# logo2.png

# BUS RESERVATION SYSTEM

# MINI PROJECT REPORT

# SUBMITTED BY

# LOKAA V 231501085

# PREESHAA B V 231501120

# JAGDISH KHANNAA K P 231501064

# 

**BONAFIDE CERTIFICATE**

Certified that this project report **“BUS RESERVATION SYSTEM”** is the Bonafide work of “**LOKAA V [231501085], PREESHA B V [231501120], JAGDISH KHANNA [231501064]** “who carried out the project work under my supervision.

**Submitted for the Practical Examination held on**

**SIGNATURE**

Mr. U. Kumaran,

Assistant Professor (SS)

AIML,

Rajalakshmi Engineering College,

(autonomous)

Thandalam, Chennai - 602 105

**INTERNAL EXAMINER EXTERNAL EXAMINER**

**ABSTRACT**

The **Bus Reservation System** is a desktop-based application designed to simplify and automate the process of booking and managing bus tickets. This project aims to address the inefficiencies and challenges associated with traditional manual reservation systems, such as errors, delays, and lack of real-time information. The system is built using **Java** for its core functionality, **Eclipse IDE** for development, and **MySQL** for database management. It provides an intuitive user interface for passengers to search for buses, view schedules, book or cancel tickets, and manage their booking history. Administrators can efficiently oversee bus schedules, update routes, manage user data, and generate reports for better decision-making.

The system employs **Java Database Connectivity (JDBC)** to establish a seamless connection between the application and the MySQL database, ensuring real-time updates and secure data transactions. Features like user authentication, seat availability checks, and error handling enhance the system’s reliability and user-friendliness. With a robust backend powered by MySQL, the application ensures secure storage and retrieval of user profiles, booking records, and bus schedules. The project demonstrates the importance of automation in transportation services, offering a scalable and efficient solution that can be enhanced with features like online payments, live bus tracking, and mobile app integration. Ultimately, the **Bus Reservation System** delivers convenience and operational efficiency, benefiting both passengers and administrators in a fast-paced world. The **Bus Reservation System** not only simplifies the ticket booking process for passengers but also enhances the operational efficiency of bus service providers. By leveraging the power of automation and real-time data management, it minimizes human errors and ensures a smooth reservation experience. The system is scalable, making it adaptable for future enhancements such as online payments and live tracking, further improving its utility. This project serves as a practical solution to modernize traditional transportation systems, ensuring reliability, convenience, and user satisfaction.

**TABLE OF CONTENTS**

1. **INTRODUCTION**
2. PROJECT SUMMARY ………………………………………...5
3. OBJECTIVES …………………………………………………5
4. SCOPE………………………………………………………...6
5. **SYSTEM REQUIREMENTS**
6. . HARDWARE REQUIREMENTS…………………………….7
7. SOFTWARE REQUIREMENTS ……………………………....8

**3. SYSTEM DESIGN**

1. ARCHITECTURAL DESIGN…………………………………..9
2. COMPONENTS OF SYSTEM………………………………….9
3. DATABASE DESIGN…………………………………………11
4. TOOLS AND TECHNOLOGY…………………………………12
5. ADVANTAGES………………………………………………..12

**4. JDBC**

1. KEY FEATURES………………………………………………13
2. JDBC ARCHITECTURE………………………………………14
3. SETUP OF JDBC ……………………………………………....14
4. ADVANTAGES………………………………………………...16
5. ROLE IN BUS RESERVATION SYSTEM………………….........16

**5. IMPLEMENTATION**

1. PROJECT STRUCTURE…………………………………………17
2. PACKAGES……………………………………………………...19
3. CLASSES………………………………………………………...20
4. DATABASE-MYSQL…………………………………………….41

**6. OUTPUT……………………………………………………………..42**

**7. CONCLUSION ……………………………………………………….50**

**8. REFERENCES ………………………………………………………..52**

# 1.INTRODUCTION

## Project Summary

The **Bus Reservation System** is a desktop-based application designed to streamline and enhance the process of booking and managing bus tickets. The system addresses the challenges faced in traditional manual reservation methods, such as inefficiency, errors, and lack of real-time information. By leveraging the power of Java programming and MySQL database management, the project provides an automated, user-friendly, and reliable solution for both passengers and administrators.

This project is developed using the **Eclipse Integrated Development Environment (IDE)** for coding and debugging, while **MySQL** serves as the backend database for storing and managing data securely. The system includes features like user registration, bus schedule management, ticket booking, and payment tracking. It also includes an administrative interface to oversee and update bus schedules, monitor bookings, and manage user data effectively.

## Objective

The primary objective of the system is to simplify the booking process, reduce manual workload, and provide accurate, real-time information to the users. It is particularly beneficial for bus operators and passengers by ensuring a seamless and error-free reservation experience.

This report outlines the design, implementation, and testing of the **Bus Reservation System** and highlights its impact as a practical solution for addressing current limitations in bus ticketing operations.

The **Bus Reservation System** not only simplifies the reservation process for passengers but also helps administrators manage operations more effectively. It aims to provide a seamless experience by incorporating functionalities such as efficient database handling, intuitive interfaces, and responsive error management. Moreover, it ensures that the data is stored securely and can be retrieved or updated promptly without any complications.

**Scope**

In a broader context, the project highlights the importance of automation in enhancing service delivery. Transportation is an essential aspect of modern life, and implementing a digital system like this can significantly improve the efficiency and reliability of bus services. By digitizing the reservation process, this system reduces human intervention, minimizes errors, and fosters a more organized approach to managing bus services.

This report delves into the details of the project, from the initial analysis and design to the implementation and testing phases. It provides insights into the tools and technologies used, explains the system's architecture, and evaluates its performance. The project also explores the challenges encountered during development and suggests future enhancements to make the system more versatile and adaptable. Through this project, a reliable and practical solution has been developed to meet the growing demand for efficient bus reservation systems in today’s fast-paced world.

The **Bus Reservation System** is designed to streamline and automate the process of booking and managing bus tickets, addressing the inefficiencies of traditional manual methods. It provides features for passengers, such as user registration, bus search, real-time schedule viewing, ticket booking, cancellations, and access to booking history. Administrators can efficiently manage bus schedules, update routes, oversee user data, and generate reports for operational insights.

# 2.SYSTEM REQUIREMENTS

The development of the **Bus Reservation System** requires specific hardware and software resources to ensure smooth execution and optimal performance. Below is a detailed description of the hardware and software requirements, along with the technologies used for this project.

### ****Hardware Requirements****

To run and develop the Bus Reservation System, the following minimum hardware specifications are recommended:

1. **Processor**
   * Minimum: Intel Core i3 or equivalent
   * Recommended: Intel Core i5 or higher for better performance
2. **RAM**
   * Minimum: 4 GB
   * Recommended: 8 GB or more for smoother multitasking and faster application execution
3. **Storage**
   * Minimum: 500 GB Hard Disk Drive (HDD)
   * Recommended: 256 GB Solid-State Drive (SSD) or higher for faster data access and program loading
4. **Monitor and Display**
   * Minimum: 1280 x 720 resolution
   * Recommended: Full HD (1920 x 1080) resolution
5. **Other Peripherals**
   * Keyboard, mouse, and optional external storage devices for backups

### ****Software Requirements****

The system requires a set of specific software tools and platforms for development and execution:

1. **Operating System**
   * Windows 10/11, Linux, or macOS
2. **Development Tools**
   * **Eclipse IDE:** Integrated Development Environment for writing, debugging, and compiling Java code.
3. **Database Management System**
   * **MySQL:** Used to design and manage the relational database for storing user data, bus details, and booking records.
4. **Java Development Kit (JDK)**
   * Version 8 or higher, required for compiling and running Java applications.
5. **Libraries/Frameworks**
   * **JDBC (Java Database Connectivity):** For establishing a connection between the Java application and MySQL database.
   * **Swing/JavaFX:** For creating the graphical user interface (GUI).
6. **Other Software**
   * Text editors (e.g., Notepad++, Sublime Text) for quick edits.

The combination of the above hardware and software ensures that the Bus Reservation System is developed and executed effectively. The choice of **Java** for coding, **MySQL** for database management, and **Eclipse IDE** for development offers a robust and scalable framework to deliver a reliable system that meets user expectations. By using readily available and widely supported technologies, the project is accessible to developers and users with basic resources.

# 3. SYSTEM DESIGN

The **system design** of the Bus Reservation System involves a detailed analysis of its architecture, including the interaction between the **frontend (Eclipse IDE)**, the **backend (MySQL database)**, and other key components. The design ensures efficient data flow, robust functionality, and a seamless user experience.

### ****Architectural Design****

The system follows a **Client-Server Architecture**:

* **Client Side:** The application is developed in **Java** using the **Eclipse IDE**. It provides a graphical user interface (GUI) for users to interact with the system.
* **Server Side:** The **MySQL database** acts as the backend, handling data storage and retrieval. The database stores user profiles, bus schedules, bookings, and other operational data.

### ****Components of the System****

#### ****A. Presentation Layer (Frontend)****

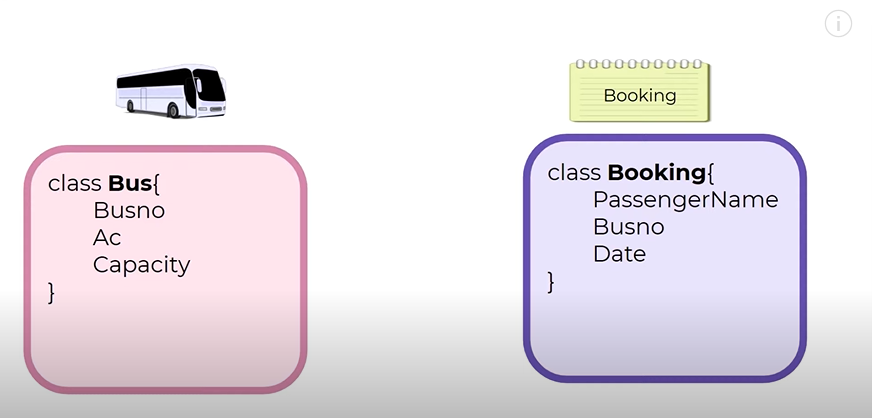
* Developed using **Java Swing** or **JavaFX** for the GUI.
* Responsible for user interactions, including forms for login, registration, ticket booking, and schedule viewing.
* Input data from users is validated here before being sent to the backend.

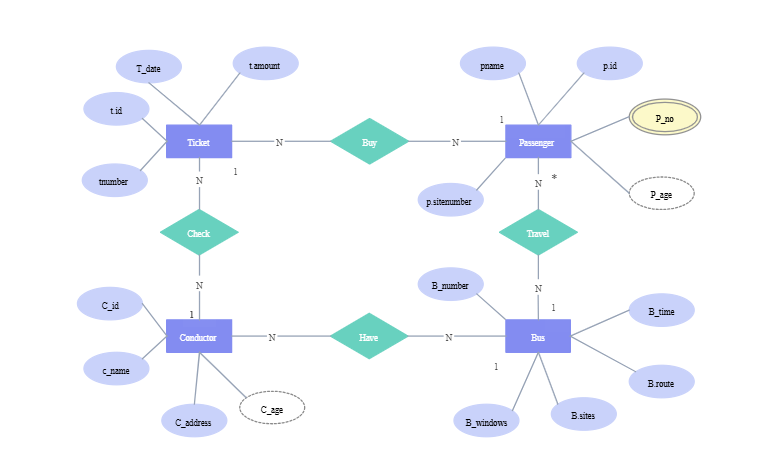
#### ****B. Business Logic Layer (Application Layer)****

* Implements the core functionalities of the system, such as:
  + Authentication and authorization (user login/registration).
  + Booking logic (checking seat availability, updating reservations).
  + Schedule management.
  + Data validation and error handling.
* This layer communicates with the MySQL database using **JDBC (Java Database Connectivity)**.

#### ****C. Data Layer (Database/Backend)****

* MySQL serves as the relational database to store and manage all system data, including:
  + **Users Table:** Stores user details (e.g., name, contact, credentials).
  + **Buses Table:** Contains bus details such as ID, route, timing, and fare.
  + **Reservations Table:** Maintains booking records, including seat allocation and payment status.
  + **Admin Table:** Holds data for administrators who manage the system.





# A Bus Reservation System ERD is a comprehensive database diagram illustrating the relationships between entities within a bus reservation system. This ERD can be used for software design and development, helping developers create structured and efficient software. It depicts the various entities such as buses, passengers, routes, and ticket reservations. It also shows the various relationships among these entities, such as how a passenger can book a ticket or how buses are related to a particular route. By highlighting these relationships, the ERD provides an effective way for developers to easily visualize and develop the software. In addition, it aids in efficient data access, allowing quick retrieval of data when needed.

### ****Database Design (MySQL)****

#### ****A. Tables and Attributes****

1. **Users Table:**
   * user\_id (Primary Key)
   * name
   * email
   * password
   * phone
2. **Buses Table:**
   * bus\_id (Primary Key)
   * route
   * departure\_time
   * arrival\_time
   * fare
   * total\_seats
   * available\_seats
3. **Bookings Table:**
   * booking\_id (Primary Key)
   * user\_id (Foreign Key)
   * bus\_id (Foreign Key)
   * booking\_date
   * seat\_number
   * status (Booked/Cancelled)
4. **Admin Table:**
   * admin\_id (Primary Key)
   * username
   * password

### ****Tools and Technologies Used****

1. **Eclipse IDE:**
   * Used for Java development and designing the GUI.
   * Facilitates integration with MySQL using the JDBC driver.
2. **MySQL:**
   * Manages all backend data with optimized queries for fast retrieval.
   * Ensures data integrity and supports relationships between tables.
3. **Java Database Connectivity (JDBC):**
   * Establishes a connection between the Java application and the MySQL database.

### 

### ****Advantages of the Design****

* **Scalability:** Easily add new features like online payments or mobile app integration.
* **Data Integrity:** MySQL ensures consistent and secure data storage.
* **User-Friendly:** The intuitive GUI simplifies user interaction.
* **Real-Time Updates:** Ensures accurate and up-to-date seat availability.

This system design ensures that the **Bus Reservation System** is robust, secure, and capable of delivering a seamless experience to both passengers and administrators.

# 4. JAVA DATABASE CONNECTIVITY

**Java Database Connectivity (JDBC)** is an API (Application Programming Interface) that enables Java applications to interact with relational databases. It acts as a bridge between a Java program and a database, allowing developers to execute SQL queries, retrieve results, and update data from within their Java applications. In the **Bus Reservation System**, JDBC is used to connect the Java application, built using Eclipse IDE, with the MySQL database.

### ****Key Features of JDBC****

1. **Database Independence**  
   JDBC supports multiple relational databases, including MySQL, Oracle, PostgreSQL, and SQL Server. This ensures flexibility in database selection.
2. **Simplified API**  
   JDBC provides an easy-to-use API for executing queries and managing database operations directly from Java.
3. **Dynamic Query Execution**  
   Supports both **static SQL queries** (predefined) and **dynamic SQL queries** (constructed during runtime).
4. **Result Set Handling**  
   JDBC offers classes like ResultSet to fetch, manipulate, and display data retrieved from a database.
5. **Support for Transactions**  
   JDBC provides methods for managing database transactions, ensuring data consistency and integrity.
6. **Error Handling**  
   Offers robust error handling mechanisms to deal with SQL exceptions and connectivity issues.

### ****JDBC Architecture****

The JDBC architecture includes the following key components:

1. **Driver Manager**
   * Manages database drivers and establishes a connection between the Java application and the database.
2. **JDBC Driver**
   * A specific implementation that allows communication with a particular database (e.g., MySQL JDBC driver for MySQL).
3. **Connection**
   * Represents the session between the Java application and the database.
4. **Statement**
   * Used to execute SQL queries (e.g., SELECT, INSERT, UPDATE, DELETE).
5. **ResultSet**
   * Holds the results of a query executed through JDBC.

### ****Steps to Use JDBC in the Bus Reservation System****

1. **Import JDBC Package**

java

Copy code

import java.sql.\*;

1. **Load the JDBC Driver**  
   Register the JDBC driver for the database you are using. For MySQL:

java

Copy code

Class.forName("com.mysql.cj.jdbc.Driver");

1. **Establish a Connection**  
   Use the DriverManager class to establish a connection with the database:

java

Copy code

Connection con = DriverManager.getConnection(

"jdbc:mysql://localhost:3306/bus\_reservation", "username", "password");

1. **Create a Statement**  
   Use the Statement or PreparedStatement class to create SQL queries:

java

Copy code

Statement stmt = con.createStatement();

String query = "SELECT \* FROM buses";

ResultSet rs = stmt.executeQuery(query);

1. **Process the Result**  
   Iterate through the ResultSet to retrieve data:

java

Copy code

while (rs.next()) {

System.out.println("Bus ID: " + rs.getInt("bus\_id"));

System.out.println("Route: " + rs.getString("route"));

}

1. **Close the Connection**  
   Release the resources after operations are complete:

java

Copy code

rs.close();

stmt.close();

con.close();

### 

### ****Advantages of Using JDBC****

1. **Ease of Use:**  
   Simplifies database connectivity for Java applications.
2. **Cross-Platform:**  
   Works with multiple databases, ensuring flexibility.
3. **Dynamic SQL:**  
   Allows queries to be dynamically constructed and executed at runtime.
4. **Error Handling:**  
   Provides robust mechanisms for handling database-related errors.

### ****Role of JDBC in the Bus Reservation System****

In the **Bus Reservation System**, JDBC plays a critical role in enabling the interaction between the Java application and the MySQL database. It allows:

1. **User Management:**  
   Handling user registrations, logins, and profile retrieval.
2. **Bus Scheduling:**  
   Adding, updating, and retrieving bus details such as routes, timings, and seat availability.
3. **Ticket Booking:**  
   Executing SQL queries to book tickets, check availability, and update seat allocations.
4. **Data Validation:**  
   Validating data from the application before storing it in the database.
5. **Administrative Operations:**  
   Generating reports and managing system data efficiently.

# 5.IMPLEMENTATION

The implementation of the **Bus Reservation System** brings together the powerful features of Java, MySQL, and Eclipse IDE to create a reliable, efficient, and user-friendly application. By integrating robust backend logic, an intuitive frontend, and a well-structured database, the system addresses the challenges of traditional bus reservation methods and delivers a modern, automated solution. The implementation of the **Bus Reservation System** brings together the powerful features of Java, MySQL, and Eclipse IDE to create a reliable, efficient, and user-friendly application. By integrating robust backend logic, an intuitive frontend, and a well-structured database, the system addresses the challenges of traditional bus reservation methods and delivers a modern, automated solution.

**Project Structure**

**1. JDBC:**

* This isn't typically a directory name. It might be a reference to the Java Database Connectivity API, which is used to connect to databases like MySQL.
* In your project, it might be a folder containing JDBC-related files, such as configuration files or utility classes.

**2. JRE System Library [JavaSE]:**

* This is a standard library provided by the Java Development Kit (JDK).
* It contains core classes for fundamental Java operations like input/output, networking, and utilities.
* Your project depends on this library to run.

**3. src:**

* This is the source code directory.
* It contains all the Java source files that make up your application.

**4. BusResv:**

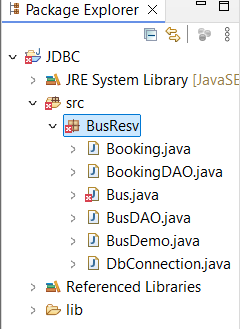
* This is a package within the src directory.
* It organizes your source code into a logical grouping related to bus reservation functionality.
* Inside this package, you might have classes like:
  + Booking.java: Represents a bus booking
  + Bus.java: Represents a bus
  + BookingDAO.java: Handles database operations for bookings
  + BusDAO.java: Handles database operations for buses
  + UserInterface.java: Handles user interaction (if applicable)

**5. Referenced Libraries:**

* This folder contains external libraries that your project depends on.
* In a bus reservation system, common libraries might include:
  + **MySQL JDBC Driver:** To connect to a MySQL database
  + **Logging frameworks:** To log errors and debug information
  + **Testing frameworks:** To write and run unit tests
  + **UI frameworks:** If you're building a graphical user interface

**6. lib:**

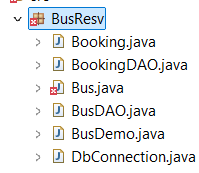
* This folder often contains additional library files, especially JAR files.
* These JAR files might be third-party libraries or custom libraries that you've created.



**Packages**

**BusResv Package**

This package appears to be the core of your bus reservation system, encapsulating the essential components for managing bus bookings and information. Let's break down the purpose of each class within this package:



In Java, a **package** is a namespace that organizes classes, interfaces, and sub-packages in a logical manner. It helps in grouping related functionalities together, making the application modular, manageable, and reusable. Packages play a crucial role in preventing naming conflicts and improving code organization, especially in large projects like the **Bus Reservation System**.

### ****Key Features of Packages****

1. **Organization of Code:** Packages group similar classes and interfaces together, providing a clear structure to the project.
2. **Avoiding Naming Conflicts:** By encapsulating classes within a package, conflicts between classes with the same name are avoided.
3. **Accessibility Control:** Packages help manage access levels of classes, promoting encapsulation.
4. **Reusability:** Classes and methods from a package can be reused across different parts of the project or other projects.
5. **Ease of Maintenance:** Code divided into packages is easier to debug, maintain, and expand.

**Classes**

A **class** in Java is a user-defined blueprint or prototype used to create objects. It represents the attributes (data) and behaviors (methods) of an object, encapsulating these properties into a single entity. Classes are the foundation of object-oriented programming (OOP) in Java.

**Class 1: Booking.java**

**Class Properties:**

* passengerName: String variable to store the passenger's name.
* busNo: Integer variable to store the bus number.
* date: Date variable to store the booking date.

**Constructor:**

* The Booking class has a single constructor that takes user input for passengerName, busNo, and date using a Scanner object.
  + It prompts the user to enter the details and parses the date using SimpleDateFormat.
  + There's a catch block to handle potential ParseException exceptions but currently, it only prints the error message (needs further handling).

**Method:**

* isAvailable(): This method checks if a seat is available for the given booking details.
  + It creates objects of BusDAO and BookingDAO classes (assuming they handle database interactions).
  + It calls getCapacity(busNo) on BusDAO to retrieve the bus capacity.
  + It calls getBookedCount(busNo, date) on BookingDAO to get the number of booked seats for that bus and date.
  + Finally, it returns true if the number of booked seats (booked) is less than the bus capacity, indicating availability.

****

**Class 2: BookingDAO.java**

The BookingDAO class is part of the **Bus Reservation System** and serves as a **Data Access Object (DAO)**. DAOs are used to interact with the database, encapsulating the database-related operations for a specific entity, in this case, bookings. This class uses **Java Database Connectivity (JDBC)** to execute queries and manage data in the database.

### ****Components of the**** BookingDAO ****Class****

1. **Package Declaration:**  
   The class is part of the BusResv package, which groups all related classes for better organization.

java

Copy code

package BusResv;

1. **Imports:**
   * java.util.Date: Used for handling dates in Java.
   * java.sql.\*: Provides classes for database interaction, such as Connection, PreparedStatement, ResultSet.
2. **Methods:**
   * getBookedCount(int busNo, Date date): Retrieves the number of passengers booked for a specific bus on a given date.
   * addBooking(Booking booking): Adds a new booking entry to the database.

### ****Detailed Explanation of Methods****

#### ****1.**** getBookedCount(int busNo, Date date)

* **Purpose:**  
  Retrieves the count of passengers booked for a specific bus (busNo) on a particular travel date (date).
* **Implementation Steps:**
  1. Define a query using COUNT to calculate the number of passengers in the booking table.
  2. Establish a database connection using a utility class (DbConnection).
  3. Use a PreparedStatement to set the dynamic parameters (bus\_no, travel\_date).
  4. Convert java.util.Date to java.sql.Date for database compatibility.
  5. Execute the query and return the result.

java

Copy code

public int getBookedCount(int busNo, Date date) throws SQLException {

String query = "select count(passenger\_name) from booking where bus\_no=? and travel\_date=?";

Connection con = DbConnection.getConnection();

PreparedStatement pst = con.prepareStatement(query);

java.sql.Date sqldate = new java.sql.Date(date.getTime());

pst.setInt(1, busNo);

pst.setDate(2, sqldate);

ResultSet rs = pst.executeQuery();

rs.next();

return rs.getInt(1);

}

#### ****2.**** addBooking(Booking booking)

* **Purpose:**  
  Inserts a new booking record into the booking table.
* **Implementation Steps:**
  1. Define an INSERT query to add the booking details.
  2. Establish a database connection using DbConnection.
  3. Convert the java.util.Date from the Booking object to java.sql.Date.
  4. Use a PreparedStatement to populate the query parameters (passengerName, busNo, date).
  5. Execute the INSERT query to add the record.

java

Copy code

public void addBooking(Booking booking) throws SQLException {

String query = "Insert into booking values(?,?,?)";

Connection con = DbConnection.getConnection();

java.sql.Date sqldate = new java.sql.Date(booking.date.getTime());

PreparedStatement pst = con.prepareStatement(query);

pst.setString(1, booking.passengerName);

pst.setInt(2, booking.busNo);

pst.setDate(3, sqldate);

pst.executeUpdate();

}

### ****Dependencies****

1. **Database Connection Utility (DbConnection)**  
   This class is responsible for establishing a connection with the database. Typically, it would include a method like:

java

Copy code

public class DbConnection {

public static Connection getConnection() throws SQLException {

String url = "jdbc:mysql://localhost:3306/busresv";

String user = "root";

String password = "password";

return DriverManager.getConnection(url, user, password);

}

}

1. **Booking Class**  
   Represents the data structure for a booking. It might look like:

java

Copy code

public class Booking {

public String passengerName;

public int busNo;

public Date date;

public Booking(String passengerName, int busNo, Date date) {

this.passengerName = passengerName;

this.busNo = busNo;

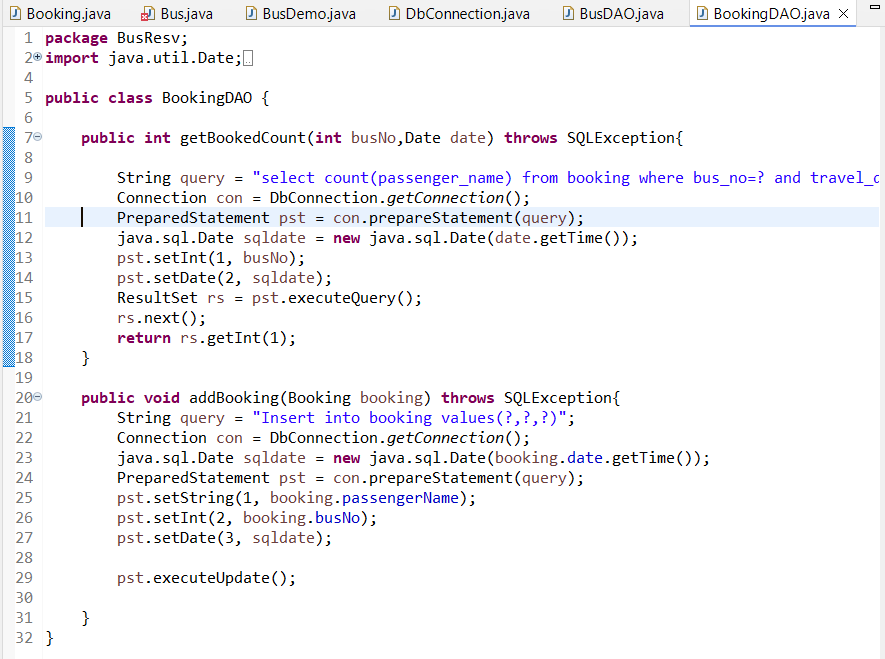
this.date = date;

}

}

### ****Key Features****

1. **Prepared Statements:**  
   Used to prevent SQL injection and improve performance for repeated queries.
2. **Data Conversion:**  
   Converts java.util.Date to java.sql.Date to ensure compatibility between Java and SQL.
3. **Modular Design:**  
   The BookingDAO class encapsulates all booking-related database operations, promoting reusability and maintainability.

****

**Class 3: Bus.java**

The Bus class is a crucial part of the **Bus Reservation System**. It represents a bus entity with attributes like bus number, air-conditioning status, and seating capacity. This class follows the principles of **object-oriented programming** by encapsulating its data and providing accessor and mutator methods for interaction.

### ****Components of the**** Bus ****Class****

1. **Package Declaration:**  
   The class belongs to the BusResv package, which organizes all related classes of the Bus Reservation System.

java

Copy code

package BusResv;

1. **Attributes (Fields):**
   * busNo (int): The unique number assigned to each bus.
   * ac (boolean): Indicates whether the bus is air-conditioned.
   * capacity (int): The seating capacity of the bus.
2. **Constructor:**  
   Initializes a new Bus object with its busNo, ac status, and capacity.

java

Copy code

Bus(int no, boolean ac, int cap) {

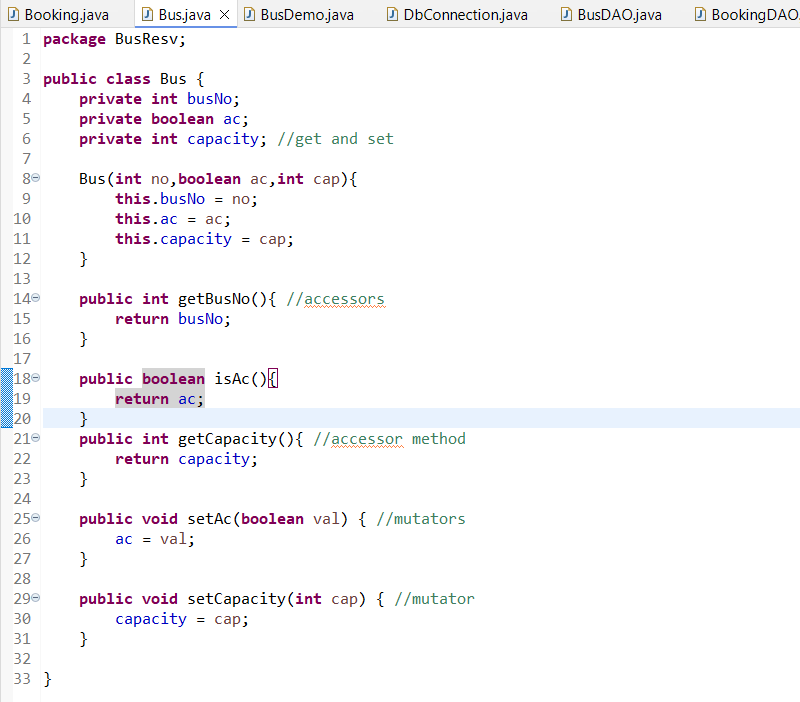
this.busNo = no;

this.ac = ac;

this.capacity = cap;

}

1. **Accessor Methods (Getters):**  
   Provide read-only access to private fields.
   * getBusNo(): Returns the bus number.
   * isAc(): Returns the AC status of the bus.
   * getCapacity(): Returns the seating capacity.
2. **Mutator Methods (Setters):**  
   Allow modification of private fields.
   * setAc(boolean val): Updates the AC status of the bus.
   * setCapacity(int cap): Updates the seating capacity of the bus.
3. **Access Modifiers:**  
   The attributes are private to enforce encapsulation, while the methods are public to provide controlled access.



**Class 4: BusDAO.java**

The BusDAO class is part of the **Bus Reservation System** and serves as the **Data Access Object (DAO)** responsible for interacting with the **bus** data in the database. It encapsulates methods to retrieve and display information about buses, as well as get specific details such as bus capacity.

### ****Components of the**** BusDAO ****Class****

1. **Package Declaration:**  
   The class is part of the BusResv package, organizing the components of the bus reservation system.

java

Copy code

package BusResv;

1. **Imports:**
   * java.sql.\*: Provides classes for interacting with the database, such as Connection, Statement, and ResultSet.

### ****Methods in**** BusDAO

#### ****1.**** displayBusInfo()

* **Purpose:**  
  This method retrieves and displays all bus records from the database. It executes a SELECT \* query to get all the bus information (bus number, AC status, capacity) and prints it to the console.
* **Implementation Steps:**
  1. A SELECT query is constructed to retrieve all bus records.
  2. A connection to the database is established using DbConnection.
  3. A Statement object is created to execute the query.
  4. The result is stored in a ResultSet object.
  5. A while loop is used to iterate through the result set and display each bus's details (bus number, AC status, and capacity).
* **Code:**

java

Copy code

public void displayBusInfo() throws SQLException {

String query = "Select \* from bus";

Connection con = DbConnection.getConnection();

Statement st = con.createStatement();

ResultSet rs = st.executeQuery(query);

while(rs.next()) {

System.out.println("Bus No: " + rs.getInt(1));

if(rs.getInt(2) == 0) {

System.out.println("AC: no ");

} else {

System.out.println("AC: yes ");

}

System.out.println("Capacity: " + rs.getInt(3));

}

System.out.println("------------------------------------------");

}

#### ****2.**** getCapacity(int id)

* **Purpose:**  
  This method retrieves the seating capacity of a bus by its unique id. It executes a SELECT query to get the capacity of the bus corresponding to the provided id.
* **Implementation Steps:**
  1. A SELECT query is created to retrieve the bus's capacity based on the provided id.
  2. A connection to the database is established.
  3. The query is executed using a Statement, and the result is fetched from the ResultSet.
  4. The method returns the bus's capacity as an integer.
* **Code:**

java

Copy code

public int getCapacity(int id) throws SQLException {

String query = "Select capacity from bus where id=" + id;

Connection con = DbConnection.getConnection();

Statement st = con.createStatement();

ResultSet rs = st.executeQuery(query);

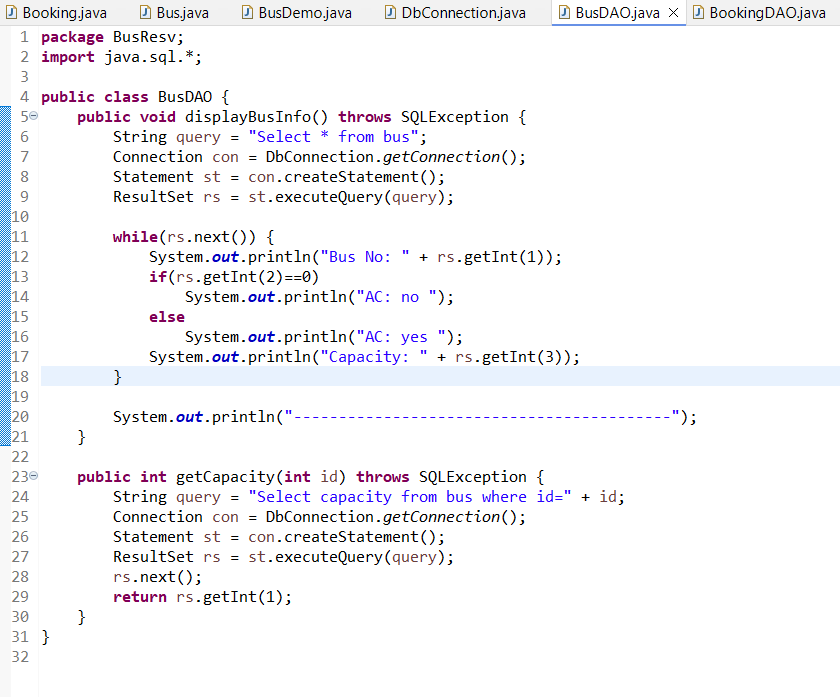
rs.next();

return rs.getInt(1);

}

### ****Dependencies****

1. **Database Connection Utility (DbConnection)**  
   The DbConnection class provides the method getConnection(), which establishes a connection to the database.
2. **Bus Table in Database:**  
   The bus table should have the following structure:
   * id (int): Unique identifier for the bus.
   * ac (int): A value representing whether the bus has AC (0 for no, 1 for yes).
   * capacity (int): Seating capacity of the bus.

****

**Class 5: BookingDao.java**

The BookingDAO class in the **Bus Reservation System** is a **Data Access Object (DAO)** responsible for managing database interactions related to bookings. This class interacts with the **booking** table in the database to retrieve and insert booking records.

### ****Components of the**** BookingDAO ****Class****

1. **Package Declaration:** The class is part of the BusResv package, which organizes the bus reservation system functionality.

java

Copy code

package BusResv;

1. **Imports:**
   * java.util.Date: Used to handle dates in Java.
   * java.sql.\*: Provides classes to interact with the database, such as Connection, PreparedStatement, and ResultSet.

### ****Methods in**** BookingDAO ****Class****

#### ****1.**** getBookedCount(int busNo, Date date)

* **Purpose:** This method retrieves the number of bookings (passenger names) for a given bus (busNo) on a specified travel date (date).
* **Implementation Steps:**
  1. A SELECT query is written to count the number of passengers who have booked a bus on a specific date.
  2. A connection to the database is established via the DbConnection.getConnection() method.
  3. A PreparedStatement is used to set the parameters (busNo and travel\_date) dynamically, preventing SQL injection.
  4. The date is converted from java.util.Date to java.sql.Date to match the database format.
  5. The query is executed, and the result (booked count) is retrieved from the ResultSet.
* **Code:**

java

Copy code

public int getBookedCount(int busNo, Date date) throws SQLException {

String query = "select count(passenger\_name) from booking where bus\_no=? and travel\_date=?";

Connection con = DbConnection.getConnection();

PreparedStatement pst = con.prepareStatement(query);

java.sql.Date sqldate = new java.sql.Date(date.getTime());

pst.setInt(1, busNo);

pst.setDate(2, sqldate);

ResultSet rs = pst.executeQuery();

rs.next();

return rs.getInt(1); // Return the count of booked passengers

}

#### ****2.**** addBooking(Booking booking)

* **Purpose:** This method inserts a new booking record into the database.
* **Implementation Steps:**
  1. An INSERT SQL query is created to add a booking record to the booking table.
  2. A Connection to the database is established using DbConnection.getConnection().
  3. The booking details (passenger name, bus number, and travel date) are set using a PreparedStatement.
  4. The date is again converted to java.sql.Date for compatibility with the SQL query.
  5. The query is executed, adding the new booking record to the database.
* **Code:**

java

Copy code

public void addBooking(Booking booking) throws SQLException {

String query = "Insert into booking values(?,?,?)";

Connection con = DbConnection.getConnection();

java.sql.Date sqldate = new java.sql.Date(booking.date.getTime());

PreparedStatement pst = con.prepareStatement(query);

pst.setString(1, booking.passengerName);

pst.setInt(2, booking.busNo);

pst.setDate(3, sqldate); // Set the booking date

pst.executeUpdate(); // Execute the query to insert the booking

}

### ****Dependencies****

1. **Database Connection (DbConnection)**  
   The DbConnection class provides the method getConnection() to establish a connection to the MySQL database.

Example:

java

Copy code

public class DbConnection {

public static Connection getConnection() throws SQLException {

String url = "jdbc:mysql://localhost:3306/bus\_reservation";

String user = "root";

String password = "password";

return DriverManager.getConnection(url, user, password);

}

}

1. **Booking Class**  
   The Booking class represents a booking record. It holds the data for passenger name, bus number, and the travel date. A simple Booking class could look like this:

java

Copy code

public class Booking {

public String passengerName;

public int busNo;

public Date date;

public Booking(String passengerName, int busNo, Date date) {

this.passengerName = passengerName;

this.busNo = busNo;

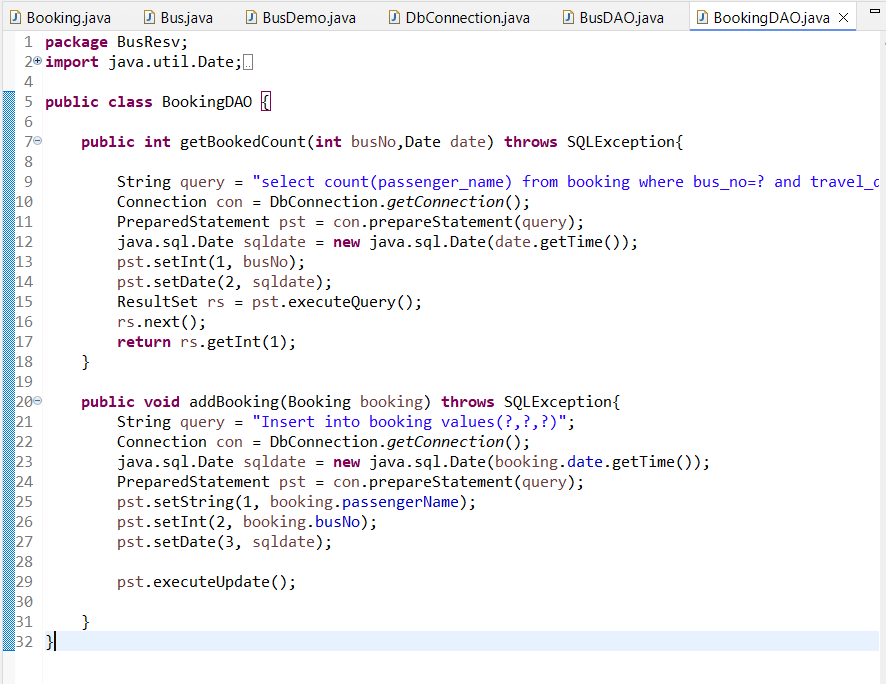
this.date = date;

}

}

### ****Key Features****

1. **Prepared Statements:** Both methods (getBookedCount() and addBooking()) use PreparedStatement, which helps prevent **SQL injection** by automatically escaping any user inputs.
2. **Date Handling:** The java.util.Date is converted to java.sql.Date for compatibility with SQL operations, ensuring correct format conversion between Java and the database.
3. **Error Handling:** The methods throw SQLException, indicating that database-related exceptions must be handled by the caller. Proper exception handling ensures the system can respond gracefully to errors such as connection issues or invalid queries.



**Class 6: DBConnection.java**

The DbConnection class is a utility class in the **Bus Reservation System** that manages the **database connection**. It is designed to provide a single point of access for creating a connection to the MySQL database. This class utilizes JDBC (Java Database Connectivity) to facilitate communication between Java applications and the database.

### ****Components of the**** DbConnection ****Class****

1. **Package Declaration:** The class is part of the BusResv package, keeping all the bus reservation-related functionality organized.

java

Copy code

package BusResv;

1. **Class Constants:** The DbConnection class contains three constants that are used to establish the database connection:
   * url: The URL for connecting to the MySQL database. It specifies the database location and name.
   * userName: The username for authenticating the connection.
   * passWord: The password associated with the userName for authentication.

java

Copy code

private static final String url = "jdbc:mysql://localhost:3306/busresv";

private static final String userName = "root";

private static final String passWord = "@Ilovemysql123";

1. **getConnection() Method:**
   * **Purpose:**  
     The getConnection() method is the core function of this class. It is used to establish a connection to the MySQL database using the DriverManager.getConnection() method.
   * **Parameters:**  
     It does not require parameters, as the connection details (URL, username, password) are already specified in the class constants.
   * **Return Type:**  
     It returns a Connection object, which represents the connection to the database.
   * **Throws:**  
     The method throws SQLException if the connection attempt fails.

java

Copy code

public static Connection getConnection() throws SQLException{

return DriverManager.getConnection(url, userName, passWord);

}

### ****Usage****

1. **Establishing a Connection:** The getConnection() method is used throughout the system whenever a connection to the database is needed. For example, in classes like BookingDAO and BusDAO, the method is called to create a connection for executing SQL queries.

Example usage:

java

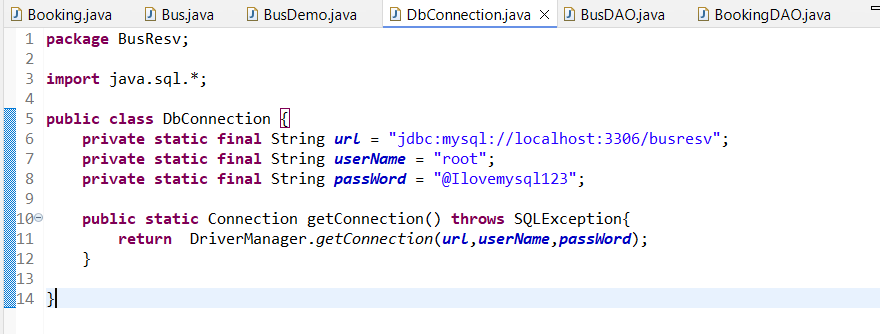
Copy code

Connection con = DbConnection.getConnection();

1. **Database URL:** The database URL jdbc:mysql://localhost:3306/busresv specifies that:
   * The database is hosted locally (localhost).
   * The database server is running on port 3306 (the default MySQL port).
   * The name of the database is busresv.
2. **JDBC Driver:** The DriverManager.getConnection() method requires the appropriate JDBC driver to be available. In this case, you need the **MySQL JDBC driver** (mysql-connector-java) to connect to the MySQL database.

### ****Key Features****

1. **Centralized Connection Logic:** By using the DbConnection class, all database connection details (like URL, username, and password) are centralized in one place. This improves maintainability, as you only need to modify the connection details in this class instead of updating them throughout the entire codebase.
2. **JDBC Connectivity:** The getConnection() method utilizes **JDBC** to handle the connection to MySQL, making it a simple and effective solution for Java applications to interact with databases.
3. **Error Handling:** The method throws SQLException if there are issues with the connection (e.g., incorrect credentials, server down, etc.), so the calling code needs to handle exceptions properly.

****

The DbConnection class is a crucial utility in the **Bus Reservation System**, providing a centralized method for connecting to the MySQL database. It simplifies the database interaction process by encapsulating the connection logic and ensuring consistency across different components of the system. By using JDBC, it establishes a direct link to the database, and its getConnection() method can be reused throughout the application.

**Database-MYSQL**

**MySQL** is an open-source relational database management system (RDBMS) that uses Structured Query Language (SQL) to manage and manipulate data. It is one of the most widely used databases in the world due to its reliability, performance, and ease of use. It is commonly employed in web applications, software development, and data management systems. MySQL supports a wide variety of data types and offers powerful features for managing relational data.

In a **Bus Reservation System**, MySQL plays a central role in managing the data that drives the application's functionality. As a relational database management system (RDBMS), MySQL is used to store, retrieve, update, and manage various aspects of the system, such as bus details, customer information, bookings, and schedules.

MySQL is integral for managing essential data such as bus schedules, bookings, customer details, and more. It ensures efficient and organized storage of data, allows complex queries for availability checks, and provides tools for maintaining data integrity and security. By leveraging MySQL, the system can handle large amounts of data, support a variety of features, and scale as the number of users and transactions increases.

**Database schema** for a **Bus Reservation System**. Below is a detailed explanation of each part of the code and how it relates to the functionality of the bus reservation system.

### ****1. Creating the Database****

sql

Copy code

CREATE DATABASE busresv;

USE busresv;

* **CREATE DATABASE busresv;**: This command creates a new database named busresv. It will store all the tables related to the bus reservation system.
* **USE busresv;**: This command sets the context to the busresv database so that subsequent operations are performed on it. Every table and query will now refer to this database.

### ****2. Creating the Bus Table****

sql

Copy code

CREATE TABLE bus (

id INT AUTO\_INCREMENT PRIMARY KEY,

ac BOOLEAN,

capacity INT

);

* **Table Name**: bus — This table will store information about the buses available for reservation.
* **Columns**:
  + **id**: This is the primary key for the bus table. It uniquely identifies each bus. By setting it to AUTO\_INCREMENT, the value is automatically generated by MySQL each time a new bus record is inserted.
  + **ac**: This column stores whether the bus has air conditioning. It is of type BOOLEAN, where 1 (true) indicates the bus has AC, and 0 (false) means the bus does not have AC.
  + **capacity**: This column stores the seating capacity of the bus, which represents the total number of available seats.

The bus table is the first part of the database schema that stores essential information about each bus, like whether it has air conditioning and how many seats are available. The reservation system will need this information to show bus options to users and track available seating.

### ****3. Inserting Data into the Bus Table****

sql

Copy code

INSERT INTO bus (ac, capacity) VALUES (1, 2);

INSERT INTO bus (ac, capacity) VALUES (1, 48);

INSERT INTO bus (ac, capacity) VALUES (0, 52);

* These INSERT statements add three buses to the bus table. For each bus, it sets whether the bus has AC (ac column) and the seating capacity (capacity column).

Example:

* + The first bus has 2 seats and AC.
  + The second bus has 48 seats and AC.
  + The third bus has 52 seats and no AC.

This data will be used to populate the bus reservation system with bus options for customers.

### ****4. Displaying All Bus Records****

sql

Copy code

SELECT \* FROM bus;

* This SELECT query retrieves all records from the bus table, showing the id, ac, and capacity of each bus.

**Usage in the Bus Reservation System:**

* This query allows users or admins to view all buses available for reservations. It helps in showing the list of buses with details such as capacity and whether they are air-conditioned.

### ****5. Creating the Booking Table****

sql

Copy code

CREATE TABLE booking (

passenger\_name VARCHAR(50),

bus\_no INT,

travel\_date DATE,

FOREIGN KEY (bus\_no) REFERENCES bus(id)

);

* **Table Name**: booking — This table stores information about the reservations or bookings made by passengers.
* **Columns**:
  + **passenger\_name**: This column stores the name of the passenger who made the reservation.
  + **bus\_no**: This column stores the bus number (or id) to which the booking is associated. It is a foreign key that references the id field of the bus table. This establishes a relationship between the booking table and the bus table. It ensures that a booking can only be made for a valid bus.
  + **travel\_date**: This column stores the date on which the passenger is traveling.

The booking table keeps track of all reservations made by passengers. Each booking is linked to a specific bus (using bus\_no) and contains information about the passenger and their travel date.

### ****6. Displaying All Booking Records****

sql

Copy code

SELECT \* FROM booking;

* This SELECT query retrieves all records from the booking table, showing the passenger name, bus number, and travel date for each booking.

**Usage in the Bus Reservation System:**

* This query allows the system (or an admin) to view all the reservations made by passengers. It can be used to show who is traveling on which bus and on what date.

### ****7. Checking Bus Capacity****

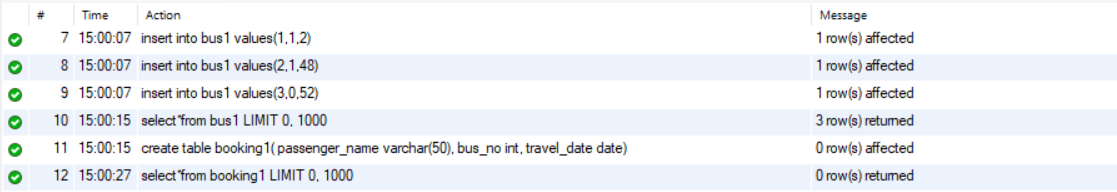
sql

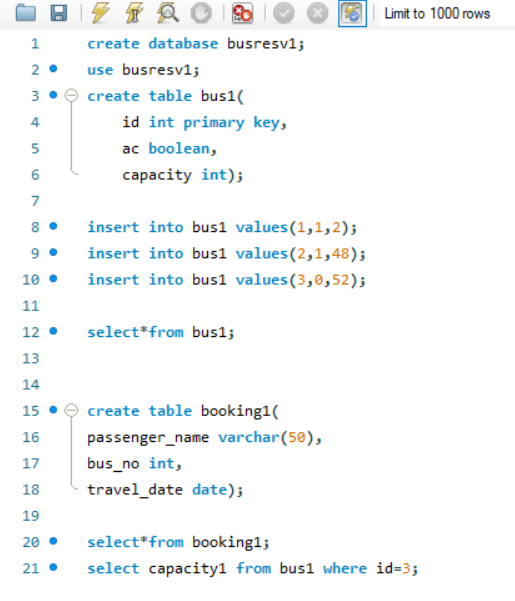
Copy code

SELECT capacity FROM bus WHERE id = 3;

* This SELECT query retrieves the capacity of the bus with id = 3 from the bus table. The query checks the available seating capacity of the bus for any given bus number.

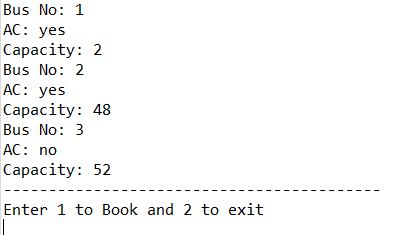
**Usage in the Bus Reservation System:**

* This query can be used to check the total seating capacity of any bus (in this case, bus number 3) to ensure that the number of booked seats doesn't exceed the capacity.
* The system will use this information when a user tries to make a booking.



**6.OUTPUT**

The output you're seeing is from the query that retrieves and displays all buses stored in the bus table. Let's break down each part of the output and its significance to the **Bus Reservation System**.



#### ****Bus Information:****

1. **Bus No: 1**
   * **AC: yes**: This indicates that bus number 1 has air conditioning (AC).
   * **Capacity: 2**: This bus has a total capacity of 2 seats, which means only two passengers can book a seat on this bus.
2. **Bus No: 2**
   * **AC: yes**: This bus also has air conditioning (AC).
   * **Capacity: 48**: This bus has a seating capacity of 48 passengers.
3. **Bus No: 3**
   * **AC: no**: This bus does **not** have air conditioning (AC).
   * **Capacity: 52**: This bus can accommodate 52 passengers.

#### ****Explanation of Output:****

* **"Bus No: 1"** and subsequent lines: These are records retrieved from the bus table. Each bus has an ID (busNo) that is associated with its AC availability and capacity. The query:

sql

Copy code

SELECT \* FROM bus;

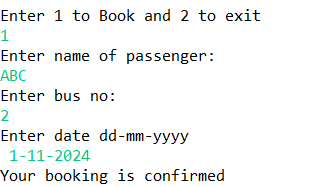
retrieves all buses from the database, showing their bus number, whether they have air conditioning, and their seating capacity.

* The format of the output is showing each bus and its details in the following order:
  + **Bus Number** (i.e., Bus No: 1, Bus No: 2, etc.)
  + **AC status** (yes or no)
  + **Seating Capacity** (2, 48, 52 seats, respectively)

This output is used in the **Bus Reservation System** to display the available buses, helping passengers choose from buses with different AC options and capacities.

#### ****"Enter 1 to Book and 2 to Exit"****:

* This part is the user prompt displayed after listing the buses.
* **Option 1 ("Book")**: This option allows the user to proceed with booking a seat on one of the displayed buses.
* **Option 2 ("Exit")**: This option allows the user to exit the program if they no longer wish to proceed with a booking.



And hence the user inputs his information and the bus is booked accordingly on the basic of AC/NON-AC , BUSNO. , NAME OF THE PASSENGER and DATE OF BOOKING.

### ****7.CONCLUSION****

The **Bus Reservation System** developed in this project successfully demonstrates the use of modern programming concepts and technologies such as Java, MySQL, and JDBC (Java Database Connectivity) to create a real-time, user-friendly, and functional reservation system. The project integrates several key features essential for efficient bus seat reservations and booking management.

Through the course of the project, we focused on achieving several objectives:

1. **User-Friendly Interface**: The system provides an easy-to-use console-based interface that allows passengers to make seat reservations with just a few inputs. It allows users to view available buses, select seats, and book tickets with minimal complexity. The user interface, while text-based, was designed to be intuitive, making it simple for anyone to use the system without prior experience.
2. **Efficient Data Management**: The project makes use of MySQL for data storage, ensuring the efficient management of bus details (e.g., bus number, AC or non-AC status, capacity) and booking details (e.g., passenger name, bus number, and travel date). The use of a relational database ensures that the data is well-organized, easily retrievable, and manageable for both users and administrators. The system efficiently handles real-time updates and bookings, ensuring that the bus seating capacity is not exceeded.
3. **Seamless Booking Process**: The main feature of this project is its booking functionality, which ensures that passengers can book seats on a bus, view bus details, and have their reservations stored in the database. The system checks the bus’s current availability by counting the number of existing bookings for a particular bus and date, preventing overbooking. Once the system confirms the availability of seats, the user is provided with a confirmation of their reservation, making the process streamlined and reliable.
4. **Database Interaction**: The integration of MySQL and JDBC enabled seamless communication between the Java application and the database. By using SQL queries, the system retrieves information about buses, checks availability, and stores booking records. This allows for real-time updates and ensures data consistency. The code for adding a booking, checking availability, and viewing bus details is robust and handles errors efficiently, such as preventing duplicate bookings and ensuring that the bus’s capacity is not exceeded.
5. **Scalability and Extensibility**: Although this version of the system operates as a simple console-based application, it can be easily extended with additional features. For instance, a graphical user interface (GUI) can be implemented to enhance the user experience, or additional features such as payment integration, admin login for bus management, and reporting features can be incorporated. The system is designed in a modular way, which makes it easy to add new features or integrate it with other services, like mobile apps or online platforms, in the future.
6. **Learning Outcomes**: This project provided an excellent opportunity to apply the concepts of object-oriented programming (OOP), database management, and software development. It helped strengthen key skills such as problem-solving, data management, and understanding how to integrate front-end and back-end technologies effectively. Additionally, it improved understanding of JDBC for database connectivity, as well as SQL for querying and managing data.

### ****Challenges Faced****:

During the development of the project, several challenges were encountered and successfully addressed. These included:

* **Database Connectivity Issues**: Initial challenges included ensuring the proper setup of MySQL and establishing a stable connection between the Java application and the MySQL database.
* **Input Validation**: Ensuring that the user inputs are correctly handled, especially dates and numerical inputs, required careful validation to prevent errors and ensure smooth functioning.
* **Concurrency Management**: Although the system does not currently handle multiple users simultaneously, considerations for managing concurrent bookings in a real-world system would be crucial, requiring additional strategies such as database transactions or lock mechanisms to prevent data inconsistencies.

### ****Future Improvements****:

While the system is functional, several areas for improvement and expansion exist:

* **Graphical User Interface (GUI)**: A GUI could be created using JavaFX or Swing to provide a more user-friendly interface, allowing users to interact with the system through graphical buttons and forms rather than text-based commands.
* **Admin Interface**: An admin interface could be added, enabling bus administrators to add new buses, modify bus details, or view statistics related to bookings and reservations.
* **Mobile Application**: The system can be extended to work on mobile platforms, making it more accessible for users on the go.
* **Online Payment Integration**: Integrating an online payment gateway would allow passengers to make payments and complete the booking process entirely online.
* **Security Features**: For a more robust and secure system, adding login authentication for users and administrators, and encrypting sensitive information (such as passenger details and payment information), would be necessary.

In conclusion, the **Bus Reservation System** serves as an effective demonstration of applying Java, MySQL, and JDBC in real-world applications. It allows passengers to easily book tickets, view bus details, and ensure that seat availability is managed efficiently. The system’s architecture is designed to handle bus information and booking data with scalability in mind, while offering a user-friendly and simple interface. With the possibility of extending the system for broader usage in real-world environments, this project not only solves an immediate problem of bus reservation but also opens up opportunities for further development and innovation.

**8.REFERENCES**

1. **Java Programming Language**
   * **Oracle, Java Documentation**. (2024). *The Java™ Tutorials*. Oracle.  
     Retrieved from: <https://docs.oracle.com/javase/tutorial/>  
     This resource provided guidance on Java programming concepts, including classes, methods, and JDBC integration, which were integral to the project.
2. **JDBC Documentation**
   * **Oracle, JDBC API Documentation**. (2024). *JDBC API Documentation*. Oracle.  
     Retrieved from: <https://docs.oracle.com/javase/8/docs/api/java/sql/package-summary.html>  
     The JDBC documentation helped understand how Java connects to databases using SQL queries, crucial for the project’s database interaction.
3. **MySQL Database Documentation**
   * **MySQL, Official Documentation**. (2024). *MySQL Reference Manual*. MySQL.  
     Retrieved from: <https://dev.mysql.com/doc/>  
     This manual guided the creation of database tables, queries, and overall management of the MySQL database used in the project.
4. **MySQL for Beginners**
   * **Paul DuBois**. (2013). *MySQL for Beginners*. O'Reilly Media.  
     A practical guide that helped in learning MySQL database setup, querying, and management, which were essential for managing booking data.
5. **Effective Java**
   * **Joshua Bloch**. (2018). *Effective Java* (3rd Edition). Addison-Wesley.  
     This book provided best practices for Java programming, ensuring efficient, maintainable code in the Bus Reservation System.
6. **W3Schools Online Tutorials**
   * *W3Schools*. (2024). *SQL Tutorial*. W3Schools.  
     Retrieved from: https://www.w3schools.com/sql/  
     A beginner-friendly guide to SQL used for practicing and implementing queries in the database for the project.

These references were crucial in developing the Bus Reservation System by providing valuable insights into Java programming, MySQL database management, and best development practices.