GUI FOR DATABASE COMMANDS

An

Object-Oriented Programming Concepts through Java

Course Project Report in partial fulfillment of the deg

BACHELOR OF TECHNOLOGY

in

ARTIFICIAL INTELLIGENCE AND MACHINELEARNING

By

Sriram Thota	2103A52070
Pranay Sripada	2103A52170

Under the Guidance of

Dr. T SAMPATH KUMAR

School of Computer Science and AI

Submitted to
School of CS & AI

SR UNIVERSITY

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

CERTIFICATE

This is to certify that the **Object Oriented Programming through Java- Course Project** Report entitled **"GUI FOR DATABASE COMMANDS"** is a record of bonafide work carried out by the students**, SRIRAM THOTA, PRANAY SRIPADA** bearing RollNo(s) 2103A52070,2103A52170 during the academic year 2023-2024in partial fulfillment of the award of the degree of **Bachelor of Technology** in **Computer Science& Engineering** by the SR University, Hasanparthy, Warangal.

Project Guide **Head of the Department**

ACKNOWLEDGEMENT

We owe an enormous debt of gratitude to our project guide **Dr. T Sampath Kumar,** as well as Head of the CSE Department **Dr. M. Sheshikala**, AssociateProfessor for guiding us from the beginning through the end of the Capstone Project with their intellectual advices and insightful suggestions. We truly value their consistent feedback on our progress, which always constructive and encouraging and ultimately drove us to the right direction.

We wish to take this opportunity to express our sincere gratitude and deep senseof respect to our beloved Principal, **Dr.V.MAHESH,** for his continuous supportand guidance to complete this project.

Finally, we express our thanks to all the teaching and non-teaching staff of the department for their suggestions and timely support.

TABLE OF CONTENTS

1. ABSTRACT

The Graphical User Interface (GUI) for database commands represents a user-friendly and intuitive solution designed to streamline and enhance the interaction between users and relational databases. This innovative interface serves as a visual gateway to execute complex database commands, offering a more accessible alternative to traditional command-line interfaces. Through the GUI, users can effortlessly manipulate and manage databases by employing a range of commands, from basic queries to intricate transactions. The interface incorporates visually intuitive elements, such as drag-and-drop functionality, graphical representations of database structures, and interactive forms, making it exceptionally user-friendly even for individuals with limited programming or database management experience. Furthermore, the GUI provides real-time feedback, highlighting syntax errors and offering suggestions to improve query construction, thereby reducing the learning curve for users. The implementation of this GUI for database commands not only enhances the efficiency of database management tasks but also promotes a more inclusive and user-centric approach to data manipulation and retrieval.

2. OBJECTIVE

The objective of implementing a Graphical User Interface (GUI) for database commands is to enhance user accessibility and streamline the interaction with the underlying database system. By creating an intuitive and user-friendly interface, the goal is to empower users, regardless of their level of technical expertise, to effortlessly execute database commands. This GUI aims to provide a visually appealing and responsive platform that simplifies the complexities of database management, allowing users to perform tasks such as querying, updating, and managing data with ease. The primary focus is on improving the overall user experience, reducing the learning curve associated with database operations, and promoting efficient and error-free database interactions through a visually guided and logically organized interface. This initiative seeks to bridge the gap between users and databases, fostering a more efficient and enjoyable interaction while maintaining the robustness and reliability of the underlying database system

3.DEFINITIONS OF THE ELEMENTS USED IN PROJECT

1. **JTextField**: The object of a JTextField class is a text component that allowsthe editing of a single-line text. It inherits JTextComponent class.

2. **JLabel:** The object of JLabel class is a component for placing text in a container. It is used to display a single line of read-only text. The text can bechanged by an application but a user cannot edit it directly. It inherits JComponent class.

3. **JButton**: The JButton class is used to create a labeled button that has a platform-independent implementation. The application results in some action when the button is pushed. It inherits AbstractButton class.

4. **JFrame:** JFrame class is a type of container which inherits java. awt.Frameclass. JFrame works the main window where components like labels, buttons, and text fields are added to create a GUI.

5. **ActionListener:** The Java Action Listener is notified whenever you click on the buttonor menu item. It is notified against Action Event. The Action Listener interface is found in java. awt. event package. it has only one method: action Performed().

6. **Swings:** Java swing is a part of Java foundation classes (JFC) that is used to createwindow-based applications It is built on top of AWT (abstract window tool kit)API and entirely in java.

7. **Action Event:** An action event occurs, whenever an action is performed by the user.Examples: When the user clicks a button, chooses a menu item, or presses Enter in a text field. The result is that an action Performed message is sent to all action listeners that are registered on the relevant component

8. Exceptions: An Exception is an Action Event: An action event occurs, wheneveran action is performed by the user. Examples: When the user clicks a button, chooses a menu item, or presses Enter in a text field. The result is that an action Performed message is sent to all action listeners that are registered on the relevant component.

9. Exceptional Handling: Exception handling is a programming concept that involves the systematic management and response to exceptional conditions or events that may occur during the execution of a program. These exceptional conditions, often referred to as "exceptions," represent situations where the normal flow of a program is disrupted due to errors, unexpected events, or invalid inputs. Exception handling provides a structured way to detect, handle, and recover from such exceptional conditions, preventing the program from terminating .

4.IMPLEMENTATION

CODE:

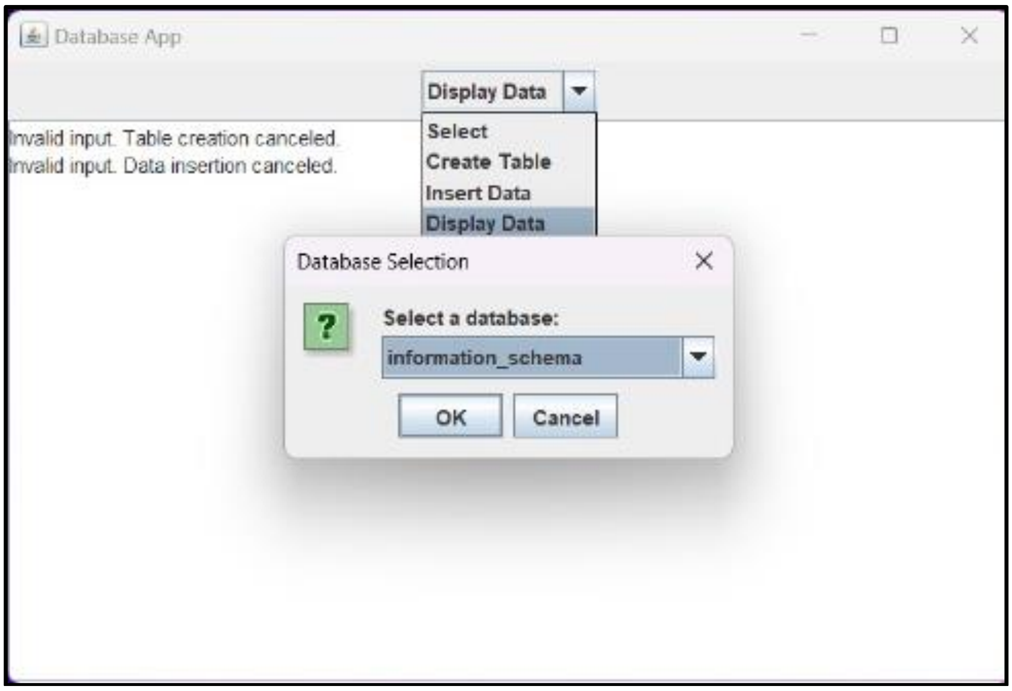
```
import java.sql.*;
import java.util.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.border.EmptyBorder;
import java.util.List;


public class SwingDatabaseApp{
private static final String URL =
"jdbc:mysql://localhost:3306/sriramnani?user=root&password=balaraju1@T";
private Connection connection;
private JFrame frame;
private JTextArea outputTextArea;

public SwingDatabaseApp() {
initializeUI();
connectToDatabase();
}

private void initializeUI() {
frame = new JFrame("Database App");
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.setSize(600, 400);
frame.setLayout(new BorderLayout());

outputTextArea = new JTextArea();
outputTextArea.setEditable(false);
JScrollPane scrollPane = new JScrollPane(outputTextArea);
frame.add(scrollPane, BorderLayout.CENTER);

JPanel controlPanel = new JPanel(new FlowLayout());

String[] options = {"Select","Create Table", "Insert Data", "Display Data", "Exit"};
JComboBox<String> optionComboBox = new JComboBox<>(options);
controlPanel.add(optionComboBox);

frame.add(controlPanel, BorderLayout.NORTH);

optionComboBox.addActionListener(new ActionListener() {
@Override
public void actionPerformed(ActionEvent e) {
String selectedOption = (String) optionComboBox.getSelectedItem();
if (selectedOption.equals("Create Table")) {
String tableName = JOptionPane.showInputDialog("Enter table name:");
String columnDefinitions = JOptionPane.showInputDialog("Enter column definitions (comma-
separated):");
if (tableName != null && !tableName.isEmpty() && columnDefinitions != null) {
createTable(tableName, columnDefinitions);
} else {
displayMessage("Invalid input. Table creation canceled.");
}
} else if (selectedOption.equals("Insert Data")) {
String tableName = JOptionPane.showInputDialog("Enter table name:");
String values = JOptionPane.showInputDialog("Enter values (comma-separated):");
if (tableName != null && !tableName.isEmpty() && values != null) {
insertData(tableName, values);
} else {
displayMessage("Invalid input. Data insertion canceled.");
}
} else if (selectedOption.equals("Display Data")) {
displayDatabase();
} else if (selectedOption.equals("Exit")) {
closeConnection();
System.exit(0);
}
}
});

frame.setVisible(true);
}

private void connectToDatabase() {
try {
connection = DriverManager.getConnection(URL);
} catch (SQLException e) {
e.printStackTrace();
displayMessage("Error connecting to the database: " + e.getMessage());
}
}

private void createTable(String tableName, String columnDefinitions) {
try {
Statement statement = connection.createStatement();
String createTableSQL = "CREATE TABLE IF NOT EXISTS " + tableName + " (" + columnDefinitions
+ ")";
statement.executeUpdate(createTableSQL);
displayMessage("Table created: " + tableName);
} catch (SQLException e) {
e.printStackTrace();
displayMessage("Error creating table: " + e.getMessage());
}
}
```

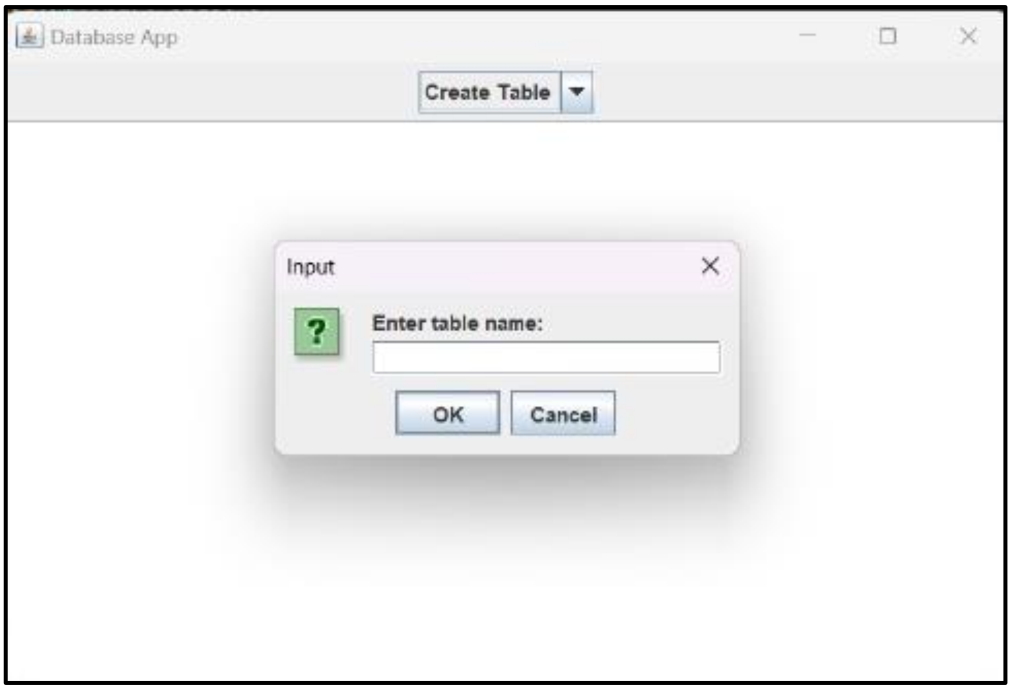
```
private void insertData(String tableName, String values) {
try {
Statement statement = connection.createStatement();
String insertSQL = "INSERT INTO " + tableName + " VALUES (" + values + ")";
statement.executeUpdate(insertSQL);
displayMessage("Data inserted into table: " + tableName);
} catch (SQLException e) {
e.printStackTrace();
displayMessage("Error inserting data: " + e.getMessage());
}
}

private void displayDatabase() {
try {
DatabaseMetaData metaData = connection.getMetaData();
ResultSet databases = metaData.getCatalogs();

List<String> databaseNames = new ArrayList<>();
while (databases.next()) {
String databaseName = databases.getString("TABLE_CAT");
databaseNames.add(databaseName);
}

databases.close();

String[] databaseArray = databaseNames.toArray(new String[0]);
String selectedDatabase = (String) JOptionPane.showInputDialog(frame,
"Select a database:", "Database Selection", JOptionPane.QUESTION_MESSAGE, null, databaseArray,
databaseArray[0]);

if (selectedDatabase != null) {
ResultSet tables = metaData.getTables(selectedDatabase, null, null, new String[]{"TABLE"});

List<String> tableNames = new ArrayList<>();
while (tables.next()) {
String tableName = tables.getString("TABLE_NAME");
tableNames.add(tableName);
}

tables.close();

String[] tableArray = tableNames.toArray(new String[0]);
String selectedTable = (String) JOptionPane.showInputDialog(frame,
"Select a table:", "Table Selection", JOptionPane.QUESTION_MESSAGE, null, tableArray,
tableArray[0]);

if (selectedTable != null) {
displayTableData(selectedDatabase, selectedTable);
} else {
```

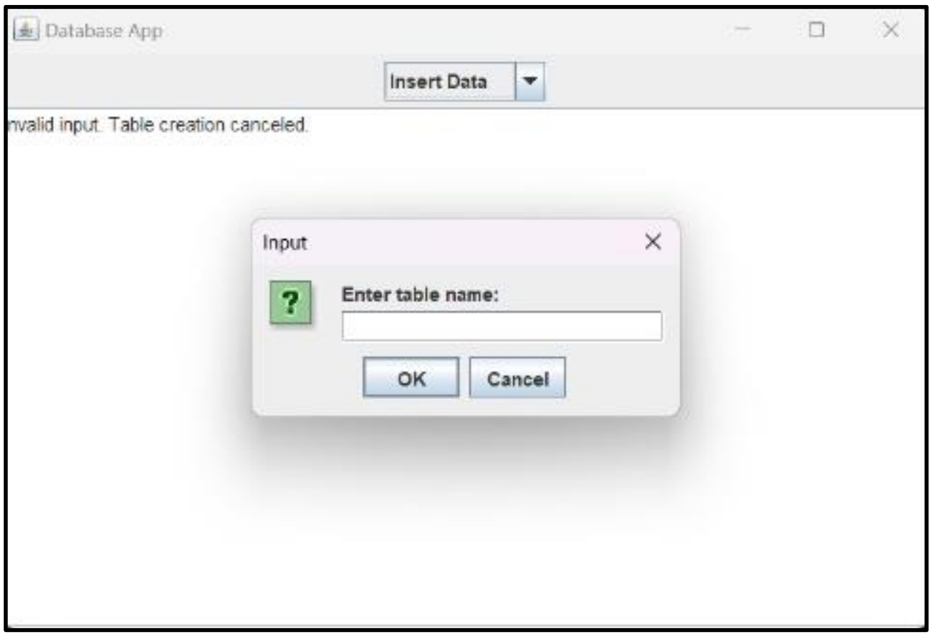
```
displayMessage("Invalid selection. Data display canceled.");
}
} else {
displayMessage("Invalid selection. Data display canceled.");
}
} catch (SQLException e) {
e.printStackTrace();
displayMessage("Error displaying data: " + e.getMessage());
}
}

private void displayTableData(String dbName, String tableName) {
try {
Connection dbConnection = DriverManager.getConnection(URL.replace("/sriramnani", "/" + dbName));
Statement statement = dbConnection.createStatement();
ResultSet resultSet = statement.executeQuery("SELECT * FROM " + tableName);

displayMessage("Displaying data from table: " + tableName);

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

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

while (resultSet.next()) {
StringBuilder row = new StringBuilder();
for (int i = 1; i <= columnCount; i++) {
row.append(resultSet.getString(i)).append("\t");
}
displayMessage(row.toString());
}

resultSet.close();
statement.close();
dbConnection.close();
} catch (SQLException e) {
e.printStackTrace();
displayMessage("Error displaying data: " + e.getMessage());
}
}
```

```
private void displayMessage(String message) {
outputTextArea.append(message + "\n");
}


private void closeConnection() {
try {
if (connection != null) {
connection.close();
}
} catch (SQLException e) {
e.printStackTrace();
displayMessage("Error closing the database connection: " + e.getMessage());
}
}

public static void main(String[] args) {
SwingUtilities.invokeLater(() -> new SwingDatabaseApp());
}
}
```

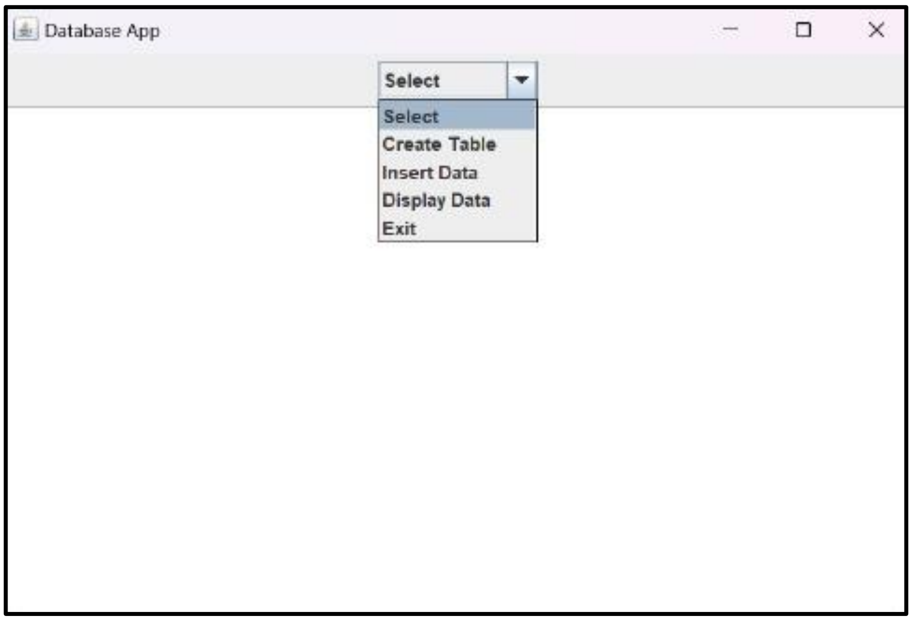
5. RESULT SCREENS

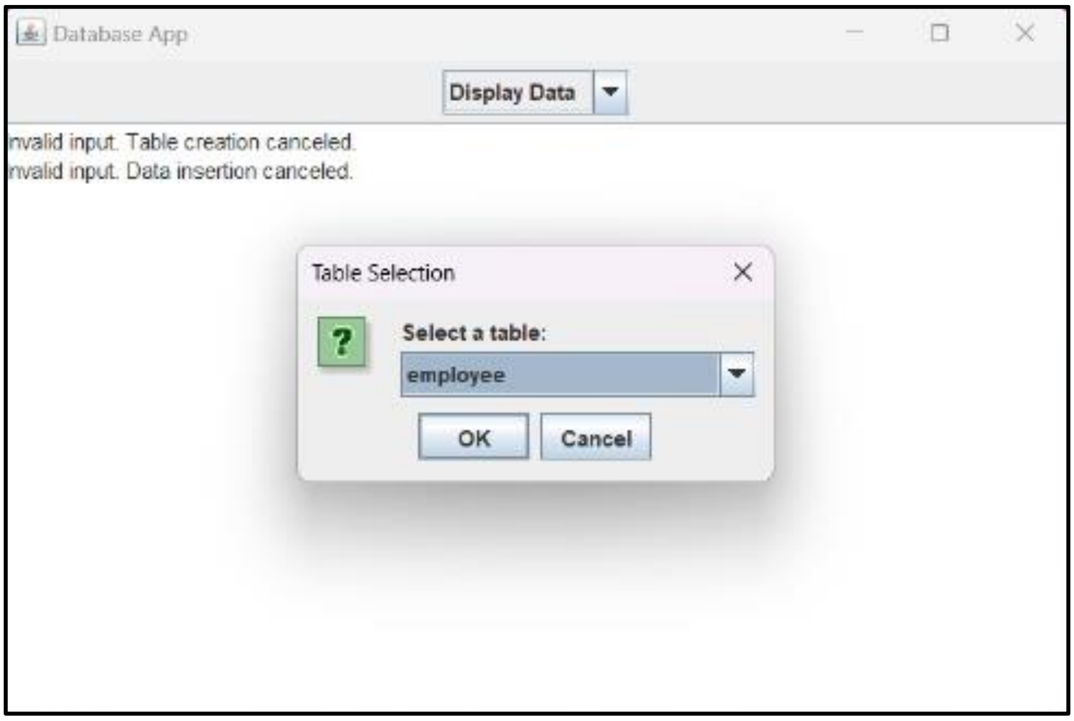

Invalid input. Table creation canceled.
Invalid input. Data insertion canceled.



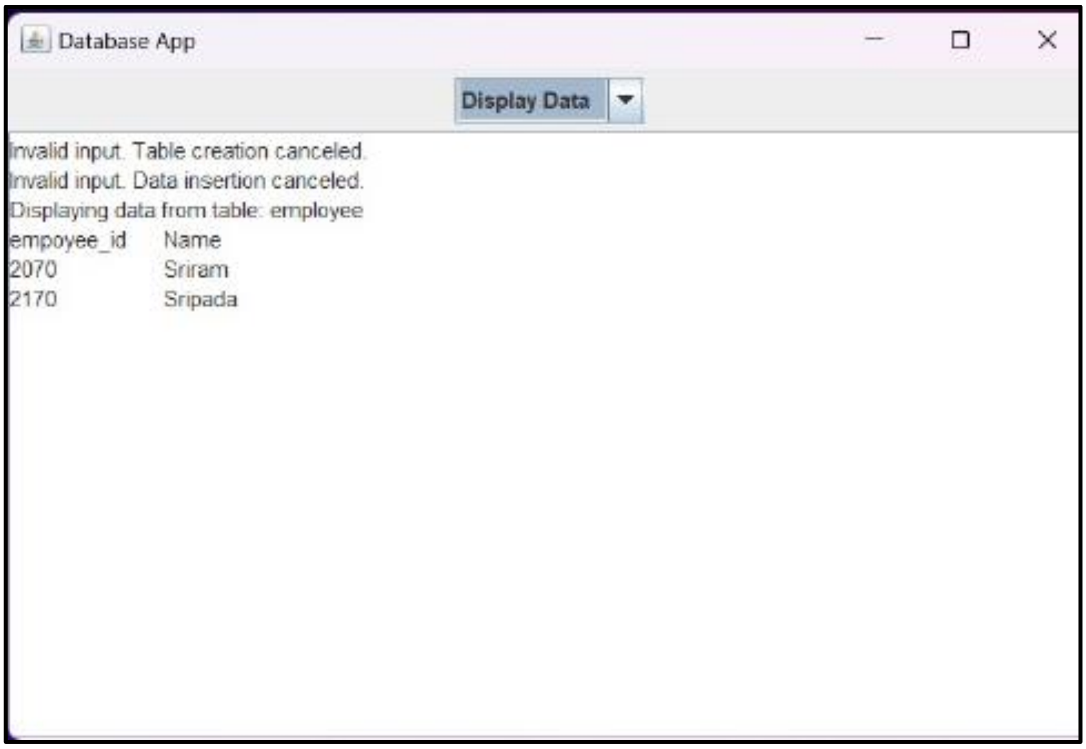


nvalid input. Table creation canceled.





```
nvalid input. Table creation canceled.
nvalid input. Data insertion canceled.
```



```
Invalid input. Table creation canceled.
Invalid input. Data insertion canceled.
Displaying data from table: employee
empoyee_id     Name
2070           Sriram
2170           Sripada
```

6.CONCLUSION

In conclusion, the implementation of a Graphical User Interface (GUI) for database commands marks a pivotal stride in enhancing user accessibility and efficiency within the realm of database management. The amalgamation of a user-friendly interface with powerful database functionality not only streamlines the execution of commands but also empowers users with varying levels of technical expertise to interact seamlessly with complex data structures. The GUI transcends the traditional barriers associated with command-line interfaces, offering an intuitive and visually engaging platform for database interaction. Its graphical representation of data schemas, coupled with point-and-click functionalities, diminishes the learning curve for novice users while affording seasoned professionals a more efficient means of managing intricate database operations. Moreover, the GUI facilitates a more interactive and dynamic experience, allowing users to visualize data relationships and dependencies with ease. As technology continues to advance, the GUI for database commands stands as a testament to the ongoing efforts to democratize access to robust data management tools, fostering a more inclusive and efficient approach to handling and manipulating databases in diverse domains.