.awt.event.ActionEvent evt)
        AddressForm add_form = new AddressForm();
        add_form.setVisible(true);
    }

    private void jbStoreFormActionPerformed(java.awt.event.ActionEvent evt) {
        StoreForm str_form = new StoreForm();
        str_form.setVisible(true);
    }

    private void jbCityFormActionPerformed(java.awt.event.ActionEvent evt) {
        CityForm cty_form = new CityForm();
        cty_form.setVisible(true);
    }

    private void jcbCustomerIDActionPerformed(java.awt.event.ActionEvent evt)
        Payment_Utils.jcbCustomer_handler(this);
    }

    private void jcbStaffIDActionPerformed(java.awt.event.ActionEvent evt) {
        Payment_Utils.jcbStaff_handler(this);
    }

    private void jbCityForm1ActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
    }

    private void jbFilmFormActionPerformed(java.awt.event.ActionEvent evt) {
        FilmForm frm = new FilmForm();
        frm.setVisible(true);
    }

    private void jbCategoryFormActionPerformed(java.awt.event.ActionEvent evt
```

```
        CategoryForm frm = new CategoryForm();
        frm.setVisible(true);
    }

    private void jbActorFormActionPerformed(java.awt.event.ActionEvent evt) {
        ActorForm frm = new ActorForm();
        frm.setVisible(true);
    }

    private void jcbPaymentIDActionPerformed(java.awt.event.ActionEvent evt)
        Payment_Utils.jcbPayment_handler(this);
    }

    public static void main(String args[]) {
        try {
            for (javax.swing.UIManager.LookAndFeelInfo info :
javax.swing.UIManager.getInstalledLookAndFeels()) {
                if ("Nimbus".equals(info.getName())) {
                    javax.swing.UIManager.setLookAndFeel(info.getClassName())
                    break;
                }
            }
        } catch (ClassNotFoundException ex) {

java.util.logging.Logger.getLogger(PaymentForm.class.getName()).log(java.util
null, ex);
        } catch (InstantiationException ex) {

java.util.logging.Logger.getLogger(PaymentForm.class.getName()).log(java.util
null, ex);
        } catch (IllegalAccessException ex) {

java.util.logging.Logger.getLogger(PaymentForm.class.getName()).log(java.util
null, ex);
        } catch (javax.swing.UnsupportedLookAndFeelException ex) {

java.util.logging.Logger.getLogger(PaymentForm.class.getName()).log(java.util
null, ex);
        }

        /* Create and display the form */
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new PaymentForm().setVisible(true);
            }
        });
    }

    // Variables declaration - do not modify
    private javax.swing.JLabel jLabel10;
    private javax.swing.JLabel jLabel11;
    private javax.swing.JLabel jLabel12;
    private javax.swing.JLabel jLabel13;
```

```
    private javax.swing.JLabel jLabel14;
    private javax.swing.JLabel jLabel15;
    private javax.swing.JLabel jLabel16;
    private javax.swing.JLabel jLabel17;
    private javax.swing.JLabel jLabel18;
    private javax.swing.JLabel jLabel19;
    private javax.swing.JLabel jLabel20;
    private javax.swing.JLabel jLabel21;
    private javax.swing.JLabel jLabel22;
    private javax.swing.JLabel jLabel23;
    private javax.swing.JLabel jLabel24;
    private javax.swing.JLabel jLabel25;
    private javax.swing.JLabel jLabel26;
    private javax.swing.JLabel jLabel27;
    private javax.swing.JLabel jLabel28;
    private javax.swing.JLabel jLabel29;
    private javax.swing.JLabel jLabel3;
    private javax.swing.JLabel jLabel30;
    private javax.swing.JLabel jLabel31;
    private javax.swing.JLabel jLabel32;
    private javax.swing.JLabel jLabel33;
    private javax.swing.JLabel jLabel34;
    private javax.swing.JLabel jLabel35;
    private javax.swing.JLabel jLabel36;
    private javax.swing.JLabel jLabel37;
    private javax.swing.JLabel jLabel4;
    private javax.swing.JLabel jLabel5;
    private javax.swing.JLabel jLabel6;
    private javax.swing.JLabel jLabel7;
    private javax.swing.JLabel jLabel8;
    private javax.swing.JLabel jLabel9;
    private javax.swing.JScrollPane jScrollPane1;
    private javax.swing.JButton jbActorForm;
    private javax.swing.JButton jbAddressForm;
    private javax.swing.JButton jbCategoryForm;
    private javax.swing.JButton jbChart;
    private javax.swing.JButton jbCityForm;
    private javax.swing.JButton jbCityForm1;
    private javax.swing.JButton jbCountryForm;
    private javax.swing.JButton jbDelete;
    private javax.swing.JButton jbEdit;
    private javax.swing.JButton jbFilmForm;
    private javax.swing.JButton jbFirst;
    private javax.swing.JButton jbInsert;
    private javax.swing.JButton jbLast;
    private javax.swing.JButton jbNext;
    private javax.swing.JButton jbPrev;
    private javax.swing.JButton jbStoreForm;
    private com.toedter.calendar.JCalendar jcPaymentDate;
    private javax.swing.JComboBox<String> jcbCustomerID;
    private javax.swing.JComboBox<String> jcbPaymentID;
    private javax.swing.JComboBox<String> jcbRentalID;
    private javax.swing.JComboBox<String> jcbStaffID;
    private javax.swing.JTable jtPayment;
```

```
    private javax.swing.JTextField jtfActorName;
    private javax.swing.JTextField jtfAddress;
    private javax.swing.JTextField jtfAddressID;
    private javax.swing.JTextField jtfAmount;
    private javax.swing.JTextField jtfCategory;
    private javax.swing.JTextField jtfCity;
    private javax.swing.JTextField jtfCountry;
    private javax.swing.JTextField jtfCustEmail;
    private javax.swing.JTextField jtfCustomerID;
    private javax.swing.JTextField jtfCustomerName;
    private javax.swing.JTextField jtfDescription;
    private javax.swing.JTextField jtfDistrict;
    private javax.swing.JTextField jtfFilmID;
    private javax.swing.JTextField jtfInventoryID;
    private javax.swing.JTextField jtfLanguageName;
    private javax.swing.JTextField jtfLastUpdate;
    private javax.swing.JTextField jtfLength;
    private javax.swing.JTextField jtfPaymentID;
    private javax.swing.JTextField jtfPhone;
    private javax.swing.JTextField jtfPostalCode;
    private javax.swing.JTextField jtfRating;
    private javax.swing.JTextField jtfReleaseYear;
    private javax.swing.JTextField jtfRentalDate;
    private javax.swing.JTextField jtfRentalDuration;
    private javax.swing.JTextField jtfRentalID;
    private javax.swing.JTextField jtfRentalRate;
    private javax.swing.JTextField jtfReplacementCost;
    private javax.swing.JTextField jtfReturnDate;
    private javax.swing.JTextField jtfSpecialFeatures;
    private javax.swing.JTextField jtfStaffEmail;
    private javax.swing.JTextField jtfStaffID;
    private javax.swing.JTextField jtfStaffName;
    private javax.swing.JTextField jtfStoreID;
    private javax.swing.JTextField jtfTitle;
    // End of variables declaration
}
```

MAIN FORM
MAIN FORM

DESIGNING GUI
DESIGNING GUI

Step 1	In the project, create a new **JFrame Form** and name it as **MainForm.java**. In the Design tab, add fifteen **JButton** to the form and set their corresponding **text** properties as CUSTOMER FORM, ADDRESS FORM, CITY FORM, COUNTRY FORM, LANGUAGE FORM, FILM FORM, CATEGORY FORM, FILM ACTOR, FILM CATEGORY, ACTOR FORM, INVENTORY FORM, PAYMENT FORM, RENTAL FORM, STORE FORM, and STAFF FORM. Then, set their corresponding **Variable Name** as **jbCustomerForm**, **jbAddressForm**, **jbCityForm**, **jbCountryForm**, **jbLanguageForm**, **jbFilmForm**, **jbCategoryForm**, **jbFilmActor**, **jbFilmCategory**,

jbActorForm, **jbInventoryForm**, **jbPaymentForm**, **jbRentalForm**, **jbStoreForm**, and **jbStaffForm**.

Step 2 Define their corresponding event listeners on **MainForm** as follows:

```
    private void jbActorFormActionPerformed(java.awt.event.ActionEvent evt) {
        ActorForm act_form = new ActorForm();
        act_form.setVisible(true);
    }

    private void jbAddressActionPerformed(java.awt.event.ActionEvent evt) {
        AddressForm add_form = new AddressForm();
        add_form.setVisible(true);
    }

    private void jbCategoryActionPerformed(java.awt.event.ActionEvent evt) {
        CategoryForm cat_form = new CategoryForm();
        cat_form.setVisible(true);
    }

    private void jbCityActionPerformed(java.awt.event.ActionEvent evt) {
        CityForm ct_form = new CityForm();
        ct_form.setVisible(true);
    }

    private void jbCountryActionPerformed(java.awt.event.ActionEvent evt) {
        CountryForm cty_form = new CountryForm();
        cty_form.setVisible(true);
    }

    private void jbCustomerActionPerformed(java.awt.event.ActionEvent evt) {
        CustomerForm cust_form = new CustomerForm();
        cust_form.setVisible(true);
    }

    private void jbFilmActorActionPerformed(java.awt.event.ActionEvent evt) {
        FilmActorForm fa_form = new FilmActorForm();
```

```
48          fa_form.setVisible(true);
49      }
50
51      private void
52  jbFilmCatActionPerformed(java.awt.event.ActionEvent
53  evt) {
54          FilmCategoryForm fc_form = new
55  FilmCategoryForm();
56          fc_form.setVisible(true);
57      }
58
59      private void
60  jbFilmActionPerformed(java.awt.event.ActionEvent evt)
61  {
62          FilmForm fm_form = new FilmForm();
63          fm_form.setVisible(true);
64      }
65
66      private void
67  jbInventoryActionPerformed(java.awt.event.ActionEvent
68  evt) {
69          InventoryForm inv_form = new InventoryForm();
70          inv_form.setVisible(true);
71      }
72
73      private void
74  jbLanguageActionPerformed(java.awt.event.ActionEvent
    evt) {
          LanguageForm lang_form = new LanguageForm();
          lang_form.setVisible(true);
      }

      private void
    jbPaymentActionPerformed(java.awt.event.ActionEvent
    evt) {
          PaymentForm pay_form = new PaymentForm();
          pay_form.setVisible(true);
      }

      private void
    jbRentalActionPerformed(java.awt.event.ActionEvent
    evt) {
          RentalForm rent_form = new RentalForm();
          rent_form.setVisible(true);
      }

      private void
    jbStoreActionPerformed(java.awt.event.ActionEvent
    evt) {
          StoreForm st_form = new StoreForm();
          st_form.setVisible(true);
      }
```

```
    private void
jbStaffActionPerformed(java.awt.event.ActionEvent
evt) {
        StaffForm stf_form = new StaffForm();
        stf_form.setVisible(true);
    }
```

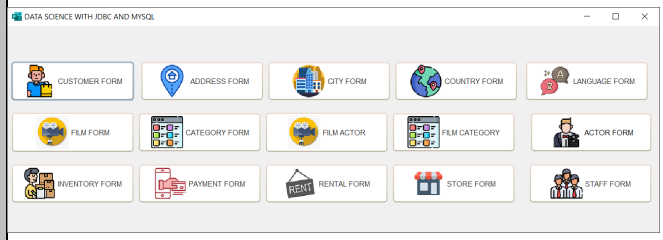


Figure 17.1 The main form

Step 3 In the driver class, **Sakila.java**, create a new object of **MainForm** class using its default constructor:

```
1  package sakila;
2
3  public class Sakila {
4      public static void main(String[]
5  args) {
6          MainForm frm = new
7  MainForm();
8          frm.setVisible(true);
       }
   }
```

Step 4 Run the project. You will see the main form to access the fifteen forms as shown in Figure 17.1.