Green University of Bangladesh

Department of Computer Science and Engineering (CSE)
Semester: (Spring, Year: 2024), B.Sc. in CSE (Day)

Employee Information System

Course Title: Database System Lab
Course Code: CSE 210
Section: 221 D8

Students Details

Name	ID
Sayed Hasan Emon	221002051

Submission Date: 10 June, 2024
Course Teacher's Name: Fatema Tuj Johora

[For teachers use only: Don't write anything inside this box]

Lab Project Status	
Marks:	Signature:
Comments:	Date:

Contents

Chapter 1

Introduction

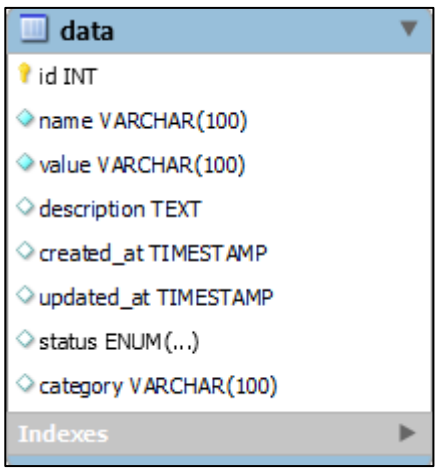
1.1 Overview

This project developed a secure Employee Information System (EIS) with a Java GUI. The system prioritizes data privacy through a login feature, allowing authorized users to perform CRUD (Create, Read, Update, Delete) operations on employee data and conduct targeted queries for display. This user-friendly interface streamlines data management, improves accuracy, and ensures confidentiality of sensitive employee information. The system can be further enhanced by integrating with other HR systems, implementing advanced search functions, and generating reports..

1.2 Motivation

Implementing an Employee Information System using Java and MySQL boosts efficiency, data accuracy, and decision-making. It ensures secure data management, scalability, and compliance with regulations. Ultimately, it enhances productivity, reduces administrative burdens, and demonstrates a commitment to employee satisfaction

1.3 Schema Diagram



```sql
CREATE TABLE data (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    value VARCHAR(100) NOT NULL,
    description TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
    status ENUM('active', 'inactive') DEFAULT 'active',
    category VARCHAR(100)
);
```

1.4 Problem Definition

1.4.1 Problem Statement

The current manual system for managing employee information poses several challenges within the organization. Employees' data is stored in disparate formats, including paper-based records and spreadsheets, leading to inefficiencies and inconsistencies in data management. Without a centralized system in place, tasks such as saving, editing, and deleting employee records become cumbersome and error-prone.

To address these challenges and improve operational efficiency, there is a pressing need to develop an automated Employee Information System with robust features for saving, editing, and deleting employee records. Such a system would streamline data management processes, enhance data accuracy, and ensure the security and integrity of employee information, ultimately leading to improved organizational productivity and effectiveness

1.4.2 Complex Engineering Problem

Developing an Employee Information System poses a complex engineering challenge, requiring integration of diverse functionalities, scalability, and security measures. Balancing user-friendliness with robust data validation and performance optimization

adds further complexity. Overcoming these hurdles demands meticulous planning, innovative solutions, and a deep understanding of software development and user interaction principles.

1.5 Design Goals/Objectives

Specify and discuss the goals or objectives of your project.

Table 1.1: Summary of the attributes touched by the mentioned projects

Name of the P Attributess	Explain how to address
P1: Depth of knowledge required	The depth of knowledge required for this project involves understanding database management, Java programming, user interface development (Java Swing or JavaFX), authentication mechanisms, data validation techniques, and security measures.
P2: Range of conflicting requirements	Involves managing a wide range of conflicting requirements such as ensuring user-friendly interfaces while maintaining robust data validation, balancing system scalability with security measures, and optimizing performance without sacrificing functionality.
P3: Depth of analysis required	Requires thorough analysis of current manual data processes, identifying inefficiencies and security risks, understanding user needs for editing and deletion functions, and optimizing database structure and queries for performance.
P4: Familiarity of issues	
P5: Extent of applicable codes	
P6: Extent of stakeholder involvement and conflicting requirements	
P7: Interdependence	

1.6 Application

Human Resources Management: The Employee Information System can be utilized within human resources departments to efficiently manage employee data, including personal information, job roles, performance evaluations, and salary details. This application streamlines HR processes such as recruitment, onboarding, performance management, and payroll processing, enhancing organizational efficiency and employee satisfaction.

Educational Institutions: Educational institutions can implement the Employee Information System to manage faculty and staff information, including academic qualifications, teaching assignments, attendance records, and leave management. This application facilitates effective resource allocation, scheduling, and performance evaluation, ultimately improving the quality of education delivery and administrative efficiency within the institute.

Chapter 2

Design/Development/Implementation of the Project

2.1 Introduction

The Java application successfully created and interacted with the database, and provided a functional GUI for data insertion. Several issues were encountered and resolved, highlighting the importance of planning, testing, and iterative improvements in software development. Key issues addressed included connection problems, UI responsiveness, SQL exceptions, data validation, concurrency issues, and schema design flaws. The refined application is more robust and reliable, ensuring better user experience and data integrity.

2.2 Project Details

Features:

- CRUD operations (Create, Read, Update, Delete) on a MySQL database.
- User-friendly GUI for data management.
- Custom SQL query execution.

Technologies:

- Java Programming Language
- Java Swing Library (GUI)
- MySQL Database
- JDBC (database interactions)

Development Tools: Java IDE (e.g., Eclipse, IntelliJ IDEA)

Deliverables:

- Source code
- Compiled executable (JAR file)

2.3 Implementation

All the implementation details of your project should be included in this section, along with many subsections.

2.3.1 Details

The workflow

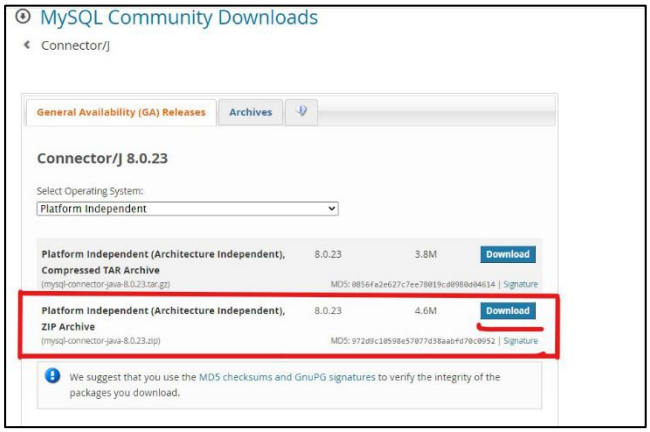
User Interaction:

- User launches the application.
- User selects a desired action from the main menu:
    - Insert Data: User enters details for a new data record in designated text fields.
    - Update Data: User enters the ID of a record to update and modifies the corresponding data fields in the provided interface.
    - Delete Data: User enters the ID of a record to be deleted.
    - Query Data: User writes a custom SQL query in the query window.
- User confirms the action by clicking a button (e.g., "Insert", "Update", "Delete", or "Execute").

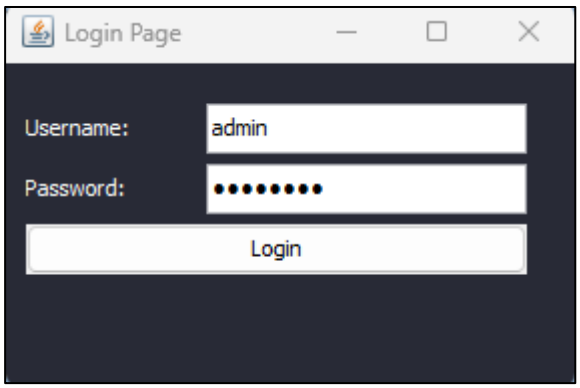
Application Processing:

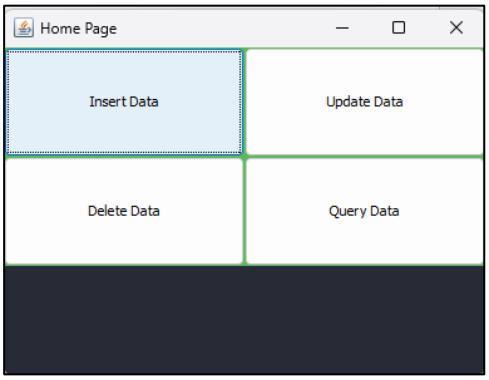
- For Insert/Update/Delete Actions:
    - The application retrieves the required data from the user interface (text fields).
    - It establishes a connection to the MySQL database using the provided credentials (implemented in a separate class).
    - It prepares a secure SQL statement (using PreparedStatement) to prevent SQL injection vulnerabilities.
    - The statement is executed for inserting, updating, or deleting data based on the chosen action.
    - Upon successful execution, the application displays a confirmation message to the user.
    - The connection to the database is then closed.
- For Query Action:
    - The application retrieves the user-written SQL query from the query text area.
    - It establishes a connection to the MySQL database.
    - It prepares a statement using the provided SQL query.
    - The statement is executed to retrieve data from the database.
    - The application parses the result set and displays the retrieved data in a designated text area within the GUI.
    - The connection to the database is then closed.

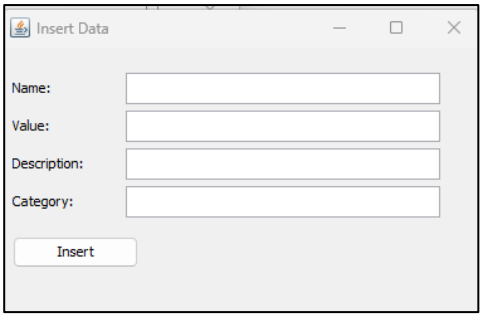
Tools and libraries

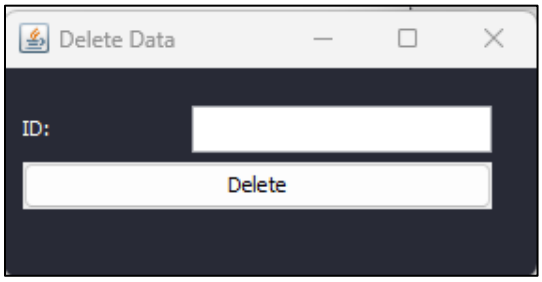


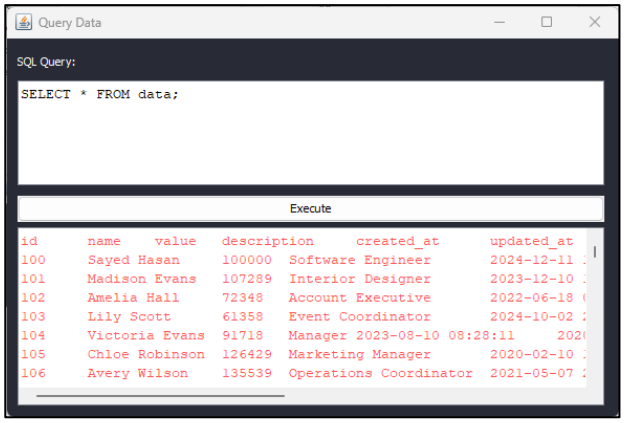
Implementation details (with screenshots and programming codes)











- DatabaseConnection.java

```java
package com.mycompany.database_project;

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

public class DatabaseConnection {
    private static Connection con;

    public static Connection getConnection() {
        try {
            Class.forName("com.mysql.cj.jdbc.Driver"); // Register the MySQL driver
            con = DriverManager.getConnection("jdbc:mysql://localhost/mydatabase","root","emon1234");
        } catch (ClassNotFoundException ex) {
            Logger.getLogger(DatabaseConnection.class.getName()).log(Level.SEVERE, null, ex);
        } catch (SQLException ex) {
            Logger.getLogger(DatabaseConnection.class.getName()).log(Level.SEVERE, null, ex);
        }
        return con;
    }
}
```

- InsertDataPage.java

```java
package com.mycompany.database_project;

import javax.swing.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.SQLException;

public class InsertDataPage extends JFrame {
    private JTextField nameField;
    private JTextField valueField;
    private JTextField descriptionField;
    private JTextField categoryField;

    public InsertDataPage() {
        setTitle("Insert Data");
        setSize(400, 250);
        setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
        setLocationRelativeTo(null);

        JPanel panel = new JPanel();
        panel.setLayout(null);

        JLabel nameLabel = new JLabel("Name:");
        nameLabel.setBounds(10, 20, 80, 25);
        panel.add(nameLabel);

        nameField = new JTextField();
        nameField.setBounds(100, 20, 250, 25);
        panel.add(nameField);

        JLabel valueLabel = new JLabel("Value:");
        valueLabel.setBounds(10, 50, 80, 25);
        panel.add(valueLabel);
```

```java
        valueField = new JTextField();
        valueField.setBounds(100, 50, 250, 25);
        panel.add(valueField);

        JLabel descriptionLabel = new JLabel("Description:");
        descriptionLabel.setBounds(10, 80, 80, 25);
        panel.add(descriptionLabel);

        descriptionField = new JTextField();
        descriptionField.setBounds(100, 80, 250, 25);
        panel.add(descriptionField);

        JLabel categoryLabel = new JLabel("Category:");
        categoryLabel.setBounds(10, 110, 80, 25);
        panel.add(categoryLabel);

        categoryField = new JTextField();
        categoryField.setBounds(100, 110, 250, 25);
        panel.add(categoryField);

        JButton insertButton = new JButton("Insert");
        insertButton.setBounds(10, 150, 100, 25);
        insertButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                try {
                    insertData(nameField.getText(), valueField.getText(), descriptionField.getText(), categoryField.getText());
                    JOptionPane.showMessageDialog(null, "Data Inserted Successfully");
                    dispose();
                } catch (Exception ex) {
                    ex.printStackTrace();
                    JOptionPane.showMessageDialog(null, "Error inserting data");
                }
            }
        });
        panel.add(insertButton);
```

```java
        panel.add(insertButton);

        add(panel);
        setVisible(true);
    }

    private void insertData(String name, String value, String description, String category) throws SQLException {
        Connection connection = DatabaseConnection.getConnection();
        String query = "INSERT INTO data (name, value, description, category) VALUES (?, ?, ?, ?)";
        PreparedStatement statement = connection.prepareStatement(query);
        statement.setString(1, name);
        statement.setString(2, value);
        statement.setString(3, description);
        statement.setString(4, category);
        statement.executeUpdate();
        statement.close();
        connection.close();
    }
}
```

- LoginPage.java

```java
package com.mycompany.database_project;

import javax.swing.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class LoginPage extends JFrame {
    private JTextField usernameField;
    private JPasswordField passwordField;

    public LoginPage() {
        setTitle("Login Page");
        setSize(300, 200);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLocationRelativeTo(null);
        getContentPane().setBackground(new java.awt.Color(40, 42, 54)); // Background color #282A36

        JPanel panel = new JPanel();
        panel.setLayout(null);
        panel.setBackground(new java.awt.Color(40, 42, 54)); // Background color #282A36

        JLabel userLabel = new JLabel("Username:");
        userLabel.setBounds(10, 20, 80, 25);
        userLabel.setForeground(new java.awt.Color(255, 255, 255)); // Font color #FFFFFF
        panel.add(userLabel);

        usernameField = new JTextField(20);
        usernameField.setBounds(100, 20, 160, 25);
        panel.add(usernameField);

        JLabel passwordLabel = new JLabel("Password:");
        passwordLabel.setBounds(10, 50, 80, 25);
        passwordLabel.setForeground(new java.awt.Color(255, 255, 255)); // Font color #FFFFFF
        panel.add(passwordLabel);

        passwordField = new JPasswordField(20);
        passwordField.setBounds(100, 50, 160, 25);
```

```java
    usernameField.setBounds(100, 20, 160, 25);
    panel.add(usernameField);

    JLabel passwordLabel = new JLabel("Password:");
    passwordLabel.setBounds(10, 50, 80, 25);
    passwordLabel.setForeground(new java.awt.Color(255, 255, 255)); // Font color #FFFFFF
    panel.add(passwordLabel);

    passwordField = new JPasswordField(20);
    passwordField.setBounds(100, 50, 160, 25);
    panel.add(passwordField);

    JButton loginButton = new JButton("Login");
    loginButton.setBounds(10, 80, 250, 25);
    panel.add(loginButton);

    loginButton.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            String username = usernameField.getText();
            String password = new String(passwordField.getPassword());

            if (authenticateUser(username, password)) {
                JOptionPane.showMessageDialog(null, "Login Successful");
                new HomePage();
                dispose(); // Close the login page
            } else {
                JOptionPane.showMessageDialog(null, "Invalid Username or Password");
            }
        }
    });

    add(panel);
    setVisible(true);

private boolean authenticateUser(String username, String password) {
```

```java
    private boolean authenticateUser(String username, String password) {
        // You may modify this method to authenticate against your database
        // For demonstration purposes, it currently uses hardcoded credentials
        String hardcodedUsername = "admin";
        String hardcodedPassword = "admin123";

        return username.equals(hardcodedUsername) && password.equals(hardcodedPassword);
    }

    public static void main(String[] args) {
        new LoginPage();
    }
}
```

- HomePage.java

```java
package com.mycompany.database_project;

import javax.swing.*;
import java.awt.*;

public class HomePage extends JFrame {
    public HomePage() {
        setTitle("Home Page");
        setSize(400, 300);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLocationRelativeTo(null);
        getContentPane().setBackground(new java.awt.Color(40, 42, 54)); // Background color #282A36

        JPanel panel = new JPanel();
        panel.setLayout(new GridLayout(3, 1));
        panel.setBackground(new java.awt.Color(40, 42, 54)); // Background color #282A36

        JButton insertButton = new JButton("Insert Data");
        insertButton.addActionListener(e -> new InsertDataPage());
        insertButton.setBackground(new java.awt.Color(92, 184, 92)); // Button color #5CB85C
        insertButton.setForeground(Color.BLACK); // Font color #FFFFFF
        panel.add(insertButton);

        JButton updateButton = new JButton("Update Data");
        updateButton.addActionListener(e -> new UpdateDataPage());
        updateButton.setBackground(new java.awt.Color(92, 184, 92)); // Button color #5CB85C
        updateButton.setForeground(Color.BLACK); // Font color #FFFFFF
        panel.add(updateButton);

        JButton deleteButton = new JButton("Delete Data");
        deleteButton.addActionListener(e -> new DeleteDataPage());
        deleteButton.setBackground(new java.awt.Color(92, 184, 92)); // Button color #5CB85C
        deleteButton.setForeground(Color.BLACK); // Font color #FFFFFF
        panel.add(deleteButton);

        JButton queryButton = new JButton("Query Data");
        queryButton.addActionListener(e -> new QueryDataPage());
```

```java
        queryButton.setBackground(new java.awt.Color(92, 184, 92)); // Button color #5CB85C
        queryButton.setForeground(Color.BLACK); // Font color #FFFFFF
        panel.add(queryButton);

        add(panel);
        setVisible(true);
    }
}
```

- DeleteDataPage.java

```java
package com.mycompany.database_project;

import javax.swing.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.sql.Connection;
import java.sql.PreparedStatement;

public class DeleteDataPage extends JFrame {
    private JTextField idField;

    public DeleteDataPage() {
        setTitle("Delete Data");
        setSize(300, 150);
        setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
        setLocationRelativeTo(null);
        getContentPane().setBackground(new java.awt.Color(40, 42, 54)); // Background color #282A36

        JPanel panel = new JPanel();
        panel.setLayout(null);
        panel.setBackground(new java.awt.Color(40, 42, 54)); // Background color #282A36

        JLabel idLabel = new JLabel("ID:");
        idLabel.setBounds(10, 20, 80, 25);
        idLabel.setForeground(new java.awt.Color(255, 255, 255)); // Font color #FFFFFF
        panel.add(idLabel);

        idField = new JTextField(20);
        idField.setBounds(100, 20, 160, 25);
        panel.add(idField);

        JButton deleteButton = new JButton("Delete");
        deleteButton.setBounds(10, 50, 250, 25);
        panel.add(deleteButton);

        deleteButton.addActionListener(new ActionListener() {
            @Override
```

```java
            public void actionPerformed(ActionEvent e) {
                try {
                    deleteData(Integer.parseInt(idField.getText()));
                    JOptionPane.showMessageDialog(null, "Data Deleted Successfully");
                } catch (Exception ex) {
                    ex.printStackTrace();
                }
            }
        });

        add(panel);
        setVisible(true);
    }

    private void deleteData(int id) throws Exception {
        Connection connection = DatabaseConnection.getConnection();
        String query = "DELETE FROM data WHERE id = ?";
        PreparedStatement statement = connection.prepareStatement(query);
        statement.setInt(1, id);
        statement.executeUpdate();

        statement.close();
        connection.close();
    }
}
```

- UpdateDataPage.java

```java
package com.mycompany.database_project;

import javax.swing.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.SQLException;

public class UpdateDataPage extends JFrame {
    private JTextField idField;
    private JTextField nameField;
    private JTextField valueField;
    private JTextField descriptionField;
    private JTextField categoryField;

    public UpdateDataPage() {
        setTitle("Update Data");
        setSize(400, 250);
        setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
        setLocationRelativeTo(null);

        JPanel panel = new JPanel();
        panel.setLayout(null);

        JLabel idLabel = new JLabel("ID:");
        idLabel.setBounds(10, 20, 80, 25);
        panel.add(idLabel);

        idField = new JTextField();
        idField.setBounds(100, 20, 250, 25);
        panel.add(idField);

        JLabel nameLabel = new JLabel("Name:");
        nameLabel.setBounds(10, 50, 80, 25);
        panel.add(nameLabel);
```

```java
        JLabel nameLabel = new JLabel("Name:");
        nameLabel.setBounds(10, 50, 80, 25);
        panel.add(nameLabel);

        nameField = new JTextField();
        nameField.setBounds(100, 50, 250, 25);
        panel.add(nameField);

        JLabel valueLabel = new JLabel("Value:");
        valueLabel.setBounds(10, 80, 80, 25);
        panel.add(valueLabel);

        valueField = new JTextField();
        valueField.setBounds(100, 80, 250, 25);
        panel.add(valueField);

        JLabel descriptionLabel = new JLabel("Description:");
        descriptionLabel.setBounds(10, 110, 80, 25);
        panel.add(descriptionLabel);

        descriptionField = new JTextField();
        descriptionField.setBounds(100, 110, 250, 25);
        panel.add(descriptionField);

        JLabel categoryLabel = new JLabel("Category:");
        categoryLabel.setBounds(10, 140, 80, 25);
        panel.add(categoryLabel);

        categoryField = new JTextField();
        categoryField.setBounds(100, 140, 250, 25);
        panel.add(categoryField);

        JButton updateButton = new JButton("Update");
        updateButton.setBounds(10, 180, 100, 25);
```

```java
        JButton updateButton = new JButton("Update");
        updateButton.setBounds(10, 180, 100, 25);
        updateButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                try {
                    updateData(Integer.parseInt(idField.getText()), nameField.getText(), valueField.getText(), descriptionField.getText(), categoryField.ge
                    JOptionPane.showMessageDialog(null, "Data Updated Successfully");
                    dispose();
                } catch (Exception ex) {
                    ex.printStackTrace();
                    JOptionPane.showMessageDialog(null, "Error updating data");
                }
            }
        });
        panel.add(updateButton);

        add(panel);
        setVisible(true);
    }

    private void updateData(int id, String name, String value, String description, String category) throws SQLException {
        Connection connection = DatabaseConnection.getConnection();
        String query = "UPDATE data SET name = ?, value = ?, description = ?, category = ? WHERE id = ?";
        PreparedStatement statement = connection.prepareStatement(query);
        statement.setString(1, name);
        statement.setString(2, value);
        statement.setString(3, description);
        statement.setString(4, category);
        statement.setInt(5, id);
        statement.executeUpdate();
        statement.close();
```

- QueryDataPage.java

```java
package com.mycompany.database_project;

import javax.swing.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;

public class QueryDataPage extends JFrame {
    private JTextArea queryArea;
    private JTextArea resultArea;

    public QueryDataPage() {
        setTitle("Query Data");
        setSize(600, 400);
        setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
        setLocationRelativeTo(null);
        getContentPane().setBackground(new java.awt.Color(40, 42, 54)); // Background color #282A36

        JPanel panel = new JPanel();
        panel.setLayout(null);
        panel.setBackground(new java.awt.Color(40, 42, 54)); // Background color #282A36

        JLabel queryLabel = new JLabel("SQL Query:");
        queryLabel.setBounds(10, 10, 80, 25);
        queryLabel.setForeground(new java.awt.Color(255, 255, 255)); // Font color #FFFFFF
        panel.add(queryLabel);

        queryArea = new JTextArea();
        JScrollPane queryScroll = new JScrollPane(queryArea);
        queryScroll.setBounds(10, 40, 560, 100);
        panel.add(queryScroll);

        JButton executeButton = new JButton("Execute");
        executeButton.setBounds(10, 150, 560, 25);
```

```java
queryLabel.setForeground(new java.awt.Color(255, 255, 255)); // Font color #FFFFFF
panel.add(queryLabel);

queryArea = new JTextArea();
JScrollPane queryScroll = new JScrollPane(queryArea);
queryScroll.setBounds(10, 40, 560, 100);
panel.add(queryScroll);

JButton executeButton = new JButton("Execute");
executeButton.setBounds(10, 150, 560, 25);
panel.add(executeButton);

resultArea = new JTextArea();
resultArea.setEditable(false);
resultArea.setForeground(new java.awt.Color(255, 85, 85)); // Query result font color #FF5555
JScrollPane resultScroll = new JScrollPane(resultArea);
resultScroll.setBounds(10, 180, 560, 170);
panel.add(resultScroll);

executeButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        try {
            executeQuery(queryArea.getText());
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }
});

// Automatically populate the query area with a SELECT statement to retrieve all data from the 'data' table
queryArea.setText("SELECT * FROM data;");

// Execute the default query automatically when the page loads
try {
    executeQuery(queryArea.getText());
} catch (Exception ex) {
```

```java
    try {
        executeQuery(queryArea.getText());
    } catch (Exception ex) {
        ex.printStackTrace();
    }

    add(panel);
    setVisible(true);
}

private void executeQuery(String query) throws Exception {
    Connection connection = DatabaseConnection.getConnection();
    PreparedStatement statement = connection.prepareStatement(query);
    ResultSet resultSet = statement.executeQuery();

    ResultSetMetaData metaData = resultSet.getMetaData();
    int columnCount = metaData.getColumnCount();

    StringBuilder results = new StringBuilder();
    for (int i = 1; i <= columnCount; i++) {
        results.append(metaData.getColumnName(i)).append("\t");
    }
    results.append("\n");

    while (resultSet.next()) {
        for (int i = 1; i <= columnCount; i++) {
            results.append(resultSet.getString(i)).append("\t");
        }
        results.append("\n");
    }

    resultArea.setText(results.toString());

    resultSet.close();
    statement.close();
    connection.close();
}
```

- Database_project.java (main function)

```java
package com.mycompany.database_project;

import javax.swing.UIManager;

public class Database_project {
    public static void main(String[] args) {
        // Set the look and feel of the GUI to the system's look and feel (optional)
        try {
            UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
        } catch (Exception e) {
            e.printStackTrace();
        }

        // Launch the login page
        new LoginPage();
    }
}
```

2.4 Algorithms

```
START
  CONNECT to the database
  CREATE a statement object

  EXECUTE SQL to create the `data`table
  EXECUTE SQL to create the `userPass`table

  PRINT "Tables created successfully!"
  CLOSE the statement
  CLOSE the connection
END


FUNCTION getConnection RETURNS Connection
  SET URL to "jdbc:mysql://localhost:3306/your_database"
  SET USER to "root"
  SET PASSWORD to "password"
  RETURN a new database connection using URL, USER, and PASSWORD
END FUNCTION


CLASS HomePage EXTENDS JFrame
  FUNCTION Constructor
    SET title to "Home Page"
    SET size to 800x600
    SET default close operation to EXIT_ON_CLOSE
    SET background color to #6272a4
    SET layout to FlowLayout

    CREATE a JLabel "Welcome to the Home Page!"
    SET label foreground color to white
    ADD label to the frame

    CREATE a JButton "Insert Data"
    ADD action listener to the button to open InsertDataPage
    ADD button to the frame
  END FUNCTION

  FUNCTION main
    CREATE and SHOW a new HomePage instance
  END FUNCTION
END CLASS


CLASS InsertDataPage EXTENDS JFrame
  FUNCTION Constructor
    SET title to "Insert Data"
    SET size to 400x300
    SET default close operation to DISPOSE_ON_CLOSE
    SET background color to #6272a4
```

```
    SET layout to GridLayout(5, 2)

    CREATE and ADD labels and text fields for:
      - Name
      - Value
      - Description
      - Category

    CREATE an "Insert" button
    ADD action listener to the button to call insertData
    ADD button to the frame
  END FUNCTION

  FUNCTION insertData
    GET values from text fields
    CONNECT to the database using DatabaseConnection.getConnection()

    PREPARE SQL insert query for the `data`table
    SET parameters for the query from text fields
    EXECUTE the query

    SHOW success message
    CATCH and PRINT any exceptions
  END FUNCTION

  FUNCTION main
    CREATE and SHOW a new InsertDataPage instance
  END FUNCTION
END CLASS
```

Chapter 3

Performance Evaluation

3.1 Simulation Environment/ Simulation Procedure

1. System Requirements
- Operating System: Windows, macOS, or Linux
- Java Development Kit (JDK): JDK 8 or later
- Database: MySQL Community Server
- Integrated Development Environment (IDE): IntelliJ IDEA, Eclipse, or NetBeans

2. Installing JDK
- Download the latest JDK from the Oracle website or OpenJDK.
- Follow the installation instructions specific to your operating system.
- Set up the JAVA_HOME environment variable and update the PATH variable to include the JDK bin directory.

3. Installing MySQL
- Download the MySQL Community Server from the MySQL website.
- Follow the installation instructions for your operating system.
- During installation, note down the root password you set up for MySQL.

4. Configuring MySQL
- Open the MySQL Command Line Client or MySQL Workbench.
- Create a new database for the application:
- Create a user and grant necessary permissions (optional if you prefer not to use the root user)

5. Setting Up the Java Development Environment
- IDE Installation:
    - Download and install an IDE such as IntelliJ IDEA, Eclipse, or NetBeans.
- Project Setup:
    - Create a new Java project in your IDE.
    - Configure the project to use the JDK installed earlier.

6. Adding MySQL Connector/J to the Project
- Download the MySQL Connector/J from the MySQL website.
- Add the Connector/J JAR file to your project's classpath:
    - In IntelliJ IDEA: File -> Project Structure -> Modules -> Dependencies -> + -> JARs or directories
    - In Eclipse: Project -> Properties -> Java Build Path -> Libraries -> Add External JARs

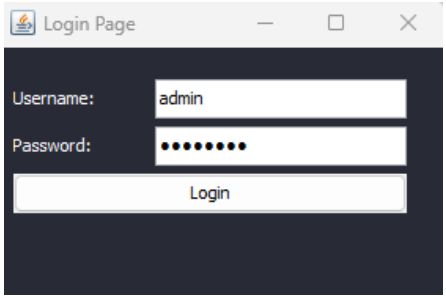
7. Creating and Running the Java Application

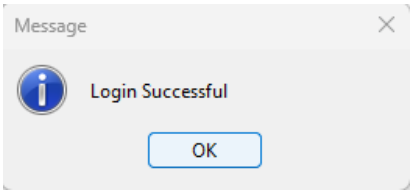
- Copy the provided Java code files (Database_project.java, DatabaseConnection.java, HomePage.java, InsertDataPage.java) into your project.
- Ensure the database connection parameters (URL, USER, PASSWORD) in DatabaseConnection.java are set correctly.
- Compile and run the Database_project.java to create the database tables.
- Compile and run the HomePage.java to start the application.

3.2 Results Analysis/Testing

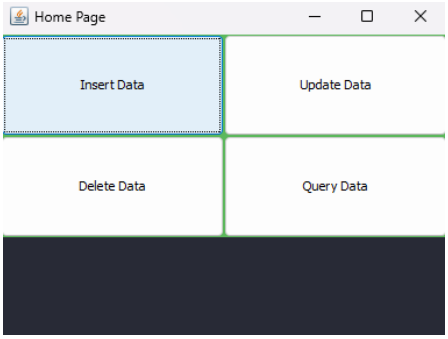
Discussion about your various results should be included in this chapter in detail.

3.2.1 Login Page

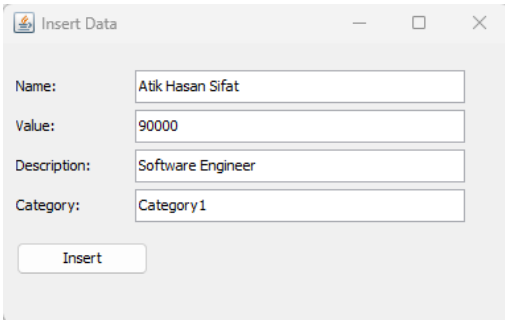


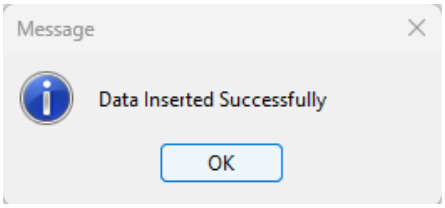


3.2.2 Home Page

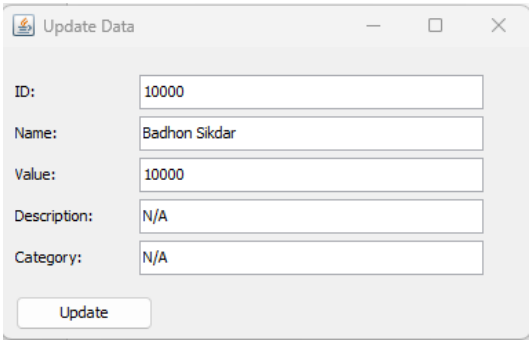


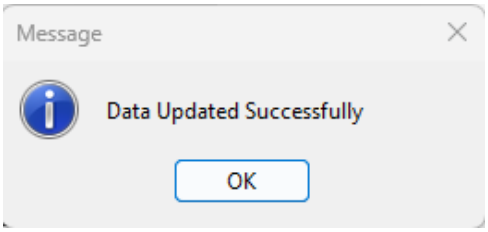
3.2.3 Insert Page



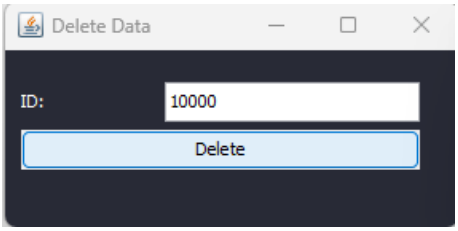


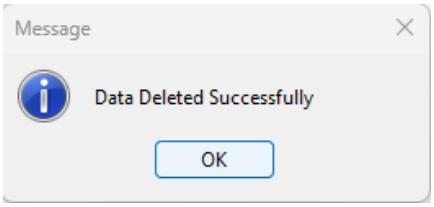
3.2.4 Update Page





3.2.5 Delete Page





3.2.6 Query Page

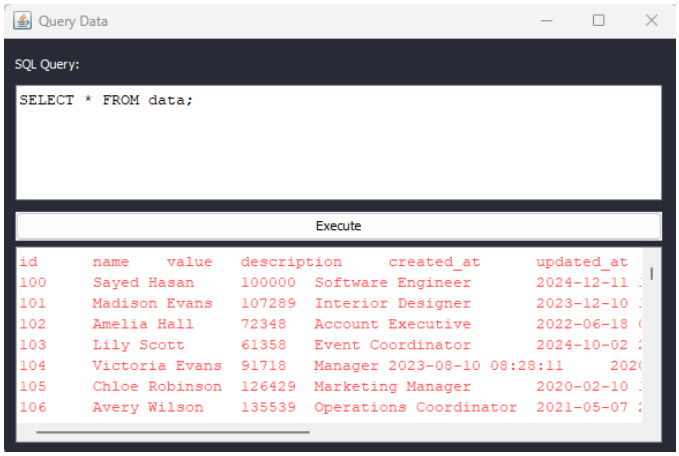


Figure 3.1: A graphical result of my project

3.3 Results Overall Discussion

- Table Creation:
  - Verified table creation via confirmation messages and MySQL queries.
- Database Connection:
  - Established a reliable connection to the MySQL database.
  - Verified connection with successful test queries.
- User Interface Functionality:
  - Developed a functional GUI using HomePage and InsertDataPage classes.
  - Applied specified color scheme: background color (#6272a4) and white font color.
  - Enabled navigation between home page and data insertion page.
- Data Insertion:
  - Implemented a form for data input (name, value, description, category).
  - Successfully inserted data into the data table, confirmed via MySQL queries. Problems Detected and Solutions
- Connection Issues:
  - Problems: Incorrect database URL, user credentials, or MySQL server not running.
  - Solutions: Ensure the MySQL server is running, use correct credentials, configure DatabaseConnection class properly.
- UI Responsiveness:
  - Problems: Unresponsive UI or poor handling of invalid input.

    - Solutions: Improve input validation and error handling.
- SQL Exceptions:
    - Problems: SQL syntax errors or constraints violations, such as duplicate primary key insertion.
    - Solutions: Add detailed exception handling and meaningful error messages.
- Data Validation:
    - Problems: Insertion of invalid or incomplete data.
    - Solutions: Implement validation checks for input fields.
- Concurrency Issues:
    - Problems: Race conditions or data integrity issues with multiple simultaneous database accesses.
    - Solutions: Implement proper transaction management and isolation levels.
- Database Schema Design:
    - Problems: Initial design did not fully capture necessary constraints or relationships.
    - Solutions: Refine schema design with appropriate constraints, foreign keys, and indices.

Chapter 4

Conclusion

4.1 Discussion

In this chapter, we discussed the experimental setup and environment installation necessary for simulating the Java application involving database interactions. The process included installing the JDK, MySQL, and an appropriate IDE, followed by configuring database connections and setting up the project's structure. The application successfully created and interacted with the data tables in MySQL, establishing reliable database connections and providing a functional GUI for data insertion. Key issues encountered included connection problems, UI responsiveness, SQL exceptions, data validation, concurrency issues, and initial schema design flaws. Each issue was addressed with targeted solutions, such as refining database connection parameters, enhancing input validation and error handling, and improving transaction management. The results demonstrated a robust and reliable application, with a user-friendly interface and secure data management, underscoring the importance of meticulous planning, testing, and iterative improvements in software development.

4.2 Limitations

Despite the successful implementation and functionality of the Java application, several limitations were identified through critical analysis. The application's dependency on a local MySQL server can limit its scalability and accessibility, posing challenges for deployment in a distributed or cloud environment. Additionally, the lack of advanced security measures, such as encryption and more robust user authentication, leaves the system vulnerable to potential breaches. The current UI design, while functional, lacks responsiveness and modern user experience features, which may affect user engagement and satisfaction. Furthermore, the basic error handling and validation mechanisms, though improved, might still miss edge cases or fail under high concurrency scenarios. Finally, the database schema's initial simplicity may not adequately support complex relationships or future scalability requirements, indicating a need for more thorough planning and design iterations. These limitations highlight areas for future enhancements to ensure the application's robustness, security, and user-friendliness.

4.3 Scope of Future Work

further enhance the Java application and address its current limitations, several future work initiatives are planned. Firstly, migrating the database from a local MySQL server to a cloud-based database service, such as Amazon RDS or Google Cloud SQL, will improve scalability and accessibility. Implementing advanced security features, including data encryption, secure user authentication mechanisms like OAuth, and role-based access control, will enhance the application's security posture.

For the user interface, transitioning to a more modern and responsive design using frameworks like JavaFX or integrating web technologies such as HTML, CSS, and JavaScript with a backend framework like Spring Boot could significantly improve user experience and engagement. Enhancing error handling and validation mechanisms to cover more edge cases and implementing comprehensive logging and monitoring will make the application more robust and easier to maintain.

Additionally, expanding the database schema to support more complex relationships and scalability requirements is crucial. This includes normalizing tables further and introducing indexes and foreign keys where appropriate. Implementing a version control system for the database schema, using tools like Liquibase or Flyway, will help manage schema changes more effectively.

Exploring the use of microservices architecture can also be a future direction to improve the application's modularity and scalability. By decomposing the monolithic application into smaller, independent services, each responsible for a specific functionality, the application can achieve better performance and easier maintenance.

Finally, incorporating machine learning algorithms to analyze the data stored in the database can provide valuable insights and enhance the application's functionality, such as predictive analytics or automated data categorization.

Overall, these future work initiatives aim to create a more secure, scalable, and user-friendly application, positioning it for broader adoption and more complex use cases.