CHAPTER 1

INTRODUCTION

1.1. PROBLEM DEFINITION

In today's fast-paced world, the demand for efficient car booking services has significantly increased. A comprehensive system is needed to streamline the process of booking cars, managing vehicle inventory, and handling administrative tasks. This system will enhance user experience by providing a seamless and user-friendly platform for both customers and administrators.

1.2. OBJECTIVES

The "Car Booking System" is designed with the following objectives in mind:

1. User-Friendly Interface: Develop an intuitive interface for users to browse and book cars.
2. Efficient Booking System: Implement a robust booking system that allows users to rent cars on an hourly or daily basis.
3. Admin Management: Provide an admin panel for managing cars and bookings.
4. Secure Authentication: Ensure secure user registration and authentication.
5. Data Persistence: Maintain a reliable database to store user, car, and booking data.

1.3. METHODOLOGY TO BE FOLLOWED

The development and implementation of the "Car Booking System" will follow a systematic and structured methodology, encompassing the following steps:

1. Requirement Analysis
2. System Design
3. Database Design
4. User Interface Development
5. Security Implementation
6. Functional Module Development
7. Testing and Quality Assurance
8. User Training and Documentation
9. Monitoring and Maintenance
10. Deployment

1.4. EXPECTED OUTCOMES

The implementation of the "Car Booking System" aims to achieve several positive outcomes, enhancing the overall booking experience for users and ensuring the system's reliability and security. The expected outcomes include:

1. Enhanced Security
2. User-Friendly Interface
3. User Login
4. Admin Management
5. Reliable Database Structure
6. Performance
7. Scalability

1.5. HARDWARE AND SOFTWARE REQUIREMENTS

HARDWARE:

2. Processor: Multi-core CPU (Intel Xeon or AMD EPYC)

3. Memory: Minimum 8 GB RAM (16 GB recommended for high load)

4. Storage: SSDs for fast data access, with sufficient capacity

5. Network: High-speed internet connection.

1.6. SOFTWARE:

1. **Operating System:**
   - Server: Linux(Ubuntu, CentOS) or Windows Server
   - Client: Windows, macOS or Linux
2. **Database Management System(DBMS)**
   - MySQL
3. **Backend Development**
   - Programming Languages: Java

CHAPTER 2

FUNDAMENTALS OF JAVA

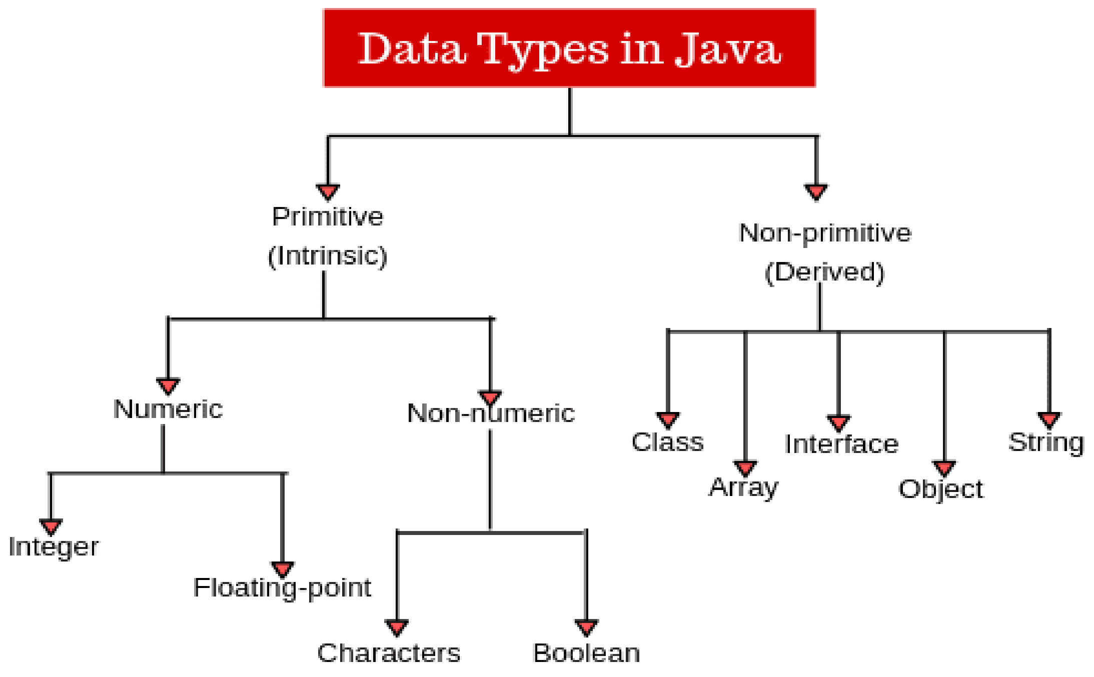
2.1. INTRODUCTION

Java is a popular programming language known for its portability, thanks to the Java Virtual Machine (JVM), which allows code to run on any platform. It uses an object-oriented programming model, promoting reusable and maintainable code. Java has extensive standard libraries and frameworks that ease development for a wide range of applications, from web to enterprise. It emphasizes security and performance, making it reliable for various domains. Its syntax is similar to C++, which helps programmers' transition smoothly.

2.2. ADVANTAGES IN JAVA

- SIMPLE
- SECURE
- PORTABLE
- ARCHITECTURAL NEUTRAL
- ROBUST
- COMPILED AND INTREPRETED
- OBJECT ORIENTED
- PLATFORM INDEPENDENT
- MULTI-THREADED
- HIGH IN PERFORMANCE
- DYNAMIC
- DISTRIBUTED

2.3. DATA TYPE



S

2.4. CONTROL FLOW

1. **Sequential Execution**: By default, Java statements executes codes from top to bottom.
2. **Conditional Statements**:
   - **if**: Executes a block of code if its condition evaluates to true.
   - **if-else**: Provides an alternative block of code if the condition is false.
   - **else if**: Chains multiple conditions together.
   - **switch**: Selects a block of code to execute from multiple options based on the value.
3. **Looping Statements**:
   - **for**: Repeats a block of code a specific number of times.
   - **while**: Repeats a block of code as long as its condition is true.
   - **do-while**: Similar to the while loop, but guarantees the block of code executes at least once.
4. **Branching Statements**:
   - **break**: Exits from the current loop or switch statement.
   - **continue**: Skips the current iteration of a loop and proceeds with the next iteration.
   - **return**: Exits from the current method and optionally returns a value

2.5. METHODS

1. Instance Methods
2. Static Methods
3. Abstract Methods
4. Final Methods
5. Synchronized Methods
6. Native Methods
7. Constructor Methods

2.6. OBJECT ORIENTED CONCEPTS

1. CLASS AND OBJECT
2. ENCAPSULATION
3. INHERITANCE
4. ABSTRACTION
5. POLYMORPHISM

2.7. EXCEPTION HANDLING

1. TRY-CATCH BLOCKS
2. MULTIPLE CATCH BLOCKS
3. FINALLY BLOCK
4. THROW
5. THROWS

2.8. FILE HANDLING

1. FILE CLASS

2. FILE INPUT/OUTPUT STREAMS

3. BYTE VS CHARACTER STREAMS

4. READING AND WRITING TEXT FILES

5. FILE AND DIRECTORY OPERATIONS

6. RANDOM ACCESS FILES

7. SERIALIZATION AND DESERIALIZATION

8. HANDLING EXCEPTIONS IN FILE I/O

2.9. PACKAGES AND IMPORT

1. PACKAGES DECLARTION

2. PACKAGE STRUCTURE

3. IMPORT STATEMENTS

4. NAMING CONFLICTS

5. DEFAULT PACKAGE

6. STATIC IMPORT

7. CLASSPATH AND PACKAGES

8. LIBRARY PACKAGES

2.10 INTERFACES

1. INTERFACE DECLARTION
2. DEFAULT METHODS
3. STATIC METHODS
4. MULTIPLE INHERITANCES WITH INTERFACES
5. IMPLEMENTING INTERFACES
6. EXTENDING INTERFACES

2.11. CONCURRENCY

1. THREADS AND MULTITHREADING
2. THREAD LIFECYCLE
3. RUNNABLE INTERFACE
4. SYNCHRONIAZTION
5. LOCKS AND MUTEX
6. DEADLOCKS
7. THREAD SAFETY
8. EXECUTORY FRAMEWORK

CHAPTER 3

JAVA COLLECTIONS GUIDE

3.1. OVERVIEW OF JAVA COLLECTIONS

1.INTRODUCTION TO JAVA COLLECTIONS

2.INTERFACES IN COLLECTION FRAMEWORK

- List
- Set
- Queue
- Map

3.CLASSES IN COLLECTION FRAMEWORK

- Array List
- Linked List
- Hash Set
- Tree Set
- Hash Map
- Tree Map
- Priority Queue

4.ITERATORS AND ITERABLES

5.COMPARATORS AND COMPARABLES

6.GENERICS IN COLLECTIONS

7.JAVA STREAMS IN API

3.2. IMPORTANTS IN JAVA DEVELOPMENT

1. JDK
2. IDE
3. JAVA SE
4. OOP PRINCIPLES
5. JAVA API DOCUMENTATION
6. JVM
7. DATABASE CONNECTIVITY (JDBC,Hibernate)

3.3. BENEFITS AND USECASES

BENEFITS:

1. PLATFORM INDEPENDENCE
2. OBJECT ORIENTED
3. RICH STANDARD LIBRARY
4. MULTITHREADING SUPPORT
5. DYNAMIC AND EXTENSIBLE
6. COMMUNITY SUPPORT
7. ROBUSTNESS

USECASES:

1. WEB DEVELOPMENT
2. ENTERPRISE APPLICATION (SCALABLE AND ROBUST)
3. MOBILE APPLICATIONS (JAVA OR KOTLIN)
4. CLOUD COMPUTING
5. DESKTOP APPLICATIONS
6. EMBEDDED SYSTEMS
7. GAME DEVELOPMENT (WITH Lib GDX AND ENGINES LIKE UNITY)

3.4. CORE COLLECTIONS INTERFACES

1. COLLECTION INTERFACE
2. LIST INTERFACES
3. SET INTERFACES
4. QUEUE INTERFACES
5. MAP INTERFACES
6. DEQUE INTERFACES
7. SORTEDSET INTERFACE
8. SORTEDMAP INTERFACE

3.5. COMMON COLLECTION IMPLEMENTATIONS

1. PRIORITY QUEUE
2. LINKED HASHSET
3. HASH TABLE
4. VECTOR
5. STACK
6. TREE MAP
7. ARRAY DEQUE
8. LINKED LIST

3.6. ITERATORS AND COLLECTIONS API

ITERATORS: An iterator in java provides methods to iterate the collection in allowing access to each element sequentially. The iterator interface in java collection framework are "hasNext()" to check if there are more elements, and "next()" to retrieve the next element in the iteration.

COLLECTIONS API:

1. COLLECTION
2. LIST
3. SET
4. QUEUE
5. MAP

3.7. CUSTOM COLLECTIONS AND GENERICS

CUSTOM COLLECTIONS:

- CUSTOM LIST IMPLEMENTATION
- CUSTOM SET APPLICATION
- CUSTOM MAP IMPLEMENTATION
- CUSTOM QUEUE IMPLEMENTATION

GENERICS:

1.CLASSES (class Box<T>)

2.INTERFACES (interface List<T>)

3.METHODS (public <T> T getElement(T[] array,int index))

4.BOUNDED TYPE PARAMETERS (<T extends Number>)

5.WILDCARDS (List<?>)

6.GENERIC CONSTRUCTORS (public <U> MyClass(U input))

CHAPTER 4

FUNDAMENTALS OF DBMS

1.1. INTRODUCTION

A Database Management System (DBMS) is a crucial tool for organizing and managing data effectively. It allows us to store, retrieve, and modify data in a structured way. With a DBMS, we can define how data is related and use query languages to interact with it easily. This system supports secure operations and ensures our data remains consistent and accessible. Overall, a DBMS plays a vital role in maintaining the integrity and usability of data across various applications.

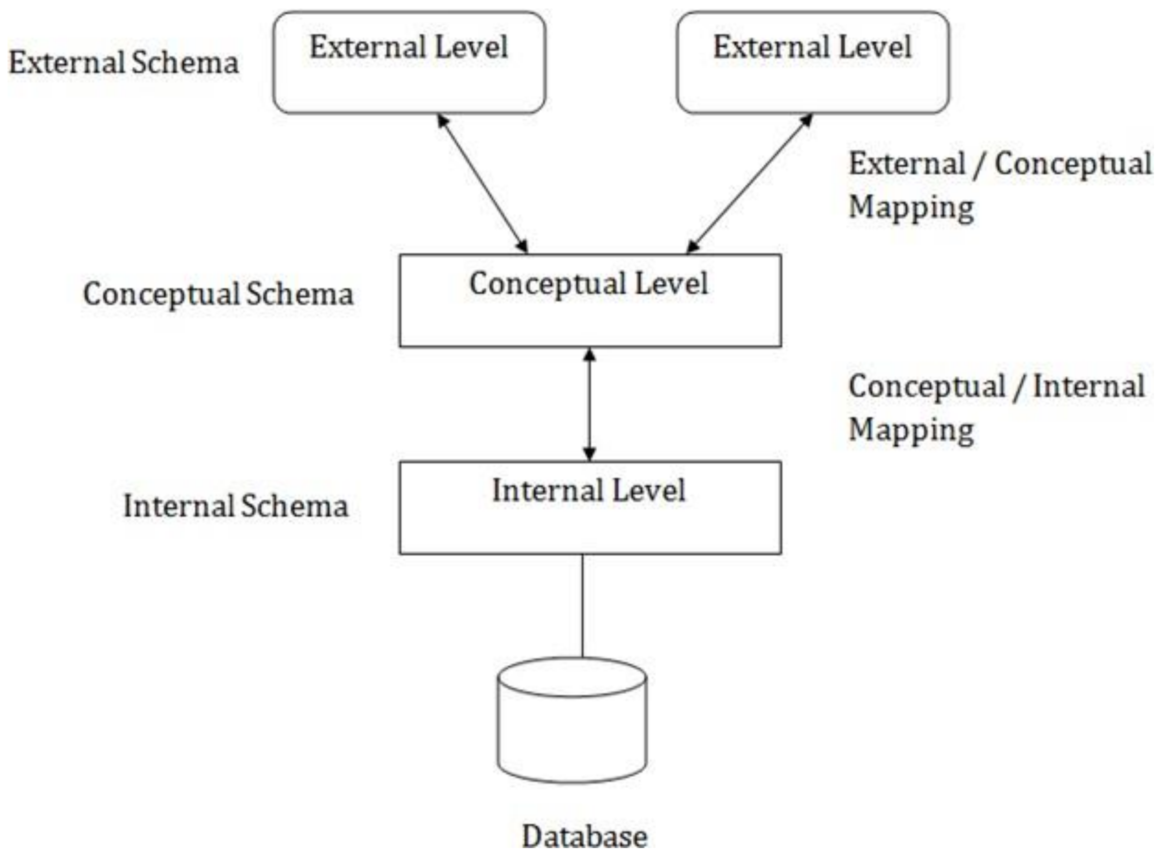
1.2. CHARACTERISTICS OF A DBMS

A Database Management System (DBMS) ensures data integrity and consistency through ACID properties (Atomicity, Consistency, Isolation, Durability). It supports data security with access controls and authentication mechanisms. The DBMS allows for efficient data retrieval and manipulation using SQL or other query languages. It provides backup and recovery options to protect against data loss. Additionally, the DBMS can handle large volumes of data and support concurrent user access without performance degradation.

1.3. DATA MODEL

In DBMS serves as a Blueprints for organizing and structuring data within a data within a database. They define the logical relationships and constraints between data elements. Common data models include the relational model, where the data is organized into tables to retrieve for display and input the commands for structured in let off the performance in hierarchical model, organized data in emphasizing models, relationships and attributes. Other models include in the network in structure with depends on the specific requirement

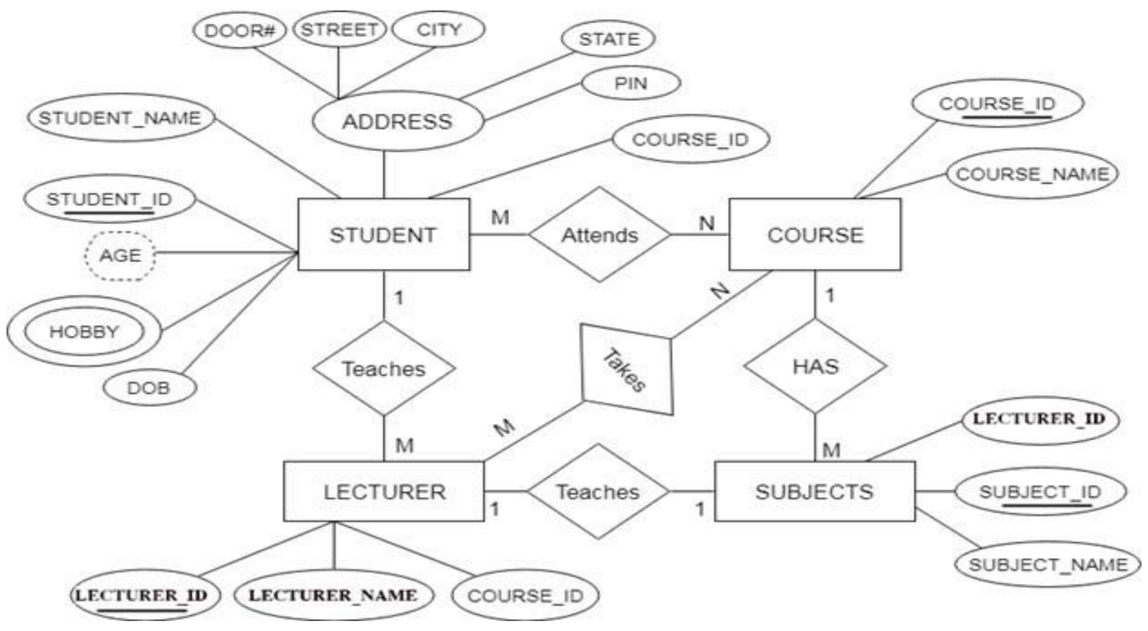
1.4. THREE SCHEMA ARCHITECTURE



1.5. DBMS COMPONENT MODULES

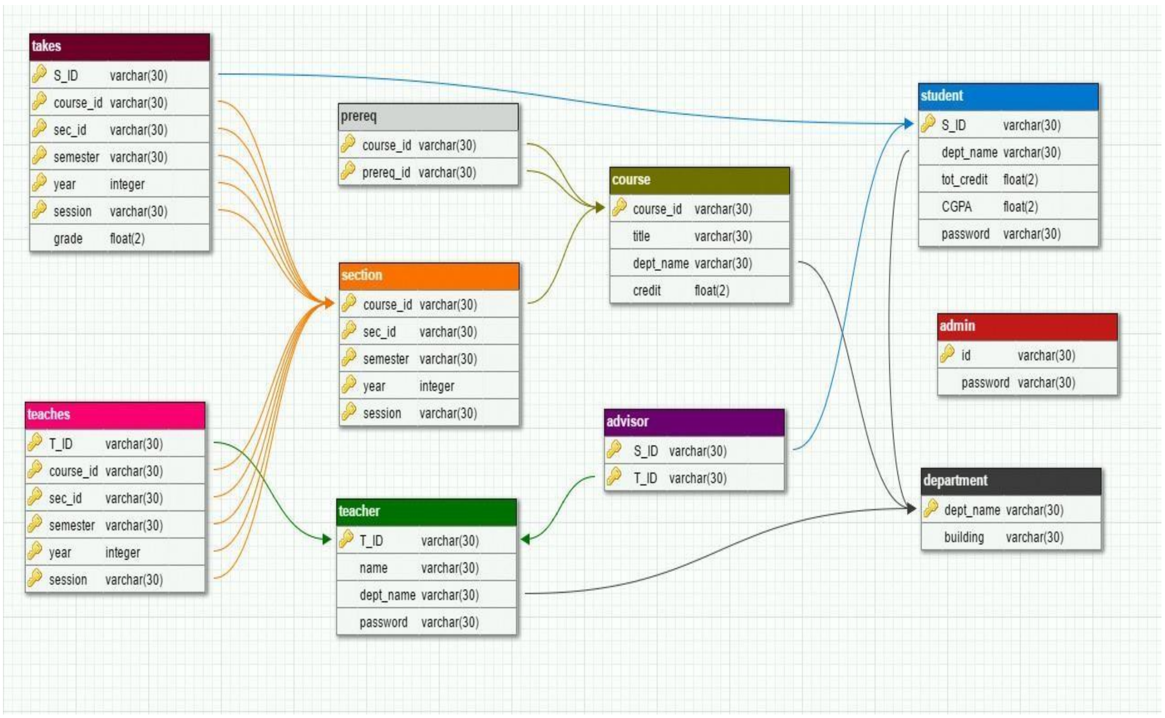
A DBMS is a several essential components and modules within the facilitates of a data management. The foundational component is the DDL, responsible for defining and managing, retrieve the data, insert the tables for displaying. The DML is having update, insert and deleting the tables. The DQL deserves the commands of select, selection, projection, join for the tables. The DCL deserves the commands like grant the permissions and revoke the permissions from user created. The TCL deserves the commands like commit, rollback and save-point for the tables.

1.6. ENTITY-RELATIONSHIP MODEL



UNIVERSITY DBMS ER MODEL

1.7. RELATIONAL SCHEMA



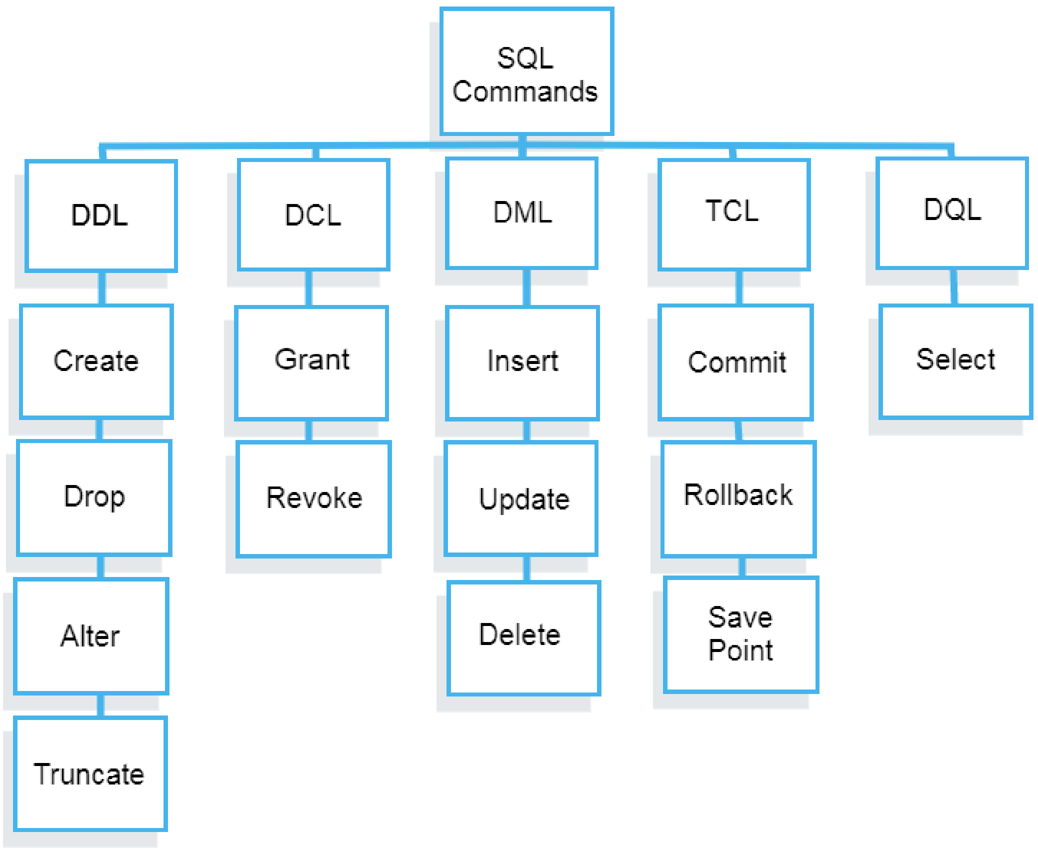
CHAPTER 5

FUNDAMENTALS OF SQL

5.1. INTRODUCTION

For this structured query language is a domain-specifications language designed for managing and manipulating relational databases, provides a set of commands for tasks and amplifications to the database for retrieve the data and underlying the database for bridge between users and Database. It consists of two main categories: DDL and DML which facilitates the manipulation the process of data within the database. SQL allows users to interact with databases seamlessly, enabling the creation the tables.

5.2. SQL COMMANDS



5.3. DATA DEFINITION LANGUAGE

DDL: It is a category of commands and managing the structure of a relational database. DDL statements enables the users to create, modify, and delete database objects such as tables, indexes and views.

1.CREATE: Used to create the new database objects like tables, indexes, views and more.

Syntax: create table table_name;

2.ALTER: Enables the modification of the structure of the table.

Syntax: alter table table_name modify column column_name;

3.DROP: Used to deletes the indexes or databases objects.

Syntax: drop table table_name;

4.TRUNCATE: Removes all records which are saved before while preserving the table structure. It is faster than the delete process.

Syntax: truncate table table_name;

5.RENAME: It renames the table name or database name to a new one.

Syntax: rename old_table_tablename new_table_tablename;

5.4. DATA MANIPULATION LANGUAGE

DML: It uses for comprises the commands that allow users to interact with and manipulate data stored in a relational database.

1.UPDATE: Modifies existing records in a table based on specified condition.

Syntax: update table_name set col1=val1 where condition;

2.INSERT: Add new records into a table.

Syntax: insert into table(col1,col2) values(value1,value2);

3.DELETE: Removes the records from a table on specified conditions only.

Syntax: delete from table where condition;

5.5. DATA CONTROL LANGUAGE

DCL: It is a category of SQL that deals with the permissions and access control within a relational DBMS.

1.GRANT: Granting the permissions to user like select, insert, update and delete.

Syntax: grant select table_name to user-name;

2.REVOKE: Revokes all the permissions back from user.

Syntax: revoke select table_name from user_name;

5.6. DATA QUERY LANGUAGE

DQL: It is a subset of SQL that specially focuses on the retrieval of data from a database. The DQL is the "select" Statement, which allows the users to specify the columns in the table.

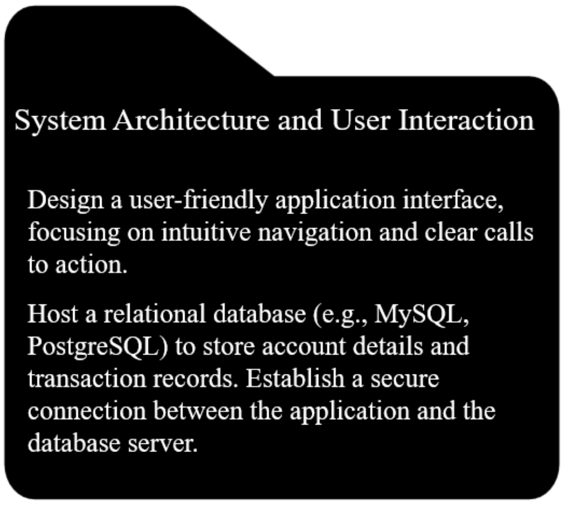
1.SELECT: Retrieves the table for displaying from the database.

Syntax: select * from table_name;

CHAPTER 6

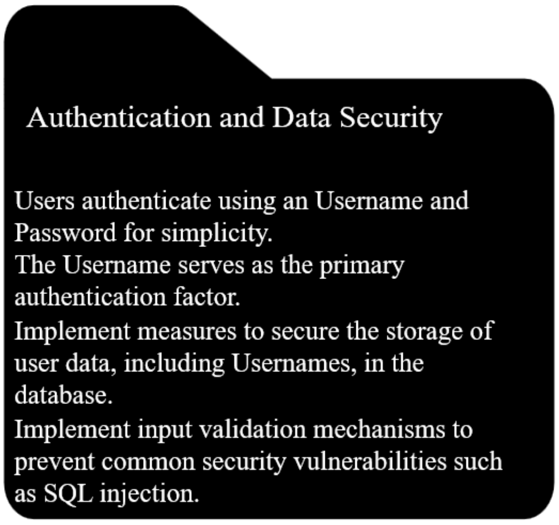
DESIGN AND ARCHITECTURE

6.1. DESIGN GOALS

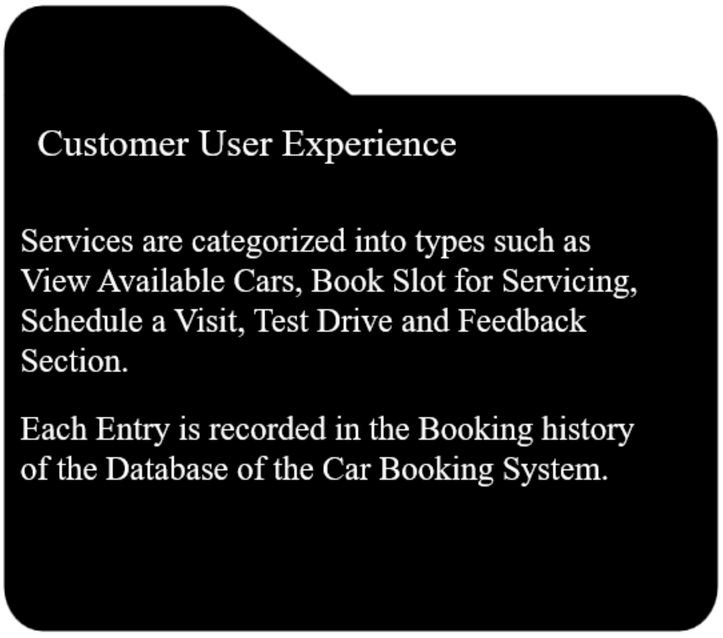

System Architecture and User Interaction

Design a user-friendly application interface, focusing on intuitive navigation and clear calls to action.

Host a relational database (e.g., MySQL, PostgreSQL) to store account details and transaction records. Establish a secure connection between the application and the database server.


Authentication and Data Security

Users authenticate using an Username and Password for simplicity.
The Username serves as the primary authentication factor.
Implement measures to secure the storage of user data, including Usernames, in the database.
Implement input validation mechanisms to prevent common security vulnerabilities such as SQL injection.


Customer User Experience

Services are categorized into types such as View Available Cars, Book Slot for Servicing, Schedule a Visit, Test Drive and Feedback Section.

Each Entry is recorded in the Booking history of the Database of the Car Booking System.

6.2. DATABASE STRUCTURES

The database structure of the Secure Banking System comprises several key tables designed to store essential information. These include:

- loginadmin: Stores information about registered admin users, including user ID, name and passwords.
- logincustomer: Stores information about registered customer users, including user ID, name and passwords.
- availablecars: Stores information about registered available cars, including car ID, make, model and availability status.
- serviceslot: Stores information about registered vehicle that wants to get serviced, vehicle number, day and slot
- visit: Stores information about registered users who want to visit, customer name, day, and timing slot.

CHAPTER 7

IMPLEMENTATION

7.1. CREATING THE DATABASE

```
create database carrentalsystem;
use carrentalsystem;
```

7.2. CONNECTING THE DATABASE TO THE APPLICATION

```
private void initializeDBConnection() {
      try {
         con = DriverManager.getConnection("jdbc:mysql://localhost/carrentalsystem", "root", "jerry");
      }
           catch (SQLException e) {
         JOptionPane.showMessageDialog(this, "Database Connection Error: " + e.getMessage());
      }
 }
```

7.3. CREATING THE LOGIN PAGES

1. Admin login:

```
package some;

import javax.swing.*;
import javax.swing.border.EmptyBorder;
import javax.swing.border.TitledBorder;
import java.awt.*;
import java.awt.event.*;
import java.sql.*;
import javax.swing.border.EtchedBorder;

public class Login extends JFrame {

   private static final long serialVersionUID = 1L;
   private JPanel contentPane;
   private JTextField txtusername;
   private JPasswordField txtpass;
   private Connection con;

   /**
    * Launch the application.
    */
```

```
public static void main(String[] args) {
    EventQueue.invokeLater(new Runnable() {
        public void run() {
            try {
                Login frame = new Login();
                frame.setVisible(true);
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    });
}

/**
 * Create the frame.
 */
public Login() {
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setBounds(100, 100, 802, 495);
    contentPane = new JPanel();
    contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));

    setContentPane(contentPane);
    contentPane.setLayout(null);

    JPanel panel = new JPanel();
    panel.setBorder(new TitledBorder(new EtchedBorder(EtchedBorder.LOWERED, new
Color(255, 255, 255), new Color(160, 160, 160)), "Admin Login", TitledBorder.LEADING,
TitledBorder.TOP, null, new Color(0, 0, 0)));
    panel.setBounds(64, 79, 654, 301);
    contentPane.add(panel);
    panel.setLayout(null);

    JLabel lblNewLabel_1 = new JLabel("USERNAME");
    lblNewLabel_1.setFont(new Font("Times New Roman", Font.PLAIN, 16));
    lblNewLabel_1.setBounds(155, 76, 119, 29);
    panel.add(lblNewLabel_1);

    JLabel lblNewLabel_1_1 = new JLabel("PASSWORD");
    lblNewLabel_1_1.setFont(new Font("Times New Roman", Font.PLAIN, 17));
    lblNewLabel_1_1.setBounds(155, 128, 119, 29);
    panel.add(lblNewLabel_1_1);

    JButton btnNewButton = new JButton("CLEAR");
    btnNewButton.setFont(new Font("Times New Roman", Font.PLAIN, 17));
    btnNewButton.setBounds(203, 208, 113, 35);
    panel.add(btnNewButton);

    txtusername = new JTextField();
    txtusername.setBounds(298, 77, 181, 29);
    panel.add(txtusername);
    txtusername.setColumns(10);

    txtpass = new JPasswordField();
    txtpass.setBounds(298, 128, 181, 26);
    panel.add(txtpass);

    JButton btnSubmit = new JButton("SUBMIT");
    btnSubmit.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        String username = txtusername.getText();
        String pass = new String(txtpass.getPassword());

        if (validateLogin(username, pass)) {
          Main m = new Main(); // Ensure Main class is correctly referenced
          Login.this.setVisible(false); // Hide current login frame
          m.setVisible(true); // Show main frame
        } else {
          JOptionPane.showMessageDialog(Login.this, "Username and Password Do not
Match");
        }
      }
    });
    btnSubmit.setFont(new Font("Times New Roman", Font.PLAIN, 17));
    btnSubmit.setBounds(366, 208, 113, 35);
    panel.add(btnSubmit);

    JLabel lblNewLabel = new JLabel("SV Car Booking System");
    lblNewLabel.setFont(new Font("Times New Roman", Font.BOLD, 25));
    lblNewLabel.setBounds(30, 24, 277, 45);
    contentPane.add(lblNewLabel);

    // Initialize database connection
    initializeDBConnection();
  }

  private void initializeDBConnection() {
    try {
            con = DriverManager.getConnection("jdbc:mysql://localhost/carrentalsystem", "root",
"jerry");
        } catch (SQLException e) {
            JOptionPane.showMessageDialog(this, "Database Connection Error: " +
e.getMessage());
        }
    }

    private boolean validateLogin(String username, String pass) {
        String query = "SELECT * FROM loginadmin WHERE Name=? AND Pass=?";
        try (PreparedStatement pst = con.prepareStatement(query)) {
            pst.setString(1, username);
            pst.setString(2, pass);
            ResultSet rs = pst.executeQuery();
            return rs.next(); // Returns true if there is at least one match
        } catch (SQLException e) {
            JOptionPane.showMessageDialog(this, "SQL Error: " + e.getMessage());
        }
        return false;
    }
}
```

2. Customer login:

```
package some;

import javax.swing.*;
import javax.swing.border.EmptyBorder;
import javax.swing.border.TitledBorder;
import java.awt.*;
import java.awt.event.*;
import java.sql.*;

public class CusLogin extends JFrame {

    private static final long serialVersionUID = 1L;
    private JPanel contentPane;
    private JTextField txtusername;
    private JPasswordField txtpass;
    private Connection con;

    /**
     * Launch the application.
     */
    public static void main(String[] args) {
        EventQueue.invokeLater(new Runnable() {
            public void run() {
                try {
                    CusLogin frame = new CusLogin();
                    frame.setVisible(true);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
    }

    /**
     * Create the frame.
     */
    public CusLogin() {
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setBounds(100, 100, 802, 495);
        contentPane = new JPanel();
        contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));

        setContentPane(contentPane);
        contentPane.setLayout(null);

        JPanel panel = new JPanel();
        panel.setBorder(new TitledBorder(null, "Customer Login", TitledBorder.LEADING,
TitledBorder.TOP, null, null));
        panel.setBounds(64, 79, 654, 301);
        contentPane.add(panel);
        panel.setLayout(null);

        JLabel lblNewLabel_1 = new JLabel("REGISTERED NAME");
        lblNewLabel_1.setFont(new Font("Times New Roman", Font.PLAIN, 16));
        lblNewLabel_1.setBounds(123, 89, 151, 29);
        panel.add(lblNewLabel_1);

        JLabel lblNewLabel_1_1 = new JLabel("PASSWORD");
        lblNewLabel_1_1.setFont(new Font("Times New Roman", Font.PLAIN, 17));
        lblNewLabel_1_1.setBounds(123, 141, 151, 29);
        panel.add(lblNewLabel_1_1);

        JButton btnNewButton = new JButton("CLEAR");
        btnNewButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
```

```
            clearFields();
        }
    });
    btnNewButton.setFont(new Font("Times New Roman", Font.PLAIN, 17));
    btnNewButton.setBounds(85, 208, 113, 35);
    panel.add(btnNewButton);

    txtusername = new JTextField();
    txtusername.setBounds(298, 92, 181, 26);
    panel.add(txtusername);
    txtusername.setColumns(10);

    txtpass = new JPasswordField();
    txtpass.setBounds(298, 145, 181, 26);
    panel.add(txtpass);

    JButton btnSubmit = new JButton("LOG IN");
    btnSubmit.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            String username = txtusername.getText();
            String pass = new String(txtpass.getPassword());

            if (validateCusLogin(username, pass)) {
                MainCus mainCusFrame = new MainCus(); // Replace with MainCus frame
                CusLogin.this.setVisible(false); // Hide current login frame
                mainCusFrame.setVisible(true); // Show MainCus frame
            } else {
                JOptionPane.showMessageDialog(CusLogin.this, "Username, Password, or Vehicle
Number Do not Match");
            }
        }
    });
    btnSubmit.setFont(new Font("Times New Roman", Font.PLAIN, 17));
    btnSubmit.setBounds(412, 208, 113, 35);
    panel.add(btnSubmit);

    JButton btnSignUp = new JButton("SIGN UP");
    btnSignUp.addMouseListener(new MouseAdapter() {
         @Override
         public void mouseClicked(MouseEvent e) {
                  CusSignUp signUpFrame = new CusSignUp();
            signUpFrame.setVisible(true);
         }
    });
```

```
        btnSignUp.setFont(new Font("Times New Roman", Font.PLAIN, 17));
        btnSignUp.setBounds(252, 208, 113, 35);
        panel.add(btnSignUp);

        JLabel lblLogin = new JLabel("LOGIN");
        lblLogin.setFont(new Font("Times New Roman", Font.BOLD, 25));
        lblLogin.setBounds(278, 21, 113, 45);
        panel.add(lblLogin);

        JLabel lblNewLabel = new JLabel("SV Car Booking System");
        lblNewLabel.setFont(new Font("Times New Roman", Font.BOLD, 25));
        lblNewLabel.setBounds(30, 24, 277, 45);
        contentPane.add(lblNewLabel);
        // Initialize database connection
        initializeDBConnection();
    }

    private void initializeDBConnection() {
        try {
            con = DriverManager.getConnection("jdbc:mysql://localhost/carrentalsystem", "root",
"jerry");
        } catch (SQLException e) {
            JOptionPane.showMessageDialog(this, "Database Connection Error: " +
e.getMessage());
        }
    }
    private boolean validateCusLogin(String username, String pass) {
        String query = "SELECT * FROM logincustomer WHERE Name=? AND Pass=?";
        try (PreparedStatement pst = con.prepareStatement(query)) {
            pst.setString(1, username);
            pst.setString(2, pass);
            ResultSet rs = pst.executeQuery();
            return rs.next(); // Returns true if there is at least one match
        } catch (SQLException e) {
            JOptionPane.showMessageDialog(this, "SQL Error: " + e.getMessage());
        }
        return false;
    }
    private void clearFields() {
             txtusername.setText("");
        txtpass.setText("");
    }
}
```

7.4. CREATING THE MAIN WINDOW

1. Admin Page:

```
package some;

import java.awt.EventQueue;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.border.EmptyBorder;
import javax.swing.JButton;
import java.awt.Font;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import javax.swing.border.TitledBorder;
import javax.swing.JLabel;
import javax.swing.border.LineBorder;
import java.awt.Color;

public class Main extends JFrame {

    private static final long serialVersionUID = 1L;
    private JPanel contentPane;

    /**
     * Launch the application.
     */
    public static void main(String[] args) {
        EventQueue.invokeLater(new Runnable() {
            public void run() {
                try {
                    Main frame = new Main();
                    frame.setVisible(true);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
    }

    /**
     * Create the frame.
     */
    public Main() {
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setBounds(100, 100, 691, 402);
        contentPane = new JPanel();
        contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));

        setContentPane(contentPane);
        contentPane.setLayout(null);

        JButton btnExitt = new JButton("EXIT");
        btnExitt.setFont(new Font("Times New Roman", Font.PLAIN, 18));
        btnExitt.setBounds(356, 248, 233, 37);
        contentPane.add(btnExitt);

        JPanel panel = new JPanel();
        panel.setBorder(new TitledBorder(null, "Admin Desk", TitledBorder.LEADING,
TitledBorder.TOP, null, null));
        panel.setBounds(56, 47, 568, 274);
        contentPane.add(panel);
        panel.setLayout(null);

        JLabel lblNewLabel = new JLabel("Welcome !!");
        lblNewLabel.setFont(new Font("Times New Roman", Font.BOLD, 30));
        lblNewLabel.setBounds(46, 60, 189, 60);
        panel.add(lblNewLabel);

                JLabel lblAdmin = new JLabel("Admin");
                lblAdmin.setFont(new Font("Times New Roman", Font.BOLD, 30));
                lblAdmin.setBounds(46, 109, 189, 60);
                panel.add(lblAdmin);

                JPanel panel_1 = new JPanel();
                panel_1.setBorder(new LineBorder(new Color(0, 0, 0)));
                panel_1.setBounds(287, 28, 262, 222);
                panel.add(panel_1);
                    panel_1.setLayout(null);

                    JButton btnViewAvailableCars = new JButton("UPDATE CARS");
                    btnViewAvailableCars.setBounds(19, 10, 233, 44);
                    panel_1.add(btnViewAvailableCars);
                    btnViewAvailableCars.addActionListener(new ActionListener() {
                      public void actionPerformed(ActionEvent e) {
                        VAC v = new VAC();
                        v.setVisible(true);
                      }
                    });
                    btnViewAvailableCars.setFont(new Font("Times New Roman", Font.PLAIN,
18));

                        JButton newlog = new JButton("LOGOUT");
                        newlog.setBounds(19, 96, 233, 37);
                        panel_1.add(newlog);
                        newlog.addActionListener(new ActionListener() {
                          public void actionPerformed(ActionEvent e) {
                            // Open the twologin frame when LOGIN button is clicked
                            twologin loginFrame = new twologin();
                            loginFrame.setVisible(true);
                          }
                        });
                        newlog.setFont(new Font("Times New Roman", Font.PLAIN, 18));
      }
}
```

2. Customer Page:

```
package some;

import java.awt.EventQueue;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.border.EmptyBorder;
import javax.swing.JButton;
import java.awt.Font;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
public class MainCus extends JFrame {
   private static final long serialVersionUID = 1L;
   private JPanel contentPane;
   /**
    * Launch the application.
    */
   public static void main(String[] args) {
      EventQueue.invokeLater(new Runnable() {
         public void run() {
            try {
               MainCus frame = new MainCus();
               frame.setVisible(true);
            } catch (Exception e) {
               e.printStackTrace();
            }
         }
      });
   }
   public MainCus() {
      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      setBounds(100, 100, 440, 565);
      contentPane = new JPanel();
      contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
      setContentPane(contentPane);
      contentPane.setLayout(null);
      JButton newlog = new JButton("LOGOUT");
      newlog.addActionListener(new ActionListener() {
         public void actionPerformed(ActionEvent e) {
            // Open the twologin frame when LOGIN button is clicked
            twologin loginFrame = new twologin();
            loginFrame.setVisible(true);
         }
      });
```

```
        newlog.setFont(new Font("Times New Roman", Font.PLAIN, 18));
        newlog.setBounds(99, 57, 233, 37);
        contentPane.add(newlog);

        JButton btnViewAvailableCars = new JButton("View Available Cars");
        btnViewAvailableCars.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                VAC v = new VAC();
                v.setVisible(true);
            }
        });
        btnViewAvailableCars.setFont(new Font("Times New Roman", Font.PLAIN, 18));
        btnViewAvailableCars.setBounds(99, 140, 233, 37);
        contentPane.add(btnViewAvailableCars);

        JButton btnBookSlotFor = new JButton("Book Slot for Servicing");
        btnBookSlotFor.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                SlotBook slotBookFrame = new SlotBook();
                slotBookFrame.setVisible(true);
            }
        });
        btnBookSlotFor.setFont(new Font("Times New Roman", Font.PLAIN, 18));
        btnBookSlotFor.setBounds(99, 227, 233, 37);
        contentPane.add(btnBookSlotFor);

        JButton btnScheduleAVisit = new JButton("Schedule a Visit");
        btnScheduleAVisit.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                Visit visitFrame = new Visit();
                visitFrame.setVisible(true);
            }
        });
        btnScheduleAVisit.setFont(new Font("Times New Roman", Font.PLAIN, 18));
        btnScheduleAVisit.setBounds(99, 322, 233, 37);
        contentPane.add(btnScheduleAVisit);

        JButton btnExitt = new JButton("EXIT");
        btnExitt.setFont(new Font("Times New Roman", Font.PLAIN, 18));
        btnExitt.setBounds(99, 408, 233, 37);
        contentPane.add(btnExitt);
    }
}
```

7.5. MAIN MENU PAGE

1. ADMIN

```
package some;

import java.awt.EventQueue;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.border.EmptyBorder;
import javax.swing.JButton;
import java.awt.Font;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import javax.swing.border.TitledBorder;
import javax.swing.JLabel;
import javax.swing.border.LineBorder;
import java.awt.Color;

public class Main extends JFrame {

    private static final long serialVersionUID = 1L;
    private JPanel contentPane;

    /**
     * Launch the application.
     */
    public static void main(String[] args) {
        EventQueue.invokeLater(new Runnable() {
            public void run() {
                try {
                    Main frame = new Main();
                    frame.setVisible(true);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
    }

    /**
     * Create the frame.
     */
    public Main() {
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

```
	setBounds(100, 100, 691, 402);
	contentPane = new JPanel();
	contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));

	setContentPane(contentPane);
	contentPane.setLayout(null);

	JButton btnExitt = new JButton("EXIT");
	btnExitt.setFont(new Font("Times New Roman", Font.PLAIN, 18));
	btnExitt.setBounds(356, 248, 233, 37);
	contentPane.add(btnExitt);

	JPanel panel = new JPanel();
	panel.setBorder(new TitledBorder(null, "Admin Desk", TitledBorder.LEADING,
TitledBorder.TOP, null, null));
	panel.setBounds(56, 47, 568, 274);
	contentPane.add(panel);
	panel.setLayout(null);

	JLabel lblNewLabel = new JLabel("Welcome !!");
	lblNewLabel.setFont(new Font("Times New Roman", Font.BOLD, 30));
	lblNewLabel.setBounds(46, 60, 189, 60);
	panel.add(lblNewLabel);

			JLabel lblAdmin = new JLabel("Admin");
			lblAdmin.setFont(new Font("Times New Roman", Font.BOLD, 30));
			lblAdmin.setBounds(46, 109, 189, 60);
			panel.add(lblAdmin);

			JPanel panel_1 = new JPanel();
			panel_1.setBorder(new LineBorder(new Color(0, 0, 0)));
			panel_1.setBounds(287, 28, 262, 222);
			panel.add(panel_1);
				panel_1.setLayout(null);

				JButton btnViewAvailableCars = new JButton("UPDATE CARS");
				btnViewAvailableCars.setBounds(19, 10, 233, 44);
				panel_1.add(btnViewAvailableCars);
				btnViewAvailableCars.addActionListener(new ActionListener() {
				   public void actionPerformed(ActionEvent e) {
				      VAC v = new VAC();
				      v.setVisible(true);
				   }
				});
				btnViewAvailableCars.setFont(new Font("Times New Roman", Font.PLAIN,
18));
```

```
                        JButton newlog = new JButton("LOGOUT");
                        newlog.setBounds(19, 96, 233, 37);
                        panel_1.add(newlog);
                        newlog.addActionListener(new ActionListener() {
                            public void actionPerformed(ActionEvent e) {
                                // Open the twologin frame when LOGIN button is clicked
                                twologin loginFrame = new twologin();
                                loginFrame.setVisible(true);
                            }
                        });
                        newlog.setFont(new Font("Times New Roman", Font.PLAIN, 18));
    }
}
```

2. CUSTOMER

```
package some;

import java.awt.EventQueue;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.border.EmptyBorder;
import javax.swing.JButton;
import java.awt.Font;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;

public class MainCus extends JFrame {

    private static final long serialVersionUID = 1L;
    private JPanel contentPane;

    /**
     * Launch the application.
     */
    public static void main(String[] args) {
        EventQueue.invokeLater(new Runnable() {
            public void run() {
                try {
                    MainCus frame = new MainCus();
                    frame.setVisible(true);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
    }

    /**
     * Create the frame.
     */
    public MainCus() {
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setBounds(100, 100, 440, 565);
        contentPane = new JPanel();
        contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));

        setContentPane(contentPane);
        contentPane.setLayout(null);
        JButton newlog = new JButton("LOGOUT");
        newlog.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                // Open the twologin frame when LOGIN button is clicked
                twologin loginFrame = new twologin();
                loginFrame.setVisible(true);
            }
        });
        newlog.setFont(new Font("Times New Roman", Font.PLAIN, 18));
        newlog.setBounds(99, 57, 233, 37);
        contentPane.add(newlog);

        JButton btnViewAvailableCars = new JButton("View Available Cars");
        btnViewAvailableCars.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                VAC v = new VAC();
                v.setVisible(true);
            }
        });
        btnViewAvailableCars.setFont(new Font("Times New Roman", Font.PLAIN, 18));
        btnViewAvailableCars.setBounds(99, 140, 233, 37);
        contentPane.add(btnViewAvailableCars);

        JButton btnBookSlotFor = new JButton("Book Slot for Servicing");
        btnBookSlotFor.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                SlotBook slotBookFrame = new SlotBook();
                slotBookFrame.setVisible(true);
            }
        });
        btnBookSlotFor.setFont(new Font("Times New Roman", Font.PLAIN, 18));
        btnBookSlotFor.setBounds(99, 227, 233, 37);
        contentPane.add(btnBookSlotFor);

        JButton btnScheduleAVisit = new JButton("Schedule a Visit");
        btnScheduleAVisit.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                Visit visitFrame = new Visit();
                visitFrame.setVisible(true);
            }
        });
        btnScheduleAVisit.setFont(new Font("Times New Roman", Font.PLAIN, 18));
        btnScheduleAVisit.setBounds(99, 322, 233, 37);
        contentPane.add(btnScheduleAVisit);

        JButton btnExitt = new JButton("EXIT");
        btnExitt.setFont(new Font("Times New Roman", Font.PLAIN, 18));
        btnExitt.setBounds(99, 408, 233, 37);
        contentPane.add(btnExitt);
    }
}
```

7.6. Viewing available cars:

```
package some;

import javax.swing.*;
import javax.swing.border.*;
import javax.swing.table.*;
import java.awt.*;
import java.awt.event.*;
import java.sql.*;
import java.util.logging.Level;
import java.util.logging.Logger;

public class VAC extends JFrame {

    private static final long serialVersionUID = 1L;
    private JPanel contentPane;
    private JTable table;
    private JScrollPane scrollPane;
    private JLabel lblMake;
    private JTextField txtMake;
    private JLabel lblModel;
    private JLabel lblAvailable;
    private JTextField txtModel;
    private JButton btnReset;
    private JButton btnCheck;
    private JButton btnAdd;
    private JButton btnDelete;
    private JLabel lblID;
    private JTextField txtID;
    private JComboBox<String> comAvl;
    private JComboBox<String> comboAvail;

    Connection con;
    PreparedStatement pst;

    /**
     * Launch the application.
     */
    public static void main(String[] args) {
        EventQueue.invokeLater(new Runnable() {
            public void run() {
                try {
                    VAC frame = new VAC();
                    frame.setVisible(true);
                } catch (Exception e) {
```

```
                e.printStackTrace();
            }
        }
    });
}

/**
 * Create the frame.
 */
public VAC() {
    initializeDBConnection();
    initializeUI();
    autoID();
    fetchData();
}

private void initializeDBConnection() {
    try {
        con = DriverManager.getConnection("jdbc:mysql://localhost/carrentalsystem", "root",
"jerry");
    } catch (SQLException e) {
        Logger.getLogger(VAC.class.getName()).log(Level.SEVERE, null, e);
    }
}

private void autoID() {
    try {
        Statement s = con.createStatement();
        ResultSet rs = s.executeQuery("select Max(carID) from availablecars");
        if (rs.next()) {
            String maxID = rs.getString("Max(carID)");
            if (maxID == null) {
                txtID.setText("C001");
            } else {
                long id = Long.parseLong(maxID.substring(2)) + 1;
                txtID.setText("C" + String.format("%03d", id));
            }
        }
    } catch (SQLException e) {
        Logger.getLogger(VAC.class.getName()).log(Level.SEVERE, null, e);
    }
}

private void initializeUI() {
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setBounds(100, 100, 901, 563);
    contentPane = new JPanel();
    contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
    setContentPane(contentPane);
    contentPane.setLayout(null);

    JPanel panel = new JPanel();
    panel.setBorder(new TitledBorder(null, "View Available Cars", TitledBorder.LEADING,
TitledBorder.TOP, null, null));
    panel.setBounds(39, 38, 409, 449);
    contentPane.add(panel);
    panel.setLayout(null);

    lblID = new JLabel("ID");
    lblID.setFont(new Font("Times New Roman", Font.PLAIN, 20));
    lblID.setBounds(55, 49, 72, 30);
    panel.add(lblID);

    txtID = new JTextField();
    txtID.setColumns(10);
    txtID.setBounds(167, 49, 179, 30);
    panel.add(txtID);

    lblMake = new JLabel("Make");
    lblMake.setFont(new Font("Times New Roman", Font.PLAIN, 20));
    lblMake.setBounds(55, 113, 72, 30);
    panel.add(lblMake);

    txtMake = new JTextField();
    txtMake.setBounds(167, 113, 179, 30);
    panel.add(txtMake);
    txtMake.setColumns(10);

    lblModel = new JLabel("Model");
    lblModel.setFont(new Font("Times New Roman", Font.PLAIN, 20));
    lblModel.setBounds(55, 181, 72, 30);
    panel.add(lblModel);

    txtModel = new JTextField();
    txtModel.setColumns(10);
    txtModel.setBounds(167, 181, 179, 30);
    panel.add(txtModel);

    lblAvailable = new JLabel("Available");
    lblAvailable.setFont(new Font("Times New Roman", Font.PLAIN, 20));
    lblAvailable.setBounds(55, 247, 87, 30);
    panel.add(lblAvailable);
```

```
comAvl = new JComboBox<>();
comAvl.setModel(new DefaultComboBoxModel(new String[] {"---", "Yes", "No"}));
comAvl.setBounds(167, 247, 179, 28);
panel.add(comAvl);

btnReset = new JButton("Reset");
btnReset.addActionListener(new ActionListener() {
   public void actionPerformed(ActionEvent e) {
      resetFields();
   }
});
btnReset.setFont(new Font("Times New Roman", Font.BOLD, 20));
btnReset.setBounds(97, 329, 92, 44);
panel.add(btnReset);

btnCheck = new JButton("Check");
btnCheck.setFont(new Font("Times New Roman", Font.BOLD, 20));
btnCheck.setBounds(238, 329, 92, 44);
btnCheck.addActionListener(new ActionListener() {
   public void actionPerformed(ActionEvent e) {
      checkAvailability();
   }
});
panel.add(btnCheck);

btnAdd = new JButton("Add");
btnAdd.addActionListener(new ActionListener() {
   public void actionPerformed(ActionEvent e) {
      addCar();
   }
});
btnAdd.setFont(new Font("Times New Roman", Font.BOLD, 20));
btnAdd.setBounds(97, 383, 92, 44);
panel.add(btnAdd);

btnDelete = new JButton("Delete");
btnDelete.addActionListener(new ActionListener() {
   public void actionPerformed(ActionEvent e) {
      deleteCar();
   }
});
btnDelete.setFont(new Font("Times New Roman", Font.BOLD, 20));
btnDelete.setBounds(238, 383, 92, 44);
panel.add(btnDelete);
```

```
    scrollPane = new JScrollPane();
    scrollPane.setBounds(480, 49, 370, 282);
    contentPane.add(scrollPane);

    table = new JTable();
    scrollPane.setViewportView(table);
    table.setBorder(new LineBorder(new Color(0, 0, 0)));
    table.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
    table.setModel(new DefaultTableModel(
        new Object[][] {
        },
        new String[] {
            "ID", "Make", "Model", "Available"
        }
    ) {
        Class[] columnTypes = new Class[] {
            String.class, String.class, String.class, String.class
        };
        public Class getColumnClass(int columnIndex) {
            return columnTypes[columnIndex];
        }
    });

    table.addMouseListener(new MouseAdapter() {
        public void mouseClicked(MouseEvent e) {
            int selectedRow = table.getSelectedRow();
            if (selectedRow != -1) {
                txtID.setText(table.getValueAt(selectedRow, 0).toString());
                txtMake.setText(table.getValueAt(selectedRow, 1).toString());
                txtModel.setText(table.getValueAt(selectedRow, 2).toString());
                comAvl.setSelectedItem(table.getValueAt(selectedRow, 3).toString());
            }
        }
    });

    JPanel panel_1 = new JPanel();
    panel_1.setBorder(new TitledBorder(null, "Status", TitledBorder.LEADING,
TitledBorder.TOP, null, null));
    panel_1.setBounds(477, 347, 373, 135);
    contentPane.add(panel_1);
    panel_1.setLayout(null);

    JLabel lblAvailability = new JLabel("Availability:");
    lblAvailability.setFont(new Font("Times New Roman", Font.PLAIN, 20));
    lblAvailability.setBounds(43, 42, 120, 44);
    panel_1.add(lblAvailability);
```

```
        comboAvail = new JComboBox<>();
        comboAvail.setModel(new DefaultComboBoxModel<>(new String[] {"--", "Yes", "No"}));
        comboAvail.setBounds(173, 42, 148, 35);
        panel_1.add(comboAvail);
    }

    private void addCar() {
        String cid = txtID.getText();
        String make = txtMake.getText();
        String model = txtModel.getText();
        String available = comAvl.getSelectedItem().toString();

        try {
            pst = con.prepareStatement("insert into availablecars(carID, make, model, avail)
values(?, ?, ?, ?)");
            pst.setString(1, cid);
            pst.setString(2, make);
            pst.setString(3, model);
            pst.setString(4, available);
            pst.executeUpdate();
            JOptionPane.showMessageDialog(this, "Car added...");
            resetFields();
            autoID();
            fetchData();
        } catch (SQLException e) {
            Logger.getLogger(VAC.class.getName()).log(Level.SEVERE, null, e);
        }
    }

    private void deleteCar() {
        String cid = txtID.getText();
        String make = txtMake.getText();
        String model = txtModel.getText();

        try {
            pst = con.prepareStatement("delete from availablecars where carID = ? and make = ?
and model = ?");
            pst.setString(1, cid);
            pst.setString(2, make);
            pst.setString(3, model);
            int rowsAffected = pst.executeUpdate();
            if (rowsAffected > 0) {
                JOptionPane.showMessageDialog(this, "Car deleted...");
                resetFields();
                autoID();
            fetchData();
         } else {
            JOptionPane.showMessageDialog(this, "No car found with the given details...");
         }
      } catch (SQLException e) {
         Logger.getLogger(VAC.class.getName()).log(Level.SEVERE, null, e);
      }
   }

   private void fetchData() {
      try {
         pst = con.prepareStatement("select * from availablecars");
         ResultSet rs = pst.executeQuery();
         DefaultTableModel model = (DefaultTableModel) table.getModel();
         model.setRowCount(0);
         while (rs.next()) {
            String id = rs.getString("carID");
            String make = rs.getString("make");
            String modelStr = rs.getString("model");
            String avail = rs.getString("avail");
            model.addRow(new Object[]{id, make, modelStr, avail});
         }
      } catch (SQLException e) {
         Logger.getLogger(VAC.class.getName()).log(Level.SEVERE, null, e);
      }
   }

   private void resetFields() {
      txtID.setText("");
      txtMake.setText("");
      txtModel.setText("");
      comAvl.setSelectedIndex(-1);
      txtMake.requestFocus();
      autoID();
   }

   private void checkAvailability() {
      String make = txtMake.getText();
      String model = txtModel.getText();
      if (make.isEmpty() || model.isEmpty()) {
         JOptionPane.showMessageDialog(this, "Please enter both make and model to check
availability.");
         return;
      }

      try {
            pst = con.prepareStatement("select avail from availablecars where make = ? and
model = ?");
            pst.setString(1, make);
            pst.setString(2, model);
            ResultSet rs = pst.executeQuery();
            if (rs.next()) {
                String availability = rs.getString("avail");
                comboAvail.setSelectedItem(availability);
                JOptionPane.showMessageDialog(this, "Availability: " + availability);
            } else {
                JOptionPane.showMessageDialog(this, "No car found with the given make and
model.");
            }
        } catch (SQLException e) {
            Logger.getLogger(VAC.class.getName()).log(Level.SEVERE, null, e);
        }
    }
}
```

7.7. PROCESSING QUERIES

These are done manually by configuring into the mySQL Workbench.

7.8. CODING THE CORE FUNCTIONALITIES

Key functionalities, including user registration, login, viewing available vehicles, Scheduling visits and booking slots for car servicing, were coded to ensure security, reliability, and a seamless user experience.

7.9. HANDLING USER INPUTS

Robust mechanisms were implemented to handle user inputs, including validation and sanitation procedures. This enhances data integrity and prevents potential security vulnerabilities.

CHAPTER 8

TESTING

1. Understanding Unit Testing

Definition:

- **Unit Testing:** Involves testing individual units of code, such as methods or classes, in isolation to ensure they perform as expected.

Purpose:

- **Verify Functionality:** Ensure that each unit performs its intended function correctly.
- **Catch Bugs Early:** Identify and fix issues early in development.
- **Refactoring Confidence:** Refactor code with assurance that existing functionality remains intact.

2. Identify Testable Units

- **Classes:** Focus on classes with significant logic or functionality (e.g., car booking, user management).
- **Methods:** Test methods performing critical operations or calculations (e.g., availability checks, booking confirmation).

3. Create Test Cases

- **Example Test Cases:**
    - **Booking Methods:** Test methods that handle booking logic.
        - **Scenario:** Validate that a method correctly calculates the total cost based on rental duration and car type.
        - **Test:** Provide input data for car type and rental duration, and verify if the method returns the expected cost.
    - **Validation Methods:** Test methods that validate user input or data consistency.
        - **Scenario:** Ensure that a method correctly validates the format of a license number.
        - **Test:** Provide a sample license number and check if the method returns true for a valid format and false for an invalid one.
    - **Registration Methods:** Test methods that handle user registration.
        - **Scenario:** Validate that a method correctly handles user registration.
        - **Test:** Provide input data for new user registration and verify if the method successfully creates a new user account.

Integration Testing

1. Understanding Integration Testing

Definition:

- **Integration Testing:** Involves testing the interactions between multiple components or modules to ensure they work together correctly.

Purpose:

- **Verify Interactions:** Ensure that different parts of the system (e.g., user interface, business logic, database) interact correctly.

2. Integration Testing Approach

- **Integration Points:**
    - **UI and Business Logic:** Test interactions between the user interface and business logic components (e.g., booking a car through the UI triggers the correct booking logic).
    - **Business Logic and Database:** Verify that business logic components interact correctly with the database (e.g., saving and retrieving user data).
- **Example Scenarios:**
    - **Booking Workflow:** Test the complete booking process from user input through to database storage.
        - **Scenario:** A user books a car through the interface, and the booking details are saved correctly in the database.
        - **Test:** Simulate a complete booking process and check if the booking details are accurately recorded and retrievable.
    - **Availability Check:** Verify that the system correctly updates car availability after a booking.
        - **Scenario:** A car is booked, and the availability status is updated in the database.
        - **Test:** Perform a booking and verify that the car's availability status is accurately reflected.

System Testing

1. Understanding System Testing

Definition:

- **System Testing:** Involves testing the entire system as a whole to ensure it meets defined requirements and performs correctly across all components and configurations.

Purpose:

- **Verify Requirements:** Check that all specified requirements are fulfilled.
- **Validate End-to-End Functionality:** Ensure that the integrated system performs as expected in a real-world environment.

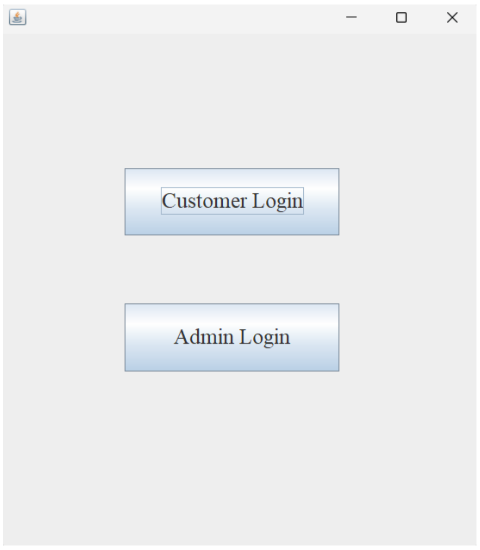
2. System Testing Approach

- **Test Scenarios:**
    - **Complete System Workflow:** Test the end-to-end functionality of the car booking system, from user registration to booking and service scheduling.
        - **Scenario:** A user registers, books a car, schedules a service, and logs out.
        - **Test:** Perform the complete user journey and verify that all components work together as expected.
    - **Performance Testing:** Assess the system's performance under various conditions (e.g., multiple concurrent users).
        - **Scenario:** Simulate multiple users booking cars simultaneously.
        - **Test:** Measure system response times and ensure the system handles load effectively.
    - **Usability Testing:** Evaluate the user interface and overall user experience.
        - **Scenario:** Test the ease of use and intuitive design of the car booking system.
        - **Test:** Gather feedback from users and make improvements based on their experiences.
    - **Security Testing:** Verify the security measures in place, such as data encryption and user authentication.
        - **Scenario:** Test access controls and data protection mechanisms.
        - **Test:** Attempt unauthorized access and ensure sensitive data is protected.

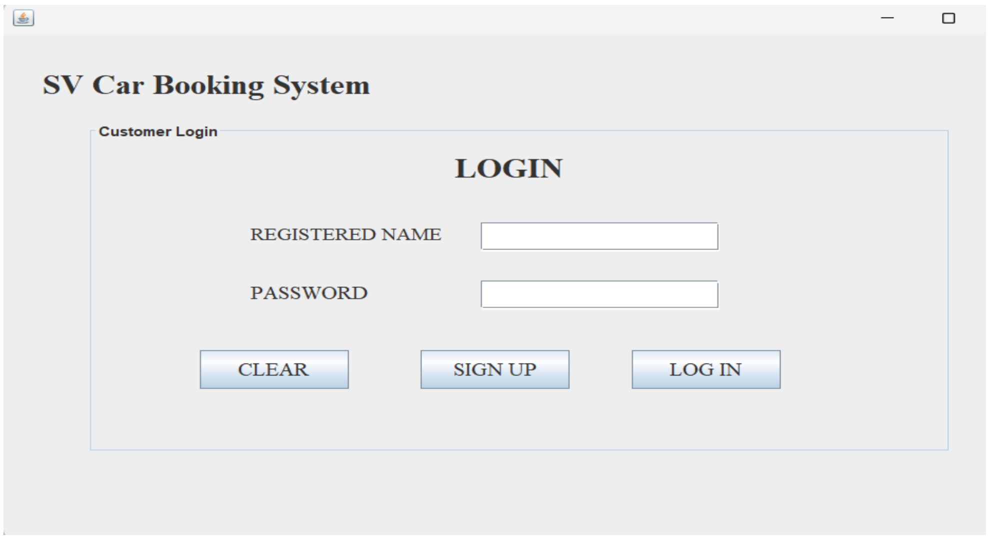
CHAPTER 9

RESULTS

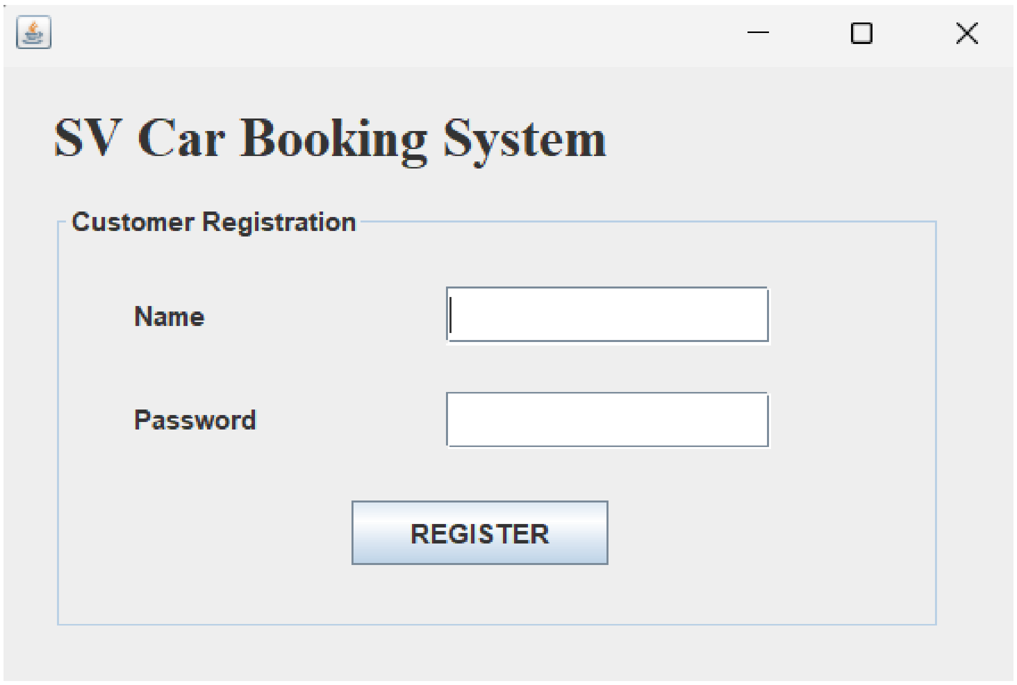
➢ **LOGIN:**



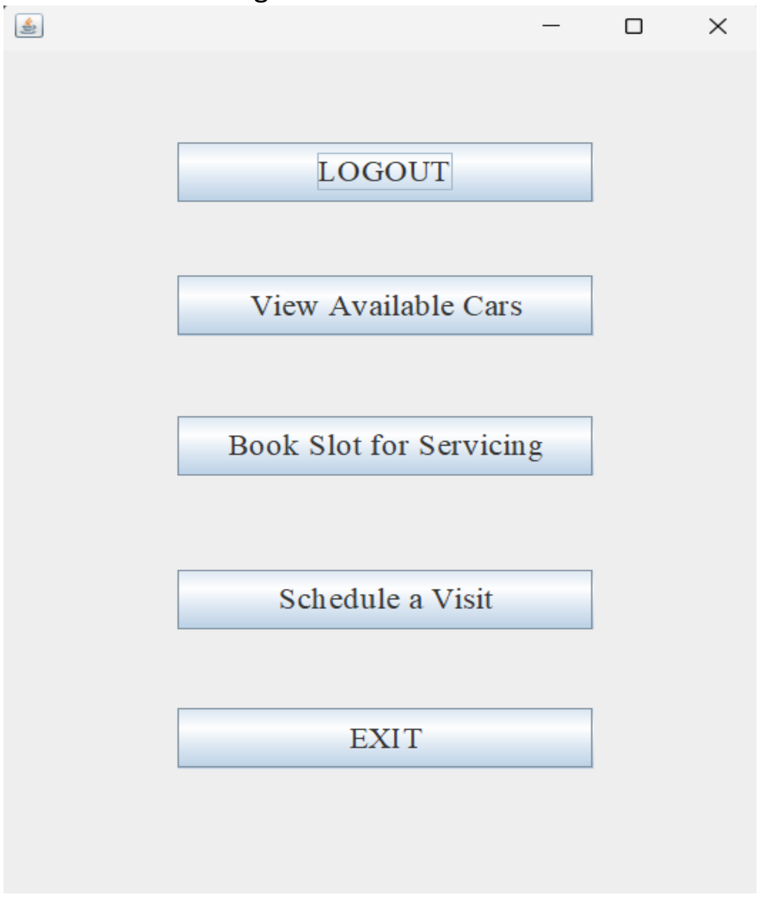
CusLogin.java



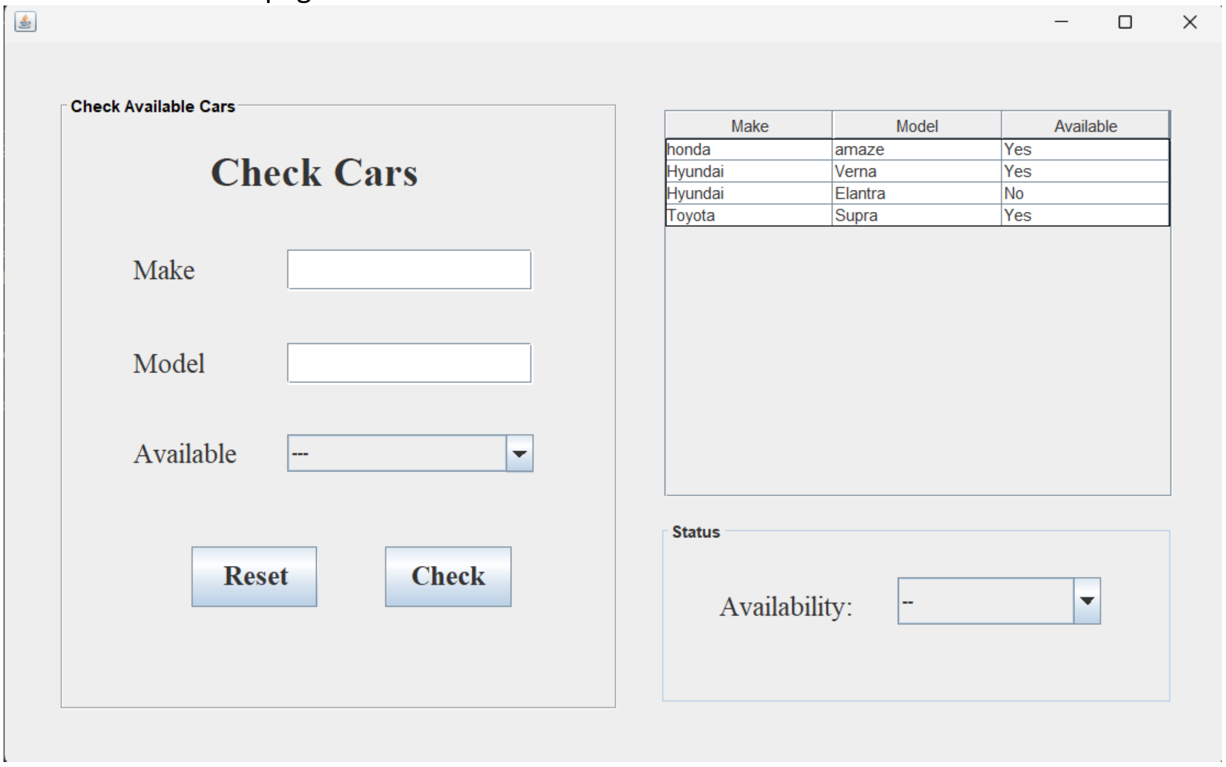
Customer Registration Page



Customer Menu Page



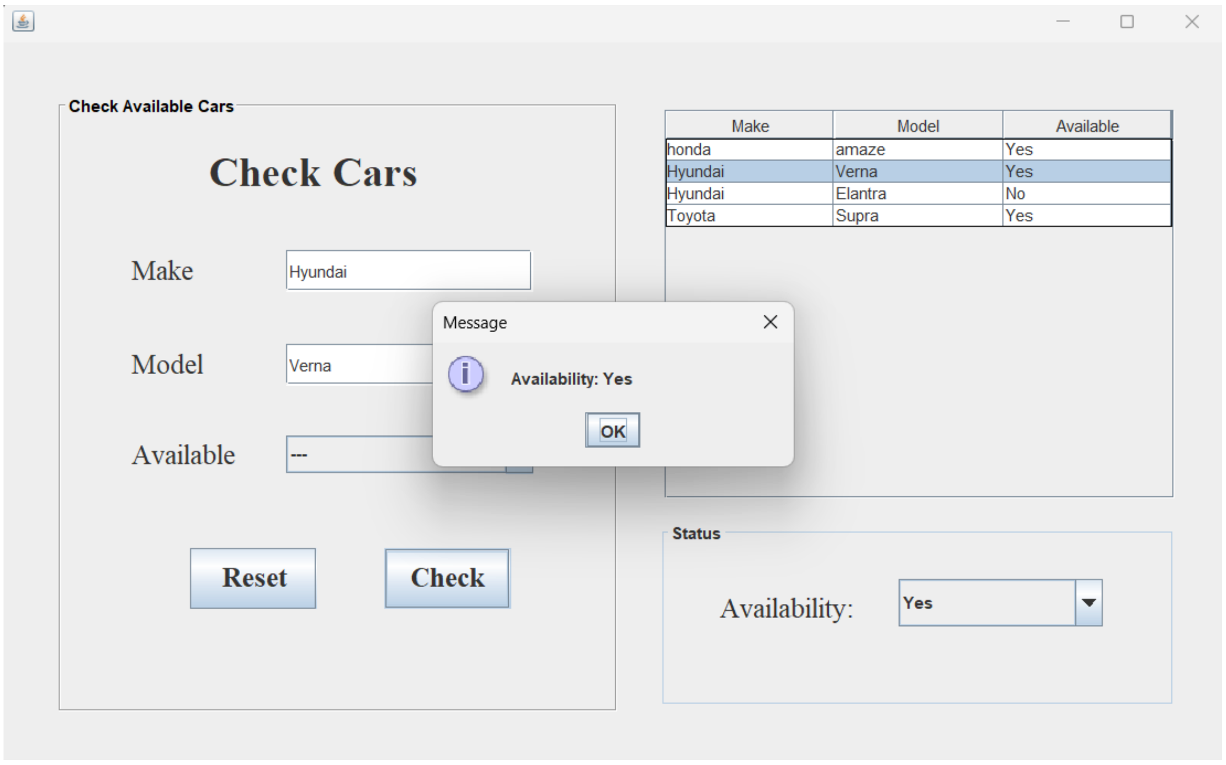
View Available Cars page:



Check Available Cars

Check Cars

Make	Model	Available
honda	amaze	Yes
Hyundai	Verna	Yes
Hyundai	Elantra	No
Toyota	Supra	Yes



Check Available Cars

Check Cars

Make: Hyundai

Model: Verna

Available: ---

Make	Model	Available
honda	amaze	Yes
Hyundai	Verna	Yes
Hyundai	Elantra	No
Toyota	Supra	Yes

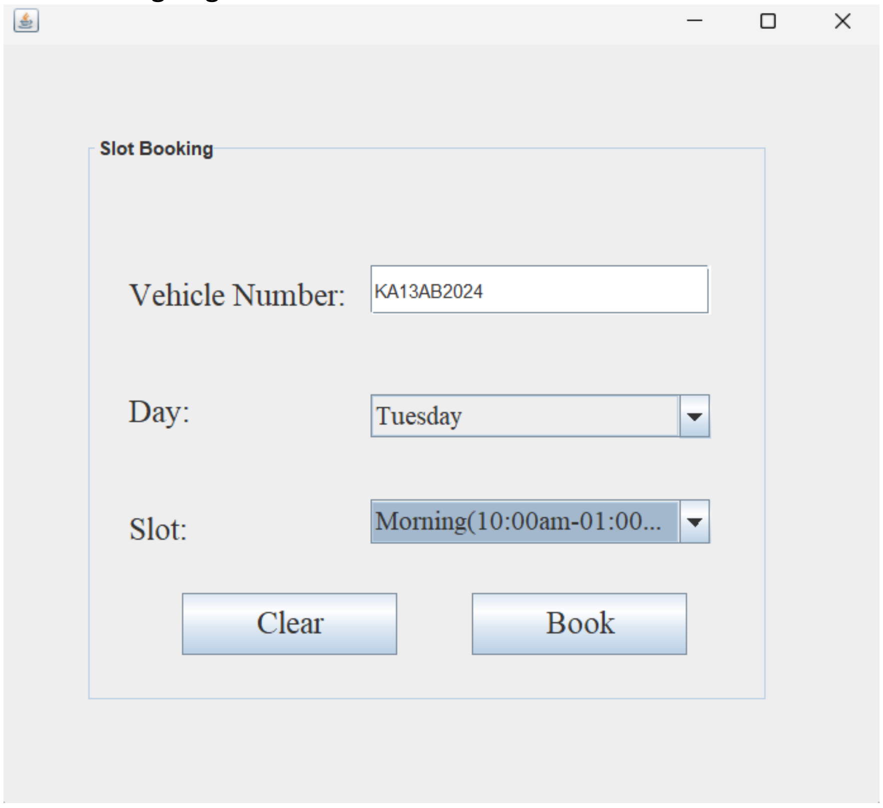
Message

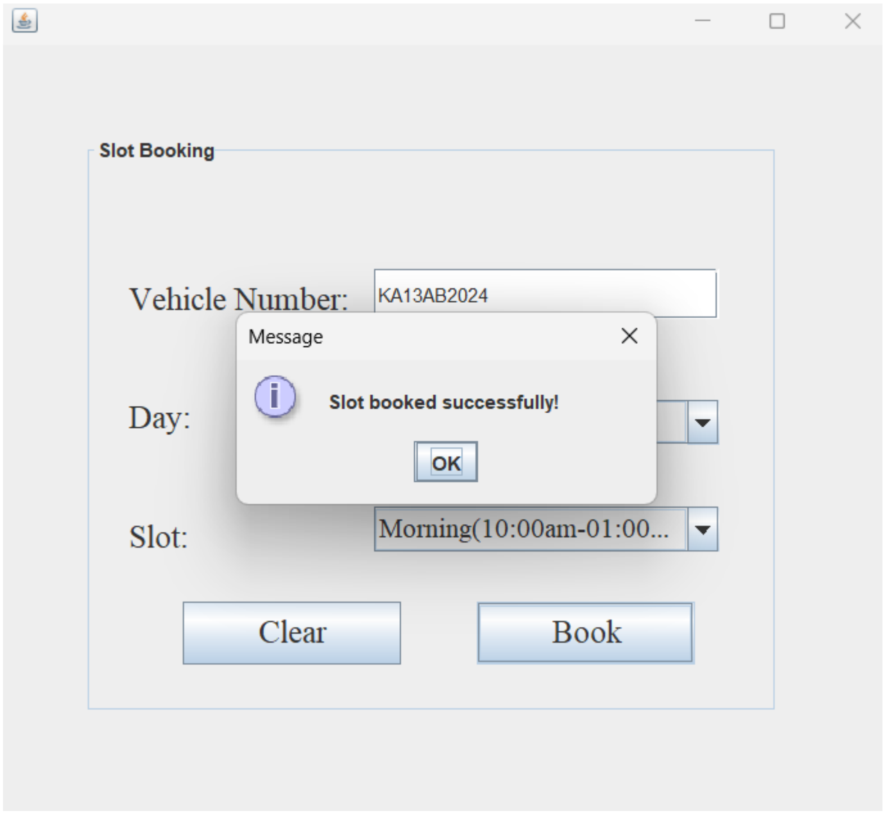
Availability: Yes

Status

Availability: Yes

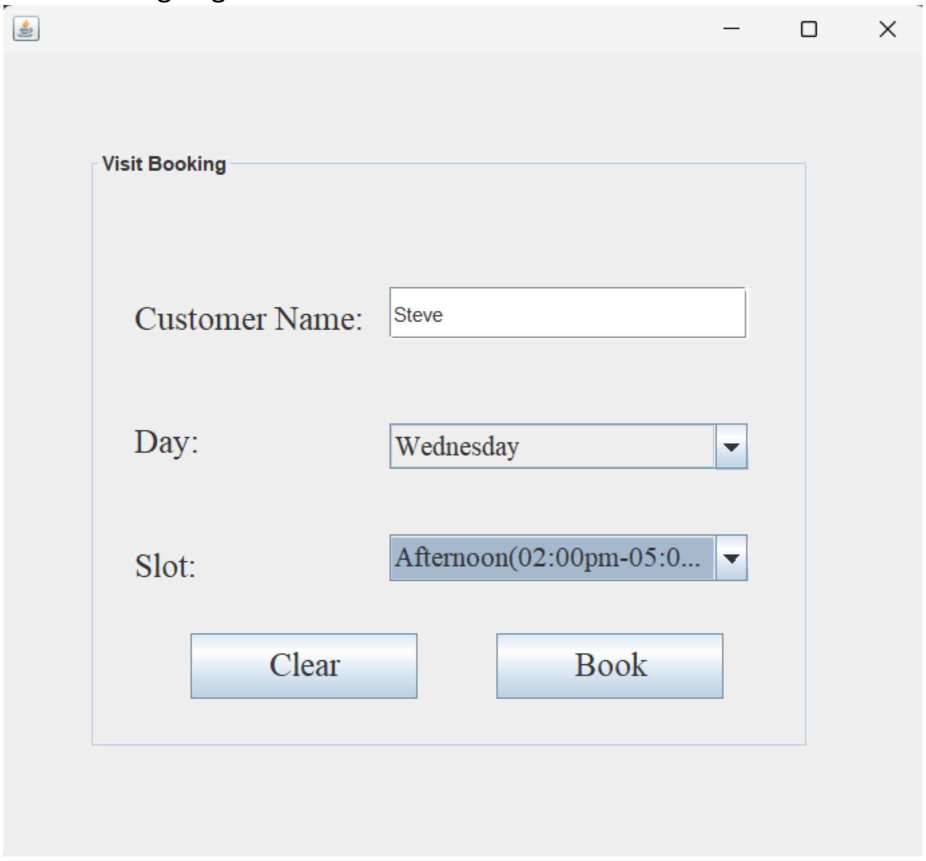
Slot Booking Page:





Slot booked successfully!

Visit Booking Page:

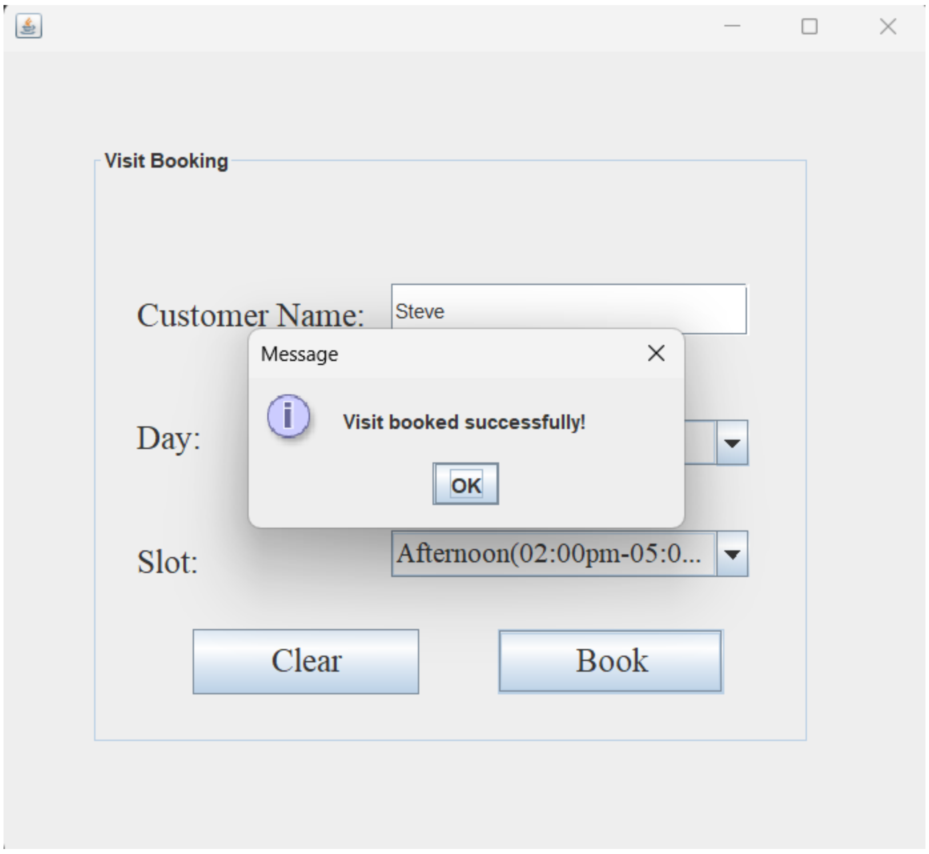


Visit Booking

Customer Name: Steve

Day: Wednesday

Slot: Afternoon(02:00pm-05:0...



Visit Booking

Customer Name: Steve

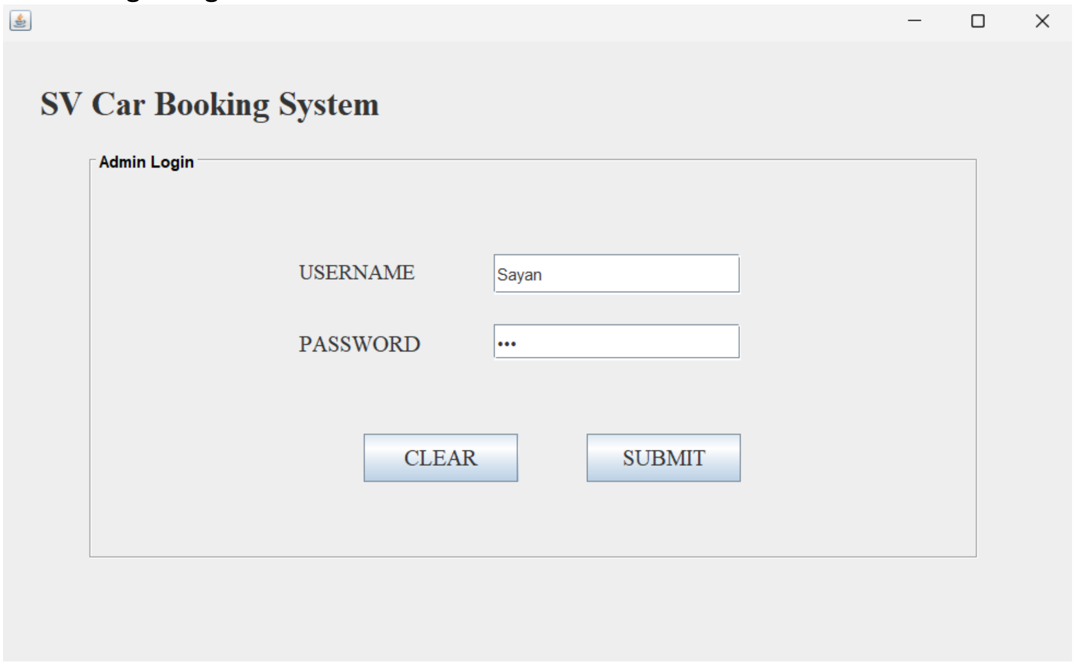
Day:

Slot: Afternoon(02:00pm-05:0...

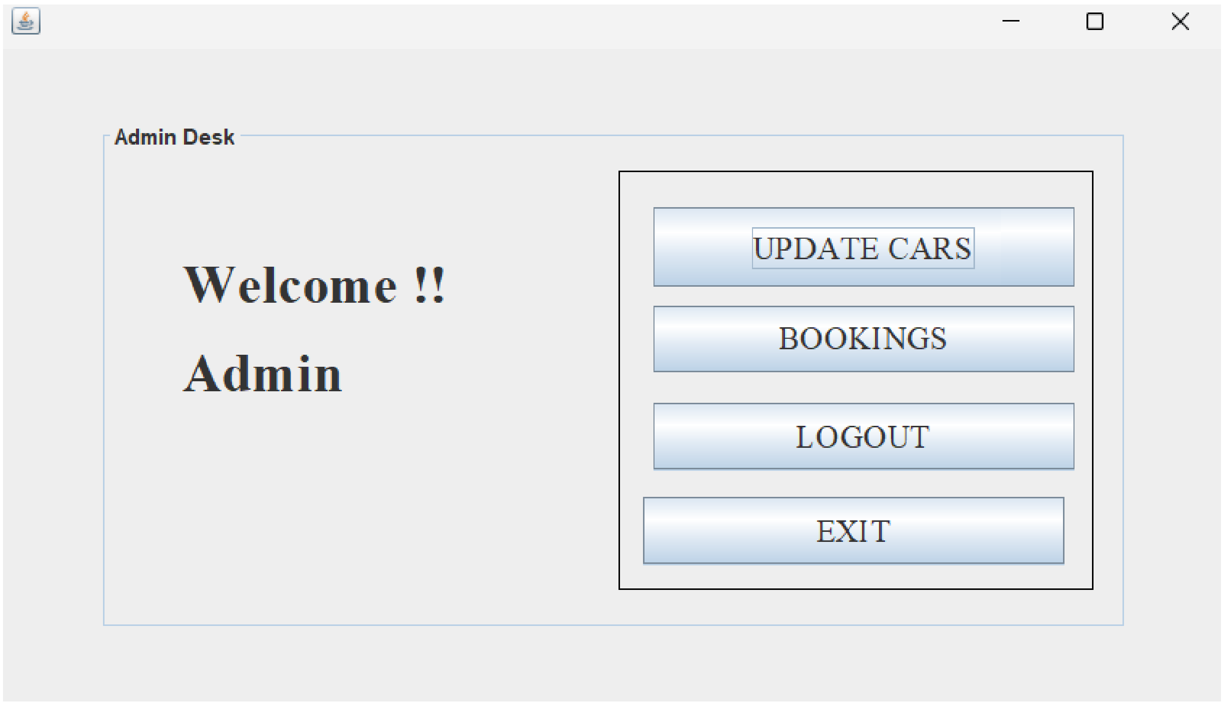
Message

Visit booked successfully!

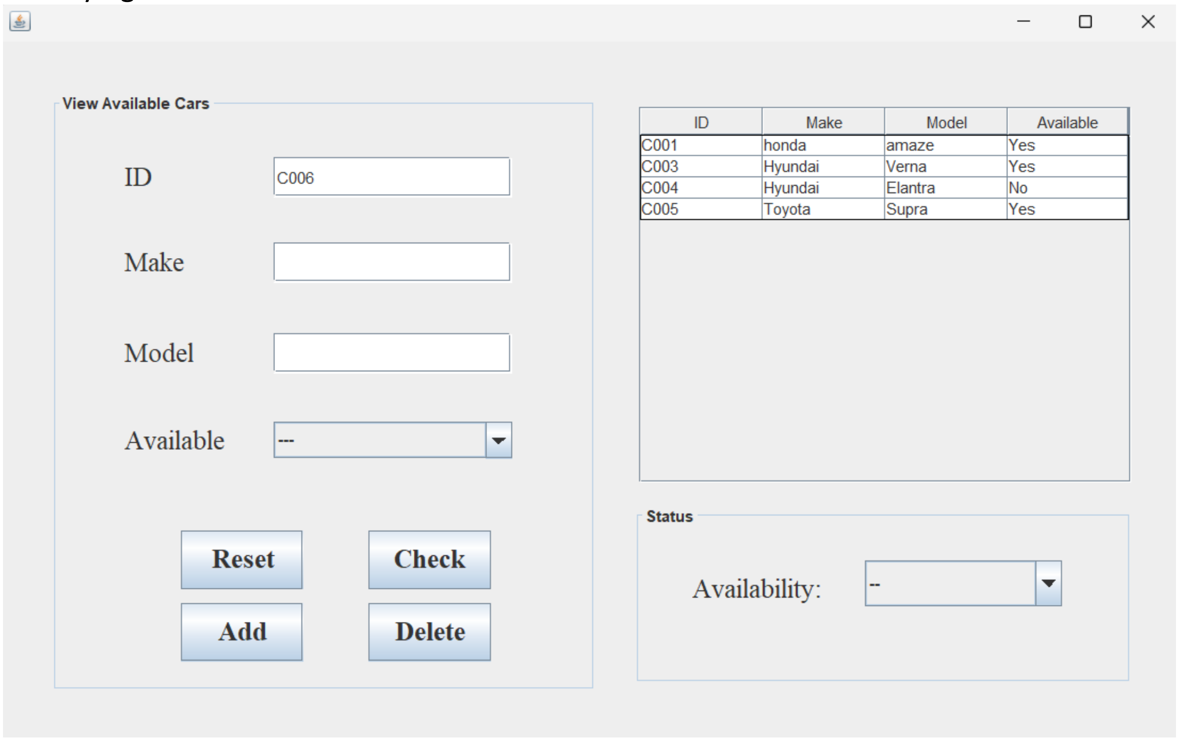
Admin Login Page:



Admin Desk:



Modifying Desk for Admin:



ID: C006

Make:

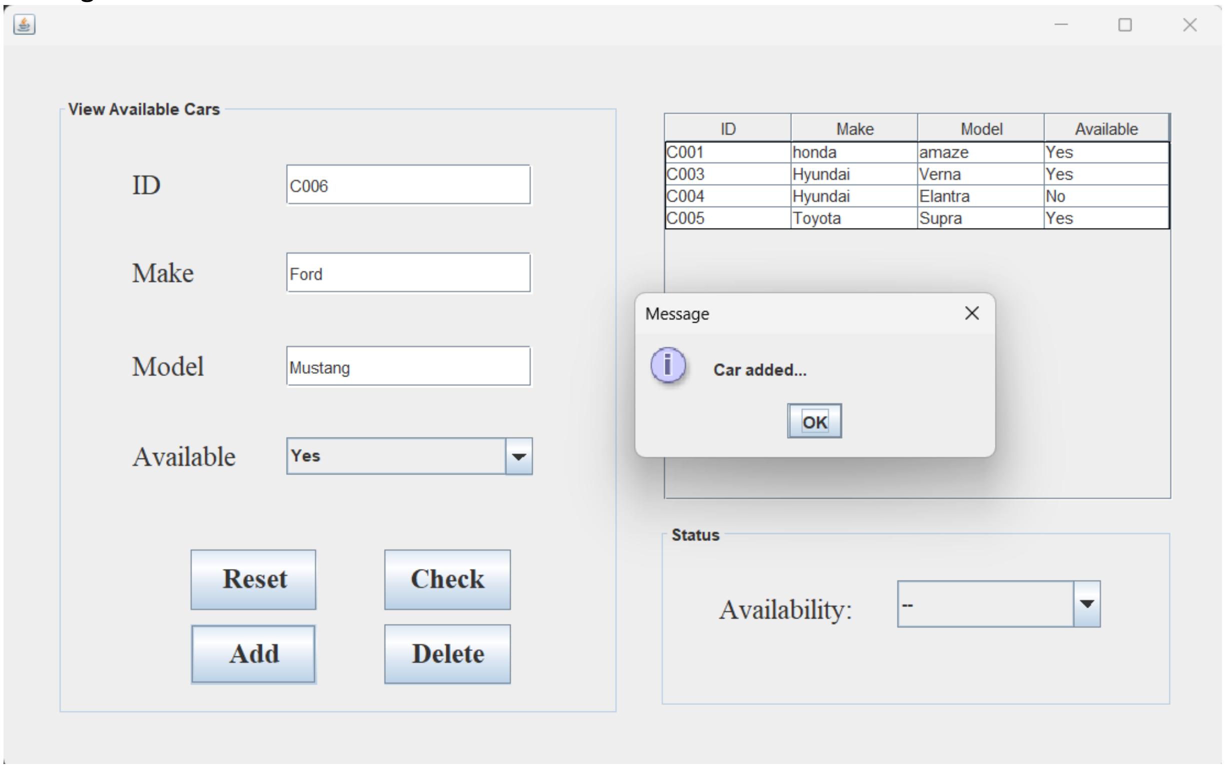
Model:

Available: ---

ID	Make	Model	Available
C001	honda	amaze	Yes
C003	Hyundai	Verna	Yes
C004	Hyundai	Elantra	No
C005	Toyota	Supra	Yes

Availability: --

Adding a Car:



ID: C006

Make: Ford

Model: Mustang

Available: Yes

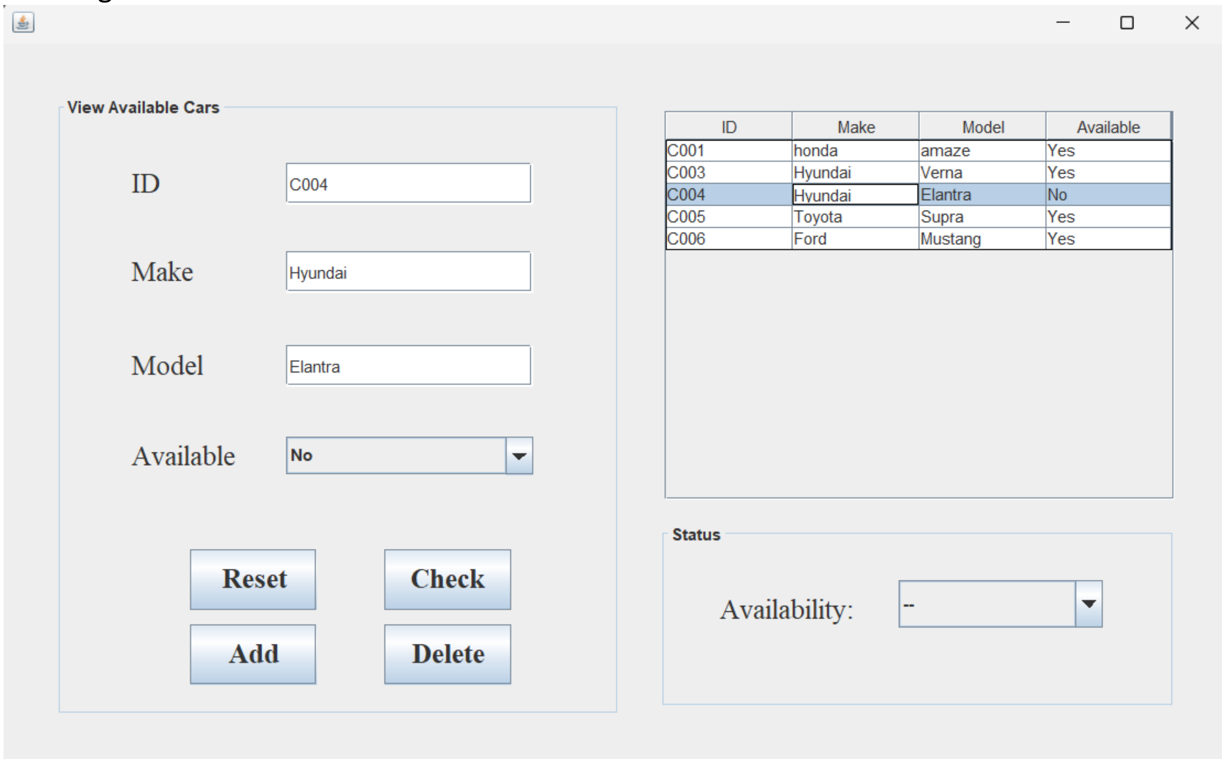
ID	Make	Model	Available
C001	honda	amaze	Yes
C003	Hyundai	Verna	Yes
C004	Hyundai	Elantra	No
C005	Toyota	Supra	Yes

Message

Car added...

Availability: --

Deleting a Car:



View Available Cars

ID C004

Make Hyundai

Model Elantra

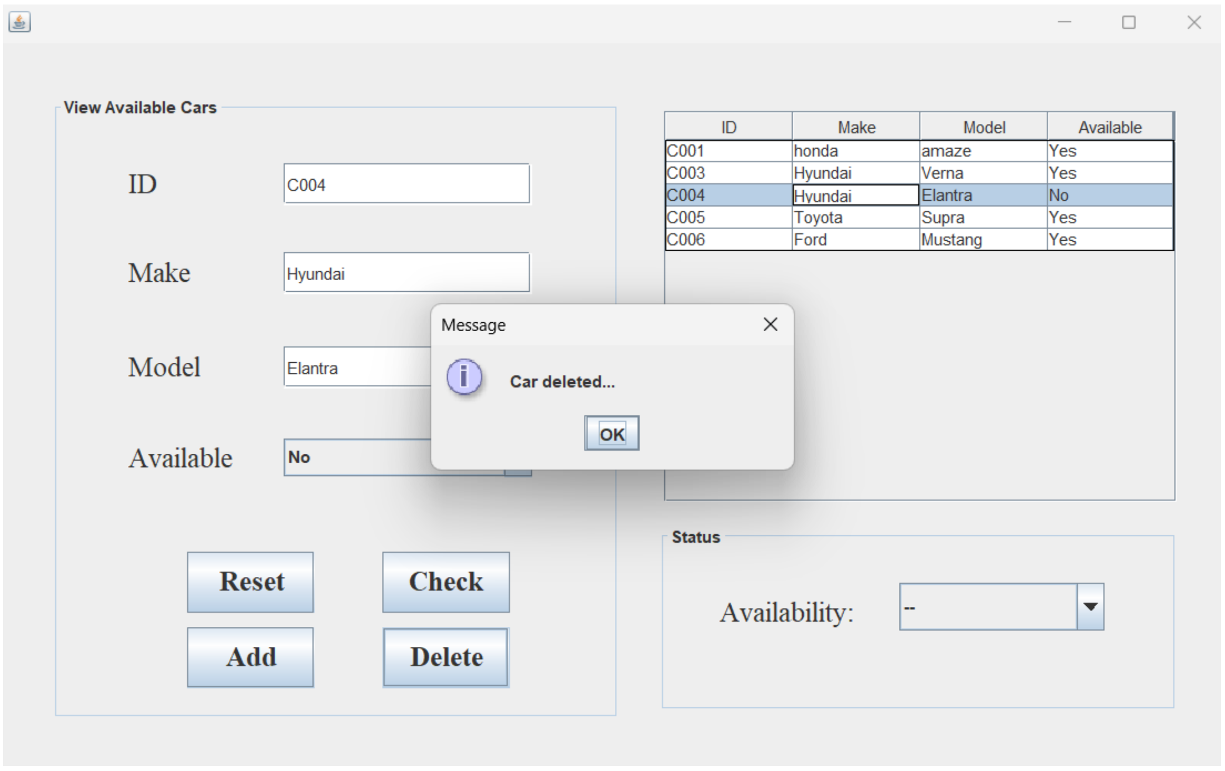
Available No

Reset Check Add Delete

ID	Make	Model	Available
C001	honda	amaze	Yes
C003	Hyundai	Verna	Yes
C004	Hyundai	Elantra	No
C005	Toyota	Supra	Yes
C006	Ford	Mustang	Yes

Status

Availability: --



View Available Cars

ID C004

Make Hyundai

Model Elantra

Available No

Reset Check Add Delete

ID	Make	Model	Available
C001	honda	amaze	Yes
C003	Hyundai	Verna	Yes
C004	Hyundai	Elantra	No
C005	Toyota	Supra	Yes
C006	Ford	Mustang	Yes

Message

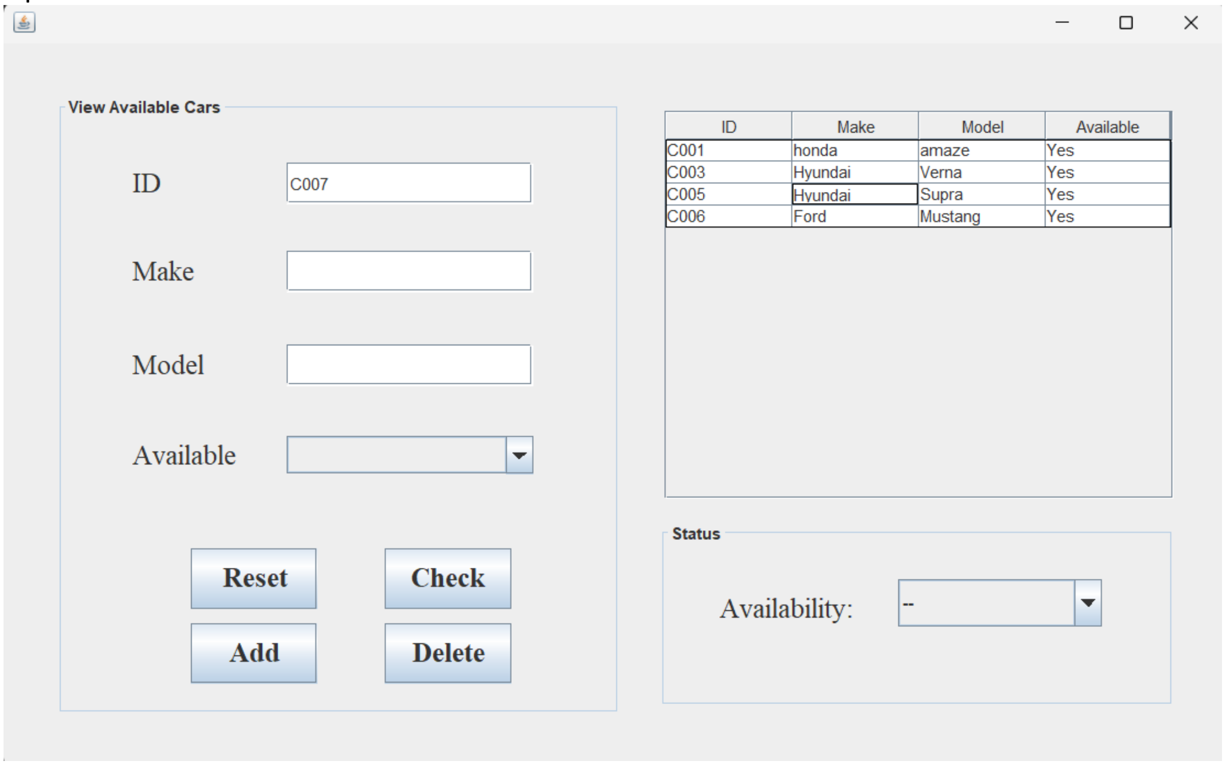
Car deleted...

OK

Status

Availability: --

Updated Table List:



View Available Cars

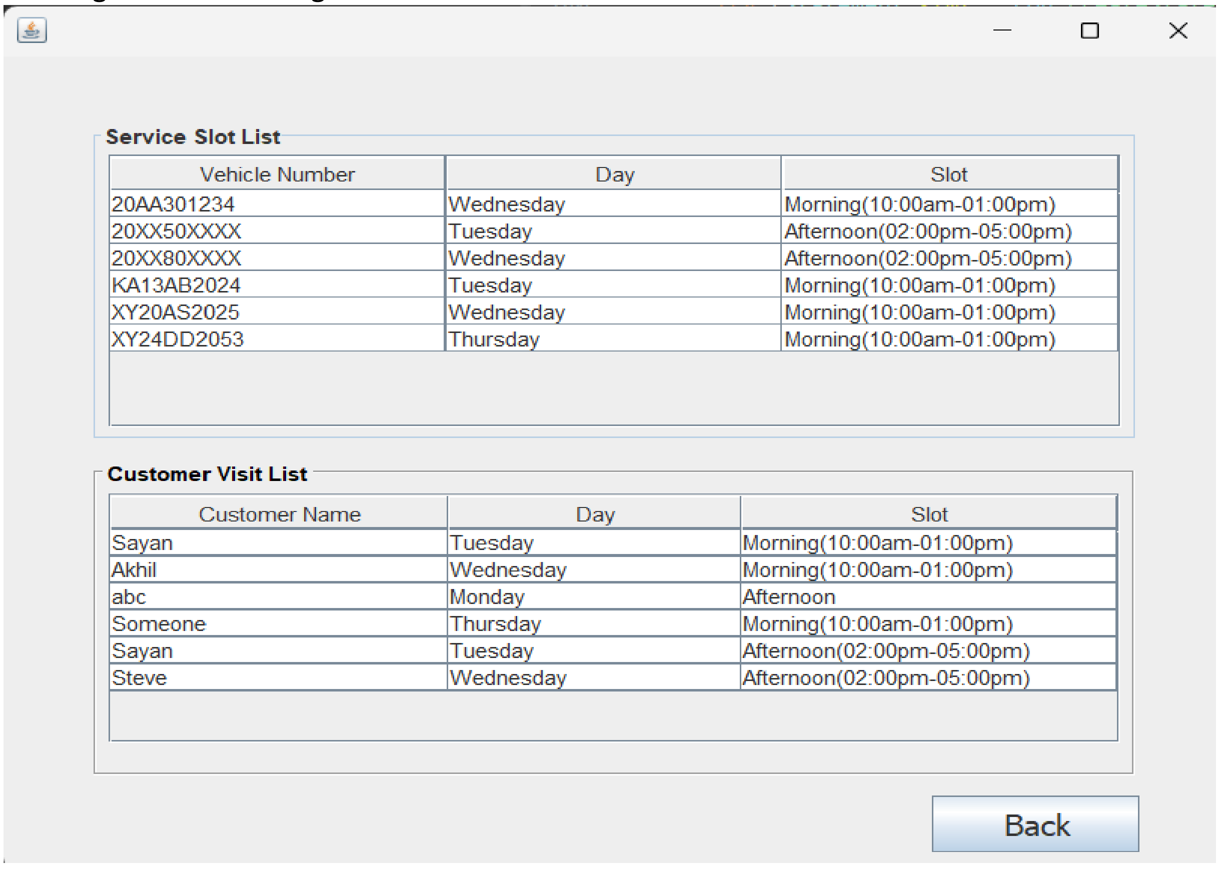
ID	C007
Make	
Model	
Available	

ID	Make	Model	Available
C001	honda	amaze	Yes
C003	Hyundai	Verna	Yes
C005	Hyundai	Supra	Yes
C006	Ford	Mustang	Yes

Status

Availability: --

Viewing The Car Bookings:



Service Slot List

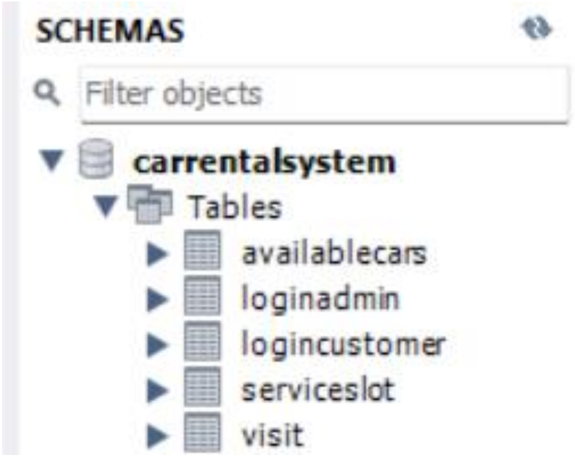
Vehicle Number	Day	Slot
20AA301234	Wednesday	Morning(10:00am-01:00pm)
20XX50XXXX	Tuesday	Afternoon(02:00pm-05:00pm)
20XX80XXXX	Wednesday	Afternoon(02:00pm-05:00pm)
KA13AB2024	Tuesday	Morning(10:00am-01:00pm)
XY20AS2025	Wednesday	Morning(10:00am-01:00pm)
XY24DD2053	Thursday	Morning(10:00am-01:00pm)

Customer Visit List

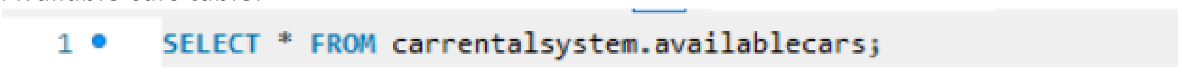
Customer Name	Day	Slot
Sayan	Tuesday	Morning(10:00am-01:00pm)
Akhil	Wednesday	Morning(10:00am-01:00pm)
abc	Monday	Afternoon
Someone	Thursday	Morning(10:00am-01:00pm)
Sayan	Tuesday	Afternoon(02:00pm-05:00pm)
Steve	Wednesday	Afternoon(02:00pm-05:00pm)

Database Result:

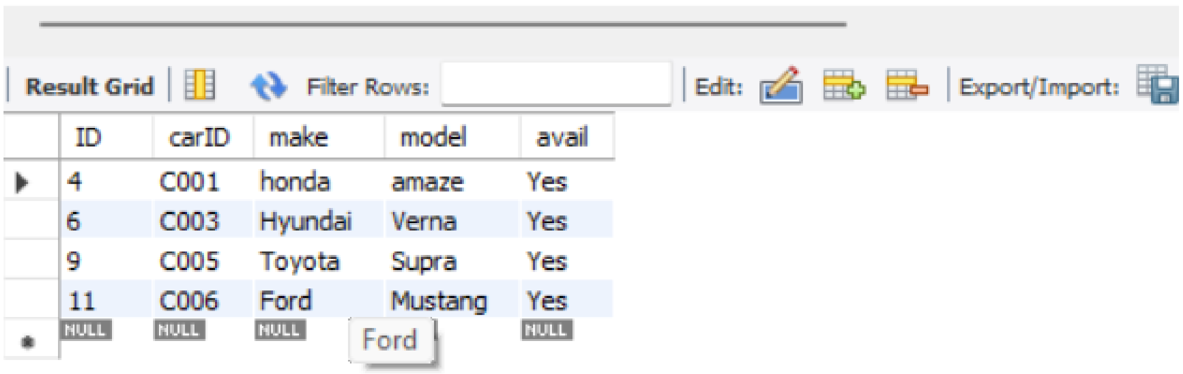
All Tables:



Available cars table:

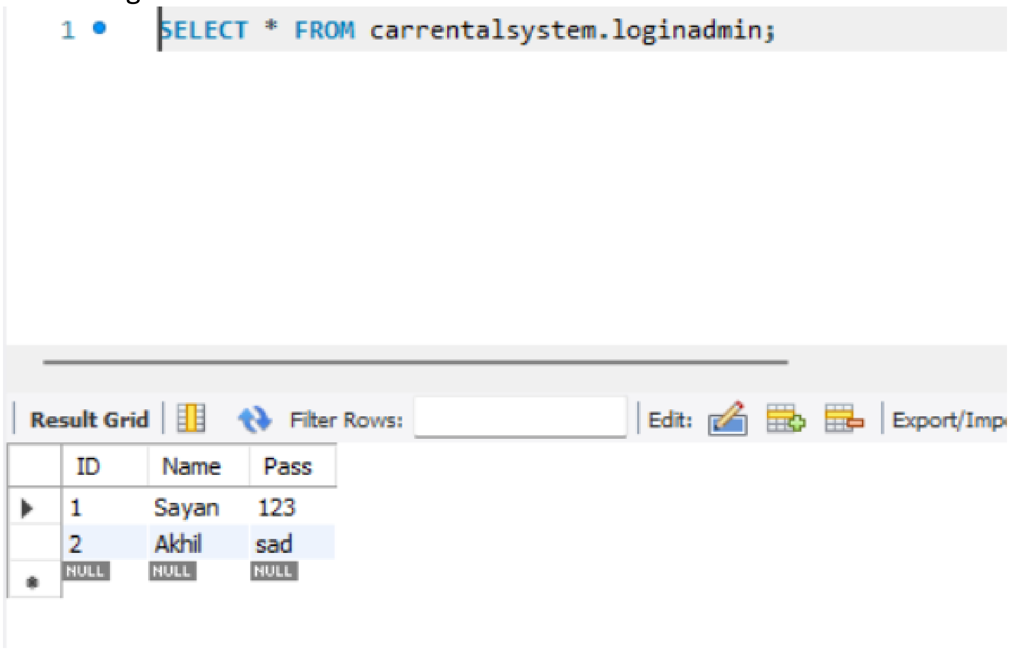


```
SELECT * FROM carrentalsystem.availablecars;
```



	ID	carID	make	model	avail
▶	4	C001	honda	amaze	Yes
	6	C003	Hyundai	Verna	Yes
	9	C005	Toyota	Supra	Yes
	11	C006	Ford	Mustang	Yes
*	NULL	NULL	NULL	Ford	NULL

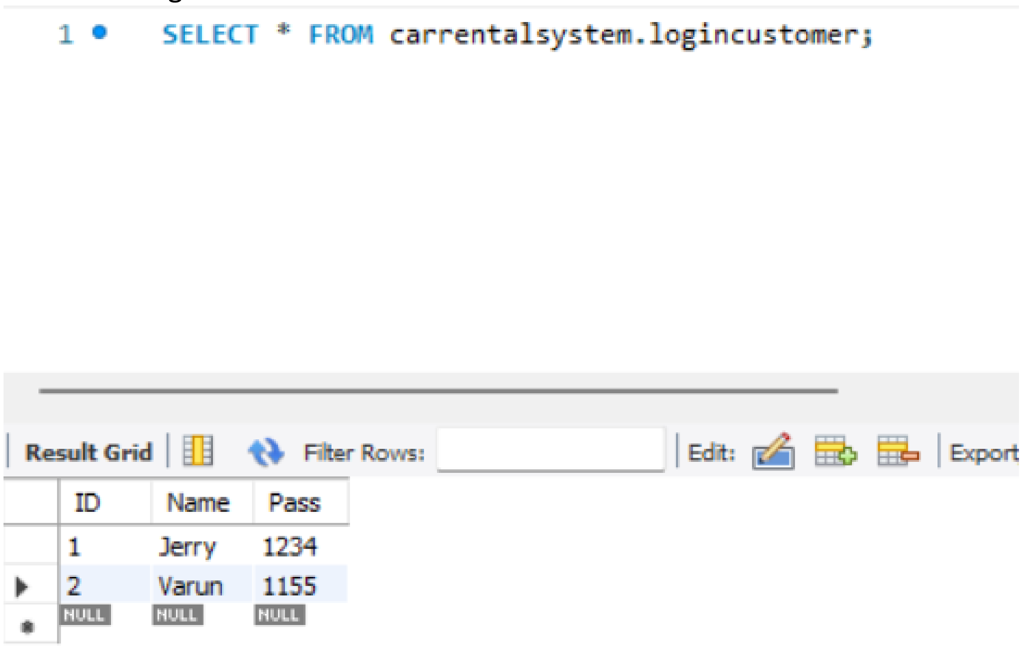
Admin Login Records:



```
SELECT * FROM carrentalsystem.loginadmin;
```

ID	Name	Pass
1	Sayan	123
2	Akhil	sad
NULL	NULL	NULL

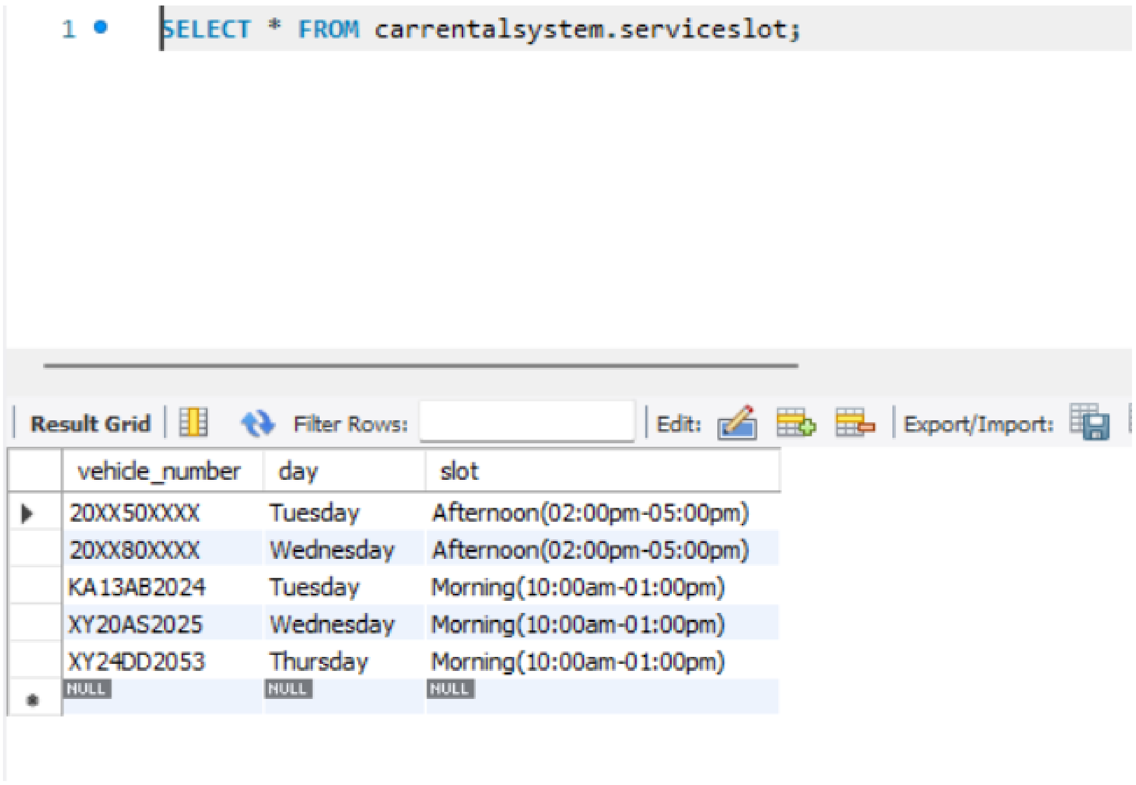
Customer Login Records:



```
SELECT * FROM carrentalsystem.logincustomer;
```

ID	Name	Pass
1	Jerry	1234
2	Varun	1155
NULL	NULL	NULL

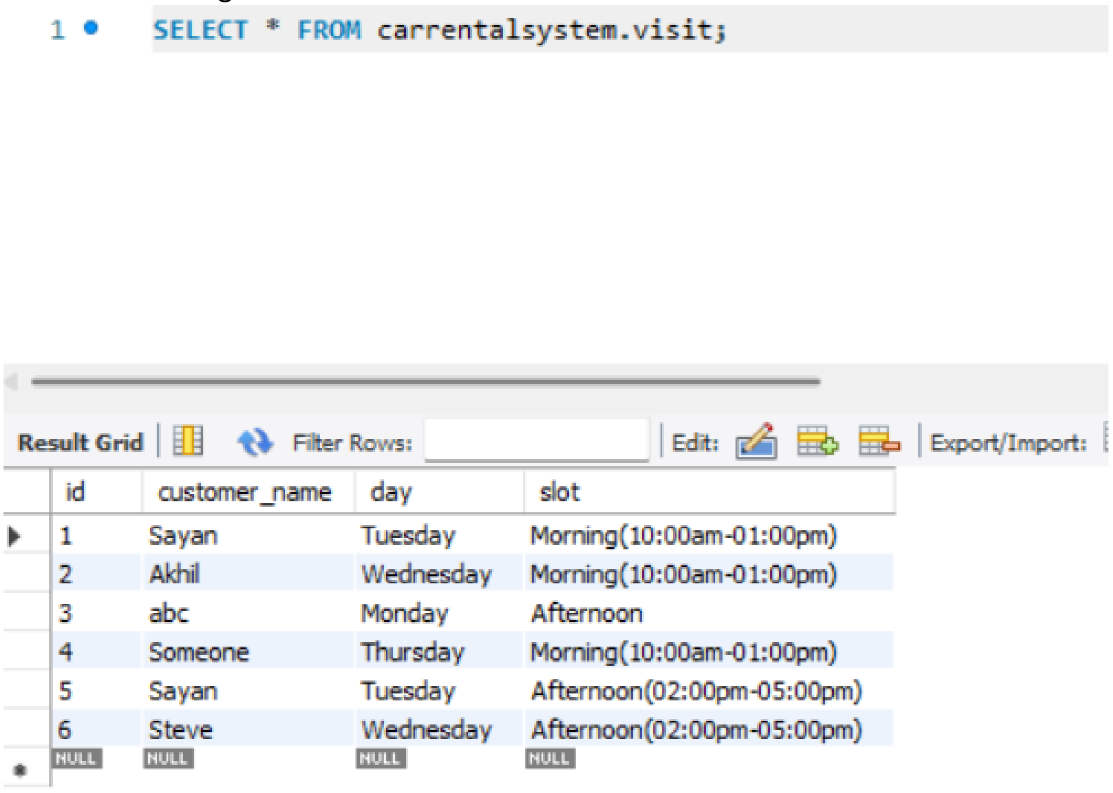
Service Slot Records:



```
SELECT * FROM carrentalsystem.serviceslot;
```

vehicle_number	day	slot
20XX50XXXX	Tuesday	Afternoon(02:00pm-05:00pm)
20XX80XXXX	Wednesday	Afternoon(02:00pm-05:00pm)
KA13AB2024	Tuesday	Morning(10:00am-01:00pm)
XY20AS2025	Wednesday	Morning(10:00am-01:00pm)
XY24DD2053	Thursday	Morning(10:00am-01:00pm)
NULL	NULL	NULL

Customer Visiting Records:



```
SELECT * FROM carrentalsystem.visit;
```

id	customer_name	day	slot
1	Sayan	Tuesday	Morning(10:00am-01:00pm)
2	Akhil	Wednesday	Morning(10:00am-01:00pm)
3	abc	Monday	Afternoon
4	Someone	Thursday	Morning(10:00am-01:00pm)
5	Sayan	Tuesday	Afternoon(02:00pm-05:00pm)
6	Steve	Wednesday	Afternoon(02:00pm-05:00pm)
NULL	NULL	NULL	NULL

CHAPTER 10

CONCLUSION

In conclusion, the car booking system developed in Java provides a comprehensive solution for managing vehicle rentals. By integrating a user-friendly graphical interface with robust backend support using MySQL, the system ensures efficient handling of both customer and administrative functionalities. Key features include secure login mechanisms, real-time vehicle availability tracking, and streamlined booking processes.

❖ REFERENCES:

1.Google

2.Youtube

3.Geeks for Geeks