Ride Booking System using JDBC

Abstract

This project is a simple console-based Ride Booking System developed in Java using JDBC (Java Database Connectivity). The application demonstrates how Java programs connect to a relational database and perform basic CRUD (Create, Read, Update, Delete) operations on ride records.

Introduction

JDBC is a standard Java API used to interact with databases such as MySQL. This project allows a user to add new rides, view all rides, update existing rides, and delete rides from the database through a simple text-based menu.

Objectives

• To understand JDBC connectivity in Java.
• To perform CRUD operations using SQL through Java.
• To build a simple console-based database application.

Software Requirements

• Java (JDK 8 or above)
• MySQL Database
• JDBC Driver (MySQL Connector/J)
• IDE or Command Line

Database Design

The following SQL is used to create the database and table:

```
-- Create database
CREATE DATABASE IF NOT EXISTS ride_db;

-- Use the database
USE ride_db;

-- Create ride_booking table
CREATE TABLE IF NOT EXISTS ride_booking (
  id INT PRIMARY KEY AUTO_INCREMENT,
  customer_name VARCHAR(50),
  pickup VARCHAR(50),
  drop_location VARCHAR(50),
  ride_date VARCHAR(20)
);
```

JDBC Connectivity

The application establishes a connection to the database using the DriverManager class. The connection string includes the database URL, username, and password. Once connected, SQL queries are executed using PreparedStatement and Statement objects.

Java Code and Function Explanation

1. Database Connection

This section of the code loads the JDBC driver and creates a connection to the database. The connection object is reused for executing SQL statements.

```
static Connection conn = null;
        conn = DriverManager.getConnection(DB_URL, USER, PASS);
```

2. Insert Ride (Create Operation)

This function accepts ride details from the user and inserts a new record into the database using a PreparedStatement to prevent SQL injection.

```
                insertBooking();
    System.out.println("1. Insert a new ride booking");
// Insert new booking
static void insertBooking() {
        System.out.println("\n--- Insert New Booking ---");
        String sql = "INSERT INTO ride_booking (customer_name, pickup, drop_location, ride_date) VA
            System.out.println("Booking inserted successfully!");
            System.out.println("Failed to insert booking.");
        System.out.println("Error inserting booking: " + e.getMessage());
```

3. View Rides (Read Operation)

This function retrieves all ride records from the database using a SELECT query and displays them in the console.

```
        String sql = "SELECT * FROM ride_booking";
        ResultSet rs = stmt.executeQuery(sql);
```

4. Update Ride (Update Operation)

This function modifies existing ride records based on the ride ID using an UPDATE SQL statement.

```
                updateBooking();
    System.out.println("3. Update a ride booking by ID");
        int rows = pstmt.executeUpdate();
// Update booking by ID
static void updateBooking() {
        System.out.println("\n--- Update Booking ---");
        System.out.print("Enter booking ID to update: ");
        String sql = "UPDATE ride_booking SET customer_name=?, pickup=?, drop_location=?, ride_date
        int rows = pstmt.executeUpdate();
            System.out.println("Booking updated successfully!");
        int rows = pstmt.executeUpdate();
```

5. Delete Ride (Delete Operation)

This function removes a ride from the database based on the ride ID using a DELETE SQL query.

```
                deleteBooking();
    System.out.println("4. Delete a ride booking by ID");
// Delete booking by ID
static void deleteBooking() {
        System.out.println("\n--- Delete Booking ---");
        System.out.print("Enter booking ID to delete: ");
        String sql = "DELETE FROM ride_booking WHERE id=?";
            System.out.println("Booking deleted successfully!");
```

System Working Flow

1. User runs the Java program.
2. Program establishes a connection with the database using JDBC.
3. Menu is displayed for ride operations.
4. User selects an option (Insert, View, Update, Delete).
5. Corresponding SQL query is executed on the database.

Conclusion

This project demonstrates a simple and effective use of JDBC for database connectivity in Java. By implementing CRUD operations, the Ride Booking System provides a clear understanding of how Java applications interact with relational databases in real-world scenarios.