21YE04 - ADVANCED JAVA PROGRAMMING

KARPAGAM
COLLEGE OF ENGINEERING
Rediscover | Refine | Redefine
(Autonomous)

ASSIGNMENT 1

SEMSTER IV

2024

RAM PRAKASH K - 717822Y141

Mohamed Aakil – 717822Y131

COMPUTER SCIENCE AND ENGINEERING (CYBER SECURITY)

Abstract:

The Hotel Booking System is a Java-based software application designed to facilitate the management of room bookings, guest details, and reservation schedules for hotels and similar establishments. Built using JDBC for database connectivity, the system follows a layered architecture pattern, comprising presentation, business logic, data access, and database layers. Through a command-line interface (CLI), hotel staff can interact with the system to perform various tasks, including adding rooms, adding guests, making bookings, processing payments, and viewing room and guest information. The system's modular design ensures scalability, maintainability, and extensibility, while its intuitive user interface enhances usability and user experience. By automating and centralizing hotel management processes, the Hotel Booking System aims to improve operational efficiency, streamline workflow, and enhance guest satisfaction.

Table of Content:

Introduction

The Hotel Booking System is a comprehensive software solution designed to streamline the management of room bookings, guest details, and reservation schedules for hotels and similar establishments. In today's fast-paced hospitality industry, efficient management of hotel operations is paramount to ensuring excellent guest experiences and maximizing revenue. Traditional paper-based systems or manual processes for handling bookings and guest information often lead to inefficiencies, errors, and delays. Therefore, there arises a need for a sophisticated digital solution that automates and centralizes the management of hotel bookings and guest data.

Objectives:

Efficient Room Management:

- Develop functionalities to add, update, and delete room information, including details such as room number, type, price, and status.

Seamless Payment Processing:

- Develop functionalities for processing payments for bookings, recording payment details such as amount, date, and method.

Data Security and Integrity:

- Implement robust security measures to protect sensitive guest information and payment data from unauthorized access or breaches.

System Analysis:

Requirements Analysis:

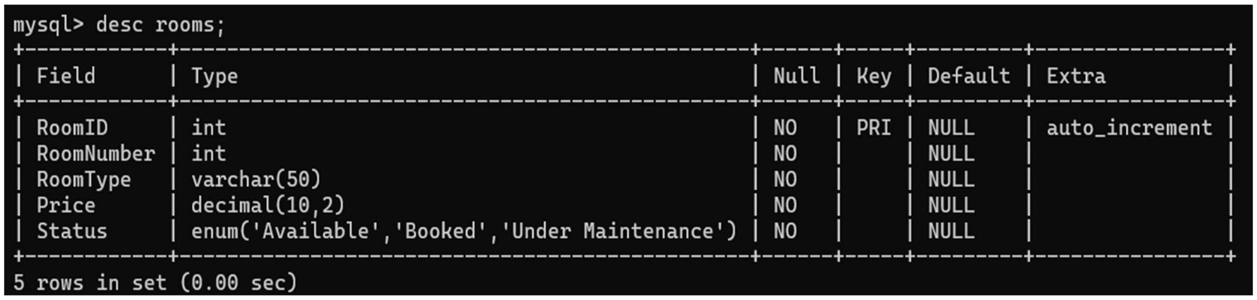
The Hotel Booking System project aims to develop a comprehensive software solution that streamlines the management of room bookings, guest details, and reservation schedules for hotels and similar establishments. Key requirements include functionalities for efficient room management, streamlined guest management, effective booking management, seamless payment processing, insightful reporting and analysis, user-friendly interface, data security and integrity, and scalability and adaptability.

System Specification:

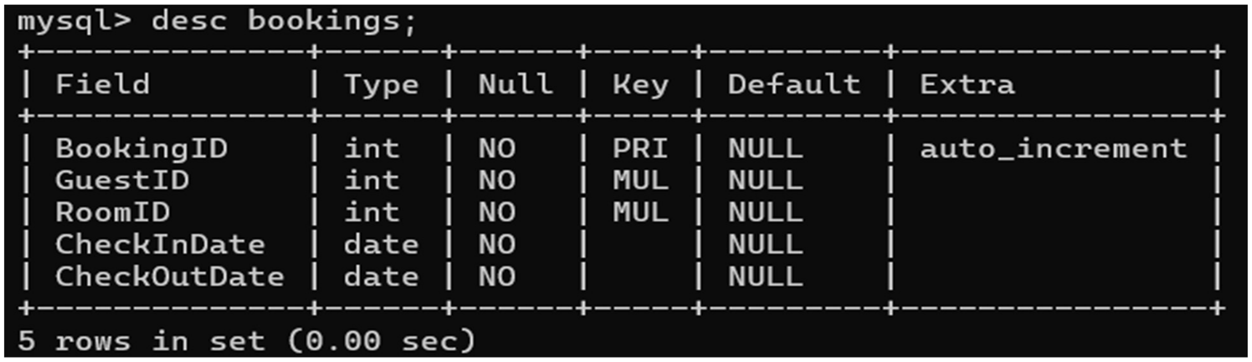
- ✓ **Memory : 4 gb ram.**
- ✓ **Processor : i3 onwards.**
- ✓ **Storage : 10 gb storage.**

System Design:

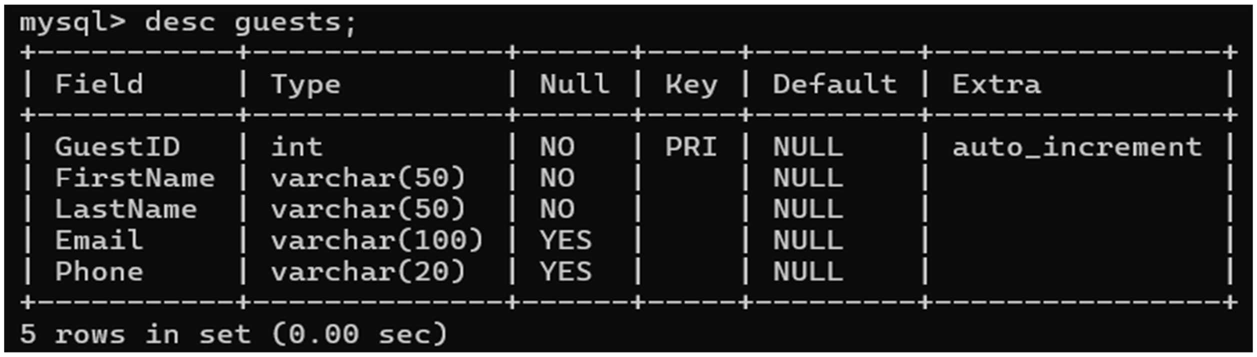
Database Schema Design:



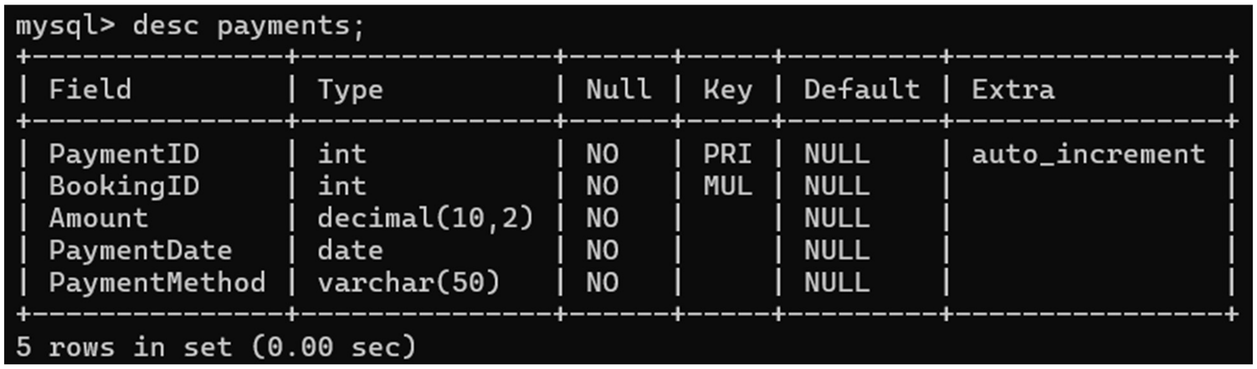
```
mysql> desc rooms;
+------------+-----------------------------------------------+------+-----+---------+----------------+
| Field      | Type                                          | Null | Key | Default | Extra          |
+------------+-----------------------------------------------+------+-----+---------+----------------+
| RoomID     | int                                           | NO   | PRI | NULL    | auto_increment |
| RoomNumber | int                                           | NO   |     | NULL    |                |
| RoomType   | varchar(50)                                   | NO   |     | NULL    |                |
| Price      | decimal(10,2)                                 | NO   |     | NULL    |                |
| Status     | enum('Available','Booked','Under Maintenance') | NO   |     | NULL    |                |
+------------+-----------------------------------------------+------+-----+---------+----------------+
5 rows in set (0.00 sec)
```



```
mysql> desc bookings;
+--------------+------+------+-----+---------+----------------+
| Field        | Type | Null | Key | Default | Extra          |
+--------------+------+------+-----+---------+----------------+
| BookingID    | int  | NO   | PRI | NULL    | auto_increment |
| GuestID      | int  | NO   | MUL | NULL    |                |
| RoomID       | int  | NO   | MUL | NULL    |                |
| CheckInDate  | date | NO   |     | NULL    |                |
| CheckOutDate | date | NO   |     | NULL    |                |
+--------------+------+------+-----+---------+----------------+
5 rows in set (0.00 sec)
```



```
mysql> desc guests;
+-----------+--------------+------+-----+---------+----------------+
| Field     | Type         | Null | Key | Default | Extra          |
+-----------+--------------+------+-----+---------+----------------+
| GuestID   | int          | NO   | PRI | NULL    | auto_increment |
| FirstName | varchar(50)  | NO   |     | NULL    |                |
| LastName  | varchar(50)  | NO   |     | NULL    |                |
| Email     | varchar(100) | YES  |     | NULL    |                |
| Phone     | varchar(20)  | YES  |     | NULL    |                |
+-----------+--------------+------+-----+---------+----------------+
5 rows in set (0.00 sec)
```



```
mysql> desc payments;
+---------------+---------------+------+-----+---------+----------------+
| Field         | Type          | Null | Key | Default | Extra          |
+---------------+---------------+------+-----+---------+----------------+
| PaymentID     | int           | NO   | PRI | NULL    | auto_increment |
| BookingID     | int           | NO   | MUL | NULL    |                |
| Amount        | decimal(10,2) | NO   |     | NULL    |                |
| PaymentDate   | date          | NO   |     | NULL    |                |
| PaymentMethod | varchar(50)   | NO   |     | NULL    |                |
+---------------+---------------+------+-----+---------+----------------+
5 rows in set (0.00 sec)
```

Application Architecture:

The Hotel Booking System application architecture follows a layered architecture pattern, which is a commonly used architectural style for designing software applications. The layered architecture divides the application into logical layers, each responsible for specific tasks, promoting modularity, maintainability, and scalability. Below is a description of the layers and their responsibilities in the Hotel Booking System:

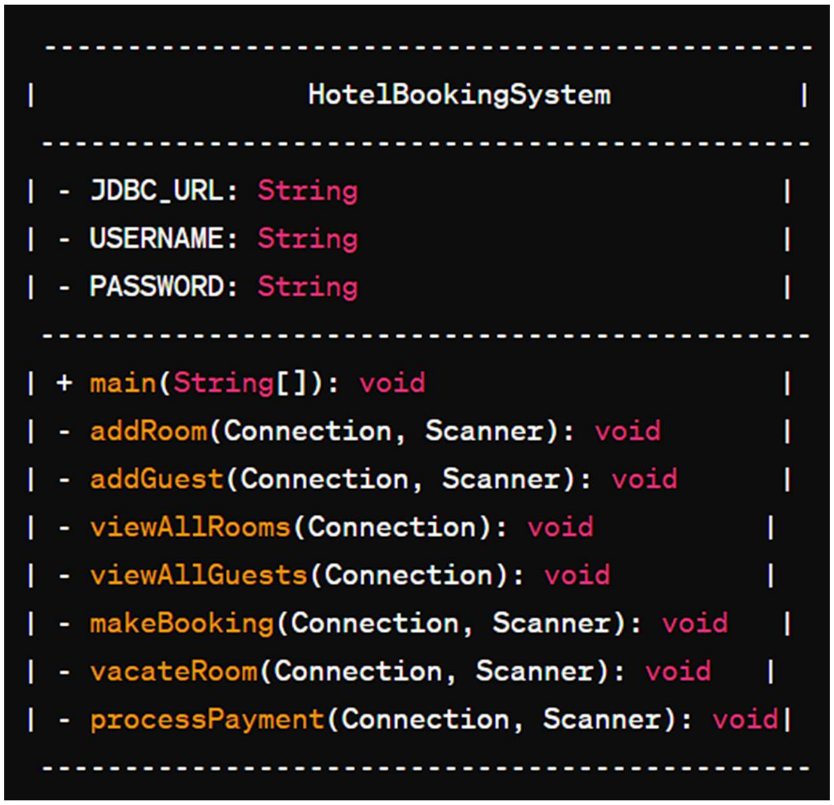
1. **Presentation Layer:**
   - Responsible for handling user interactions and presenting information to the user.
   - In the provided Java program, the presentation layer is represented by the main method in the **HotelBookingSystem** class.
   - It interacts with the user through the command-line interface (CLI) and presents menu options for various actions such as adding rooms, making bookings, etc.

2. **Business Logic Layer:**
   - Contains the core business logic of the application, including validation, processing, and orchestration of business rules.
   - Implements the use cases and business workflows of the Hotel Booking System.
   - In the provided Java program, the business logic is embedded within the methods of the **HotelBookingSystem** class, such as **addRoom**, **addGuest**, **makeBooking**, etc.
   - This layer interacts with the data access layer to perform CRUD (Create, Read, Update, Delete) operations on the database.

3. **Data Access Layer:**
   - Responsible for accessing and manipulating data stored in the database.
   - Executes SQL queries and updates to retrieve and store information in the database.
   - In the provided Java program, the data access layer is encapsulated within the methods that interact with the database, such as **addRoom**, **addGuest**, **makeBooking**, etc.
   - It uses JDBC (Java Database Connectivity) to establish connections with the MySQL database and execute SQL statements.

4. **Database Layer:**
   - Represents the physical storage of data in the MySQL database.
   - Consists of tables and relationships defined in the database schema (e.g., Rooms, Guests, Bookings, Payments).
   - Stores information related to rooms, guests, bookings, payments, and other entities relevant to the Hotel Booking System.
   - The database layer is external to the application code and managed by the MySQL database management system.

UML Diagram:


```
 -----------------------------------------
|            HotelBookingSystem           |
 -----------------------------------------
| - JDBC_URL: String                      |
| - USERNAME: String                      |
| - PASSWORD: String                      |
 -----------------------------------------
| + main(String[]): void                  |
| - addRoom(Connection, Scanner): void    |
| - addGuest(Connection, Scanner): void   |
| - viewAllRooms(Connection): void        |
| - viewAllGuests(Connection): void       |
| - makeBooking(Connection, Scanner): void|
| - vacateRoom(Connection, Scanner): void |
| - processPayment(Connection, Scanner): void|
 -----------------------------------------
```

Implementation:

Environment Setup:

1. **Database Choice:**
    - Choose a relational database management system (RDBMS) to store employee data, salaries, and payroll transactions. Common options include MySQL, PostgreSQL, Oracle Database, or SQLite. For simplicity and ease of use, MySQL is often preferred.
2. **Database Installation:**
    - Install and configure the chosen database on your local machine or a server. Follow the installation instructions provided by the database vendor.
3. **JDBC Setup:**
    - Ensure that you have the JDBC driver for your chosen database. You can download the JDBC driver from the official website of the database vendor or use a dependency management tool like Maven or Gradle to include it in your project.
    - Add the JDBC driver to your project's classpath to enable database connectivity.
4. **Database Connection Configuration:**
    - Set up the database connection parameters such as URL, username, and password. These details will be used to establish a connection to the database from your Java application.
    - Make sure the database server is running and accessible from your application environment.
5. **Schema Design:**
    - Design the database schema based on the requirements of your Employee Payroll System. Define tables, columns, primary keys, foreign keys, and any necessary constraints.
    - Ensure that the database schema aligns with the functionalities of your application, such as storing employee details, salary information, and payroll transactions.
6. **Testing Database Connectivity:**
    - Write a simple Java program to test the database connectivity using JDBC. Establish a connection to the database, execute a sample query, and verify the results.
    - This step ensures that your application can communicate with the database successfully.
7. **Additional Configurations:**
    - Depending on your project requirements, you may need to configure additional settings such as connection pooling, database authentication, and security measures.
    - Implement error handling and logging mechanisms to handle database-related exceptions gracefully and log relevant information for debugging purposes.

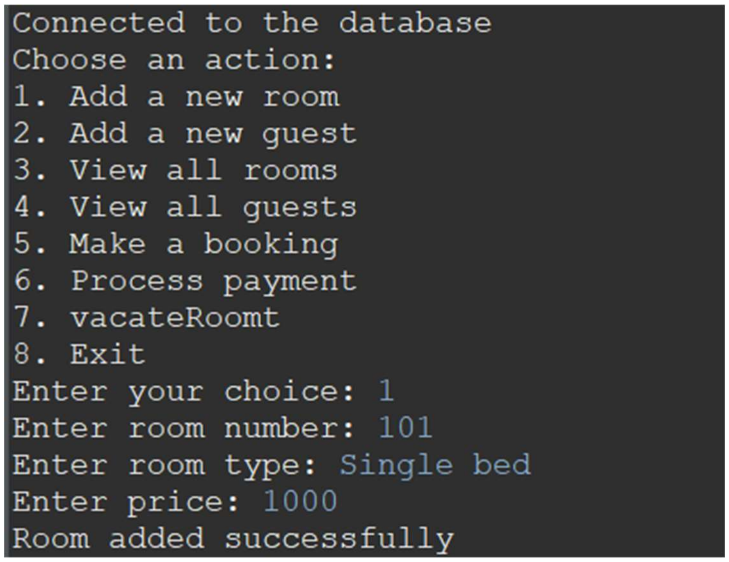
Code:

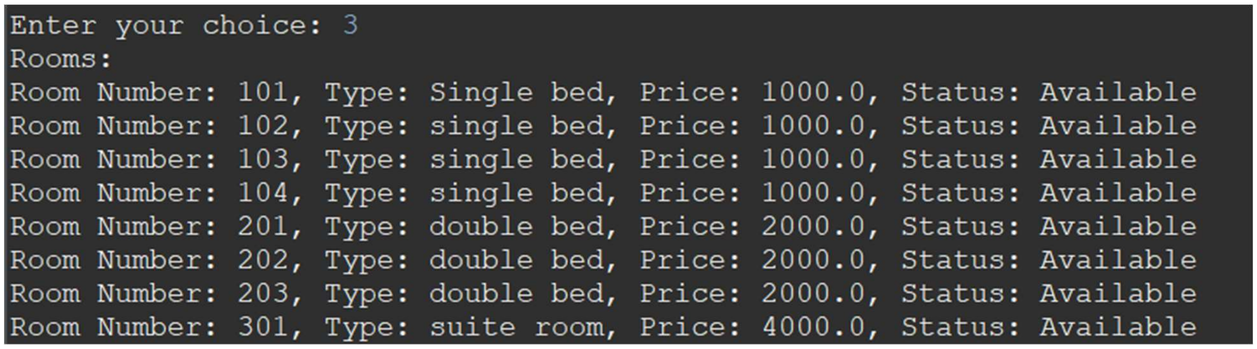
```
import java.sql.*;
public class HotelBookingSystem {
  // JDBC URL, username, and password of MySQL server
  private static final String JDBC_URL = "jdbc:mysql://localhost:3306/HBS";
  private static final String USERNAME = "root";
  private static final String PASSWORD = "1234";

  public static void main(String[] args) {
    try (Connection conn = DriverManager.getConnection(JDBC_URL, USERNAME,
PASSWORD)) {
      System.out.println("Connected to the database");

      java.util.Scanner scanner = new java.util.Scanner(System.in);
      int choice;
      do {
        System.out.println("Choose an action:");
        System.out.println("1. Add a new room");
        System.out.println("2. Add a new guest");
        System.out.println("3. View all rooms");
        System.out.println("4. View all guests");
        System.out.println("5. Make a booking");
        System.out.println("6. Process payment");
        System.out.println("7. vacateRoomt");
        System.out.println("8. Exit");
        System.out.print("Enter your choice: ");

        choice = scanner.nextInt();
        scanner.nextLine(); // Consume newline
```

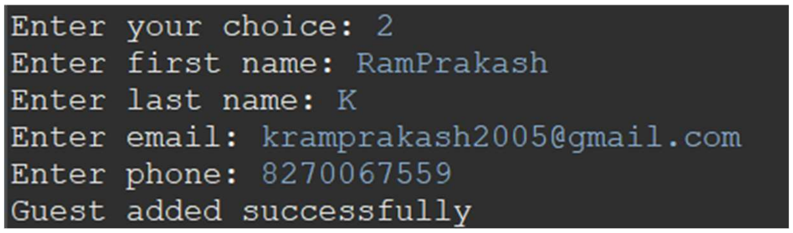
```
            switch (choice) {
                case 1:
                    addRoom(conn, scanner);
                    break;
                case 2:
                    addGuest(conn, scanner);
                    break;
                case 3:
                    viewAllRooms(conn);
                    break;
                case 4:
                    viewAllGuests(conn);
                    break;
                case 5:
                    makeBooking(conn, scanner);
                    break;
                case 6:
                    processPayment(conn, scanner);
                    break;
                case 7:
                    vacateRoom(conn, scanner);
                    break;
                case 8:
                    System.out.println("Exiting...");
                    break;
                default:
                    System.out.println("Invalid choice");
            }
        } while (choice != 7);
    } catch (SQLException e) {
            System.err.println("Error: " + e.getMessage());
        }
    }

    private static void addRoom(Connection conn, java.util.Scanner scanner) throws SQLException {
        System.out.print("Enter room number: ");
        int roomNumber = scanner.nextInt();
        scanner.nextLine(); // Consume newline
        System.out.print("Enter room type: ");
        String roomType = scanner.nextLine();
        System.out.print("Enter price: ");
        double price = scanner.nextDouble();

        String sql = "INSERT INTO Rooms (RoomNumber, RoomType, Price, Status) VALUES (?, ?, ?, ?)";
        try (PreparedStatement statement = conn.prepareStatement(sql)) {
            statement.setInt(1, roomNumber);
            statement.setString(2, roomType);
            statement.setDouble(3, price);
            statement.setString(4, "Available"); // Assuming newly added room is available
            int rowsInserted = statement.executeUpdate();
            if (rowsInserted > 0) {
                System.out.println("Room added successfully");
            }
        }
    }

    private static void addGuest(Connection conn, java.util.Scanner scanner) throws SQLException {
        System.out.print("Enter first name: ");
        String firstName = scanner.next();
        System.out.print("Enter last name: ");
        String lastName = scanner.next();
        scanner.nextLine(); // Consume newline
        System.out.print("Enter email: ");
        String email = scanner.nextLine();
        System.out.print("Enter phone: ");
        String phone = scanner.next();

        String sql = "INSERT INTO Guests (FirstName, LastName, Email, Phone) VALUES (?, ?,
?, ?)";
        try (PreparedStatement statement = conn.prepareStatement(sql)) {
            statement.setString(1, firstName);
            statement.setString(2, lastName);
            statement.setString(3, email);
            statement.setString(4, phone);
            int rowsInserted = statement.executeUpdate();
            if (rowsInserted > 0) {
                System.out.println("Guest added successfully");
            }
        }
    }

    private static void viewAllRooms(Connection conn) throws SQLException {
        String sql = "SELECT * FROM Rooms";
        try (Statement statement = conn.createStatement();
             ResultSet resultSet = statement.executeQuery(sql)) {
            System.out.println("Rooms:");
            while (resultSet.next()) {
                System.out.println("Room Number: " + resultSet.getInt("RoomNumber") +
                                   ", Type: " + resultSet.getString("RoomType") +
                                   ", Price: " + resultSet.getDouble("Price") +
                    ", Status: " + resultSet.getString("Status"));
            }
        }
    }

    private static void viewAllGuests(Connection conn) throws SQLException {
        String sql = "SELECT * FROM Guests";
        try (Statement statement = conn.createStatement();
             ResultSet resultSet = statement.executeQuery(sql)) {
            System.out.println("Guests:");
            while (resultSet.next()) {
                System.out.println("Name: " + resultSet.getString("FirstName") + " " +
resultSet.getString("LastName") +
                    ", Email: " + resultSet.getString("Email") +
                    ", Phone: " + resultSet.getString("Phone"));
            }
        }
    }

    private static void makeBooking(Connection conn, java.util.Scanner scanner) throws
SQLException {
        System.out.print("Enter guest ID: ");
        int guestID = scanner.nextInt();
        System.out.print("Enter room number: ");
        int roomNumber = scanner.nextInt();
        scanner.nextLine(); // Consume newline
        System.out.print("Enter check-in date (YYYY-MM-DD): ");
        String checkInDate = scanner.nextLine();
        System.out.print("Enter check-out date (YYYY-MM-DD): ");
        String checkOutDate = scanner.nextLine();
        String sql = "INSERT INTO Bookings (GuestID, RoomID, CheckInDate, CheckOutDate)
VALUES (?, ?, ?, ?)";
        try (PreparedStatement statement = conn.prepareStatement(sql)) {
            statement.setInt(1, guestID);
            statement.setInt(2, roomNumber);
            statement.setString(3, checkInDate);
            statement.setString(4, checkOutDate);
            int rowsInserted = statement.executeUpdate();
            if (rowsInserted > 0) {
                System.out.println("Booking made successfully");
            }
        }
    }

    private static void vacateRoom(Connection conn, java.util.Scanner scanner) throws
SQLException {
        System.out.print("Enter room number to vacate: ");
        int roomNumber = scanner.nextInt();

        String sql = "UPDATE Rooms SET Status = 'Available' WHERE RoomNumber = ?";
        try (PreparedStatement statement = conn.prepareStatement(sql)) {
            statement.setInt(1, roomNumber);
            int rowsUpdated = statement.executeUpdate();
            if (rowsUpdated > 0) {
                System.out.println("Room " + roomNumber + " has been vacated");
            } else {
                System.out.println("Room " + roomNumber + " not found or already available");
            }
        }
    }
```

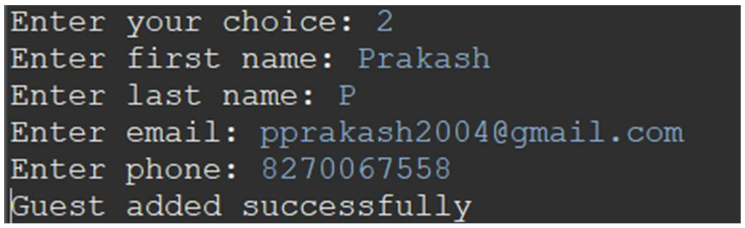
```
   private static void processPayment(Connection conn, java.util.Scanner scanner) throws SQLException {
        System.out.print("Enter booking ID: ");
        int bookingID = scanner.nextInt();
        System.out.print("Enter amount: ");
        double amount = scanner.nextDouble();
        scanner.nextLine(); // Consume newline
        System.out.print("Enter payment date (YYYY-MM-DD): ");
        String paymentDate = scanner.nextLine();
        System.out.print("Enter payment method: ");
        String paymentMethod = scanner.nextLine();

        String sql = "INSERT INTO Payments (BookingID, Amount, PaymentDate, PaymentMethod) VALUES (?, ?, ?, ?)";
        try (PreparedStatement statement = conn.prepareStatement(sql)) {
            statement.setInt(1, bookingID);
            statement.setDouble(2, amount);
            statement.setString(3, paymentDate);
            statement.setString(4, paymentMethod);
            int rowsInserted = statement.executeUpdate();
            if (rowsInserted > 0) {
                System.out.println("Payment processed successfully");
            }
        }
    }
}
```

OUTPUT:

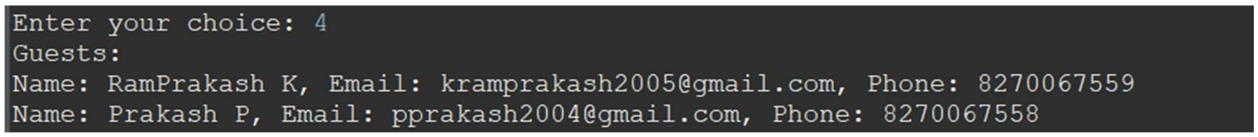

```
Connected to the database
Choose an action:
1. Add a new room
2. Add a new guest
3. View all rooms
4. View all guests
5. Make a booking
6. Process payment
7. vacateRoomt
8. Exit
Enter your choice: 1
Enter room number: 101
Enter room type: Single bed
Enter price: 1000
Room added successfully
```


```
Enter your choice: 3
Rooms:
Room Number: 101, Type: Single bed, Price: 1000.0, Status: Available
Room Number: 102, Type: single bed, Price: 1000.0, Status: Available
Room Number: 103, Type: single bed, Price: 1000.0, Status: Available
Room Number: 104, Type: single bed, Price: 1000.0, Status: Available
Room Number: 201, Type: double bed, Price: 2000.0, Status: Available
Room Number: 202, Type: double bed, Price: 2000.0, Status: Available
Room Number: 203, Type: double bed, Price: 2000.0, Status: Available
Room Number: 301, Type: suite room, Price: 4000.0, Status: Available
```

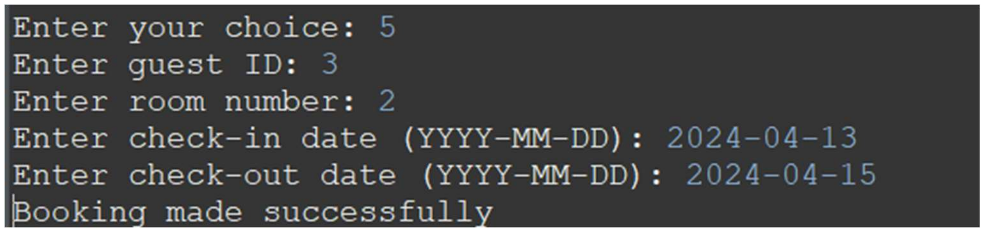

```
Enter your choice: 2
Enter first name: RamPrakash
Enter last name: K
Enter email: kramprakash2005@gmail.com
Enter phone: 8270067559
Guest added successfully
```



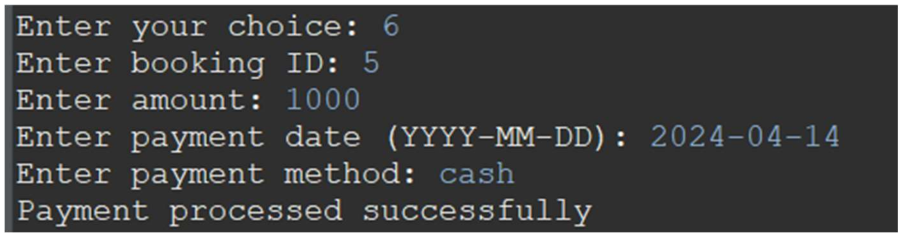
```
Enter your choice: 2
Enter first name: Prakash
Enter last name: P
Enter email: pprakash2004@gmail.com
Enter phone: 8270067558
Guest added successfully
```



```
Enter your choice: 4
Guests:
Name: RamPrakash K, Email: kramprakash2005@gmail.com, Phone: 8270067559
Name: Prakash P, Email: pprakash2004@gmail.com, Phone: 8270067558
```



```
Enter your choice: 5
Enter guest ID: 3
Enter room number: 2
Enter check-in date (YYYY-MM-DD): 2024-04-13
Enter check-out date (YYYY-MM-DD): 2024-04-15
Booking made successfully
```



```
Enter your choice: 6
Enter booking ID: 5
Enter amount: 1000
Enter payment date (YYYY-MM-DD): 2024-04-14
Enter payment method: cash
Payment processed successfully
```

References:

- **Java Database Connectivity (JDBC) Documentation:**
    - Official documentation provided by Oracle for JDBC. It covers everything from basic concepts to advanced topics related to database connectivity in Java.
    - Reference: JDBC Documentation
- **MySQL Documentation:**
    - MySQL documentation provides comprehensive guides, tutorials, and references for setting up MySQL databases, managing data, and optimizing performance.
    - Reference: MySQL Documentation