Programming Assignment 1: Car Pooling System

CENG351 Data Management and File Structures, Fall 2024-2025

Due date: 16 December 2024, Sunday, 23:59

Contents

1 Introduction **2**

2 Database Schema **2**

3 Project Structure **3**

4 Maven and Embedded H2 Database **3**

5 Tasks and Implementation Details **4**

5.1 Task 1: Create Database Tables (5 pts) 4
5.2 Task 2: Insert Data into Tables (5 pts) 4
5.3 Task 3: Find All Participants Who are Recorded as Both Drivers and Passengers (5 pts) 4
5.4 Task 4: Find Drivers with No Cars (5 pts) 4
5.5 Task 5: Delete Drivers with No Cars (5 pts) 5
5.6 Task 6: Find All Cars that have not Taken Part in Any Trips (5 pts) . . 5
5.7 Task 7: Find All Passengers Who didn't Book any Trips (5 pts) 5
5.8 Task 8: Find Trips Departing from a Given City to a Given Destination City on a Given Date (5 pts) 5
5.9 Task 9: Find Passengers with Bookings to All Trips for a City (5 pts) . . 5
5.10 Task 10: Find Driver's Cars with At Most Two Bookings (5 pts) 6
5.11 Task 11: Calculate Average Age of Passengers with Confirmed Bookings for Trips Departing from a Specified City within a Date Range (5 pts) . . 6
5.12 Task 12: Find Passengers in a Given Trip (5 pts) 6
5.13 Task 13: Find Drivers' Scores (10 pts) 6
5.14 Task 14: Find Average Ratings of Drivers Who Have Trips Destined to Each City (10 pts) 7
5.15 Task 15: Find Total Bookings by Membership Status (10 pts) 7
5.16 Task 16: Update Driver Ratings (5 pts) 7
5.17 Task 17: Drop Database Tables (5 pts) 7

6 Extra Questions **8**

6.1 Task 18: Find Trips Departing from a Given City (Optional) 8
6.2 Task 19: Find All Trips that Have Never Been Booked (Optional) 8
6.3 Task 20: Find the most booked trip(s) per driver (Optional) 8
6.4 Task 21: Find Full Cars (Optional) 9

7 Database Connection **9**

8 Statements and Prepared Statements **10**
8.1 Statement 10
8.2 PreparedStatement 10
8.3 Example Methods 10
8.3.1 Example 1: Get TripIDs of Trips Booked by "Jane Jones" 10
8.3.2 Example 2: Get TripIDs of Trips Booked by a Given Passenger . 11

9 Regulations **11**

10 Submission Instructions **12**

11 Grading **12**

1 Introduction

In this homework assignment, you are tasked with implementing a Car Pooling System using Java and SQL. The objective is to manage a database through a Java application, focusing on SQL query implementations within Java methods. You are provided with a project structure and pre-implemented classes to streamline the development process.

This assignment aims to familiarize you with database interactions in Java, Maven project management, and the use of an embedded database (H2). Even if you are not familiar with Java development environments, detailed explanations and instructions are provided to guide you through the assignment.

2 Database Schema

The Car Pooling System uses the following relational database schema:

- **Participants** (PIN, p_name, age)
- **Passengers** (PIN, membership_status) *PIN REFERENCES Participants(PIN)*
- **Drivers** (PIN, rating) *PIN REFERENCES Participants(PIN)*
- **Cars** (CarID, PIN, color, brand) *PIN REFERENCES Drivers(PIN)*
- **Trips** (TripID, CarID, date, departure, destination, num_seats_available) *CarID REFERENCES Cars(CarID)*
- **Bookings** (TripID, PIN, booking_status) *TripID REFERENCES Trips(TripID), PIN REFERENCES Passengers(PIN)*

Each table represents an entity in the system, with primary keys underlined and foreign key relationships specified.

3 Project Structure

The project is organized as follows:

```
carPoolingSystem/
  pom.xml
  src/
    main/
      java/
        ceng/
          ceng351/
            carPoolingSystem/
              Booking.java
              Car.java
              CarPoolingSystem.java
              Driver.java
              Evaluation.java
              FileOperations.java
              ICarPoolingSystem.java
              Participant.java
              Passenger.java
              QueryResult.java
              Trip.java
      resources/
        data/
          Bookings.txt
          Cars.txt
          Drivers.txt
          Participants.txt
          Passengers.txt
          Trips.txt
```

Each database table is represented by a corresponding Java class. These classes are already implemented for your convenience. Further, the classes `Evaluation`, `FileOperations`, and `QueryResult` are provided for testing purposes. You need to complete the `CarPoolingSystem.java` class. Then, running `Evaluation.java` will create and write the return values to the file `Output.txt` under the `output` directory.

4 Maven and Embedded H2 Database

The project uses Maven to manage dependencies and build configurations, simplifying the development process. Maven is a build automation tool used primarily for Java projects. The `pom.xml` file contains all the necessary information to build the project.

We utilize the embedded H2 database for managing the database without requiring additional installations. The H2 database is included as a dependency in the `pom.xml` file. This ensures that the database is ready to use without any extra setup.

We strongly recommend using an advanced IDE like IntelliJ IDEA, which is installed on the department computers (*ineks*). Opening the `pom.xml` file in IntelliJ IDEA will automatically set up the project for you.

5 Tasks and Implementation Details

You are required to implement the methods defined in the `ICarPoolingSystem` interface within the `CarPoolingSystem.java` class. Below are the tasks to be implemented:

5.1 Task 1: Create Database Tables (5 pts)

Method: `createTables()`
You will create all the tables according to the schema described in Section 2. Ensure that all **primary keys and foreign keys are properly defined**. You can assume that tables will be created before executing any other database operation.
Output: The number of tables that are created successfully.

5.2 Task 2: Insert Data into Tables (5 pts)

Methods:

- `insertParticipants(Participant[] participants)`
- `insertPassengers(Passenger[] passengers)`
- `insertDrivers(Driver[] drivers)`
- `insertCars(Car[] cars)`
- `insertTrips(Trip[] trips)`
- `insertBookings(Booking[] bookings)`

You will insert data into the appropriate tables using the provided arrays of objects.
Output: For each method, return the number of rows inserted successfully.

5.3 Task 3: Find All Participants Who are Recorded as Both Drivers and Passengers (5 pts)

Method: `getBothPassengersAndDrivers()`
The Car Pooling system allows participants to be recorded as both passengers and drivers. Your task is to find all participants who are recorded as both drivers and passengers.
Note: The result should be sorted by the `PIN` field in ascending order to ensure the smallest PIN values appear first in the list.
Output: An array of `Participants` objects containing `PIN`, `p_name`, and `age`.

5.4 Task 4: Find Drivers with No Cars (5 pts)

Method: `getDriversWithNoCars()`
You will find the PINs, names, ages, and ratings of drivers who do not own any cars in the system.
Note: The result should be sorted by the `PIN` field in ascending order to ensure the smallest PIN values appear first in the list.

Output: An array of `QueryResult.DriverPINNameAgeRating` objects containing `PIN`, `p_name`, `age`, and `rating`.

5.5 Task 5: Delete Drivers with No Cars (5 pts)

Method: `deleteDriversWithNoCars()`
You will find the drivers who do not own any cars in the system and delete these drivers from Drivers table. Note that you should not delete these drivers from Participants table.
Output: Number of rows deleted successfully.

5.6 Task 6: Find All Cars that have not Taken Part in Any Trips (5 pts)

Method: `getCarsWithNoTrips()`
You will find all cars that have not taken part in any trips.
Note: The result should be sorted by the `CarID` field in ascending order to ensure the smallest CarID values appear first in the list.
Output: An array of `Cars` objects containing `CarID`, `PIN`, `color`, and `brand`.

5.7 Task 7: Find All Passengers Who didn't Book any Trips (5 pts)

Method: `getPassengersWithNoBooks()`
You will find all passengers who didn't book any trips.
Note: The result should be sorted by the `PIN` field in ascending order to ensure the smallest PIN values appear first in the list.
Output: An array of `Passengers` objects containing `PIN`, and `membership_status`.

5.8 Task 8: Find Trips Departing from a Given City to a Given Destination City on a Given Date (5 pts)

Method: `getTripsFromToCitiesOnSpecificDate(String departure, String destination, String date)`
You will find all trips that depart from the specified city to specified destination city on specific date.
Note: The result should be sorted by the `TripID` field in ascending order to ensure the smallest TripID values appear first in the list.
Output: An array of `Trip` objects containing `TripID`, `CarID`, `date`, `departure`, `destination`, and `num_seats_available`.

5.9 Task 9: Find Passengers with Bookings to All Trips for a City (5 pts)

Method:
`getPassengersWithBookingsToAllTripsForCity(String city)`
You will find the PINs, names, ages, and membership_status of passengers who have bookings on all trips destined at a particular city.

Note: The result should be sorted by the `PIN` field in ascending order to ensure the smallest PIN values appear first in the list.
Output: An array of `QueryResult.PassengerPINNameAgeMembershipStatus` objects containing `PIN`, `p_name`, `age`, and `membership_status`.

5.10 Task 10: Find Driver's Cars with At Most Two Bookings (5 pts)

Method: `getDriverCarsWithAtMost2Bookings(int driverPIN)`
For a given driver PIN, you will find the `CarID`s that the driver owns and were booked at most twice.
Note: The result should be sorted by the `CarID` field in ascending order to ensure the smallest CarID values appear first in the list.
Output: An array of `Integer` objects containing `CarID`s.

5.11 Task 11: Calculate Average Age of Passengers with Confirmed Bookings for Trips Departing from a Specified City within a Date Range (5 pts)

Method: `getAvgAgeOfPassengersDepartFromCityBetweenTwoDates(String city, String start_date, String end_date )`
You will compute the average age of passengers with `"Confirmed"` bookings (i.e., booking_status is "Confirmed") on trips departing from a given city and within a specified date range.
Output: A `Double` value representing the average age.

5.12 Task 12: Find Passengers in a Given Trip (5 pts)

Method: `getPassengerInGivenTrip(int TripID)`
For a given TripID, you will find the passengers who have booked for that trip.
Note: The result should be sorted by the `PIN` field in ascending order to ensure the smallest PIN values appear first in the list.
Output: An array of `QueryResult.PassengerPINNameAgeMembershipStatus` objects containing `PIN`, `p_name`, `age`, and `membership_status`.

5.13 Task 13: Find Drivers' Scores (10 pts)

Method: `getDriversScores()`
For each driver, retrieve the driver's rating and the total number of bookings involving the driver's car(s) within trips. Multiply the driver's rating by the number of bookings to calculate the driver_score.
Note: The result should be sorted by the `driver_score` field in descending order to ensure the biggest driver_score values appear first in the list. In the event that multiple drivers have the same driver_score, the drivers should be further sorted in ascending order by their `DriverPIN`.
Output: An array of `QueryResult.DriverScoreRatingNumberOfBookingsPIN` objects containing `driver_score`, `rating`, `numberOfBookings`, and `DriverPIN`.

5.14 Task 14: Find Average Ratings of Drivers Who Have Trips Destined to Each City (10 pts)

Method: `getDriversAverageRatingsToEachDestinatedCity()`
For each city that has been a destination of any trip, retrieve the average rating of drivers with a trip destined at that city.
Note: The results should include the destination city and the average driver rating for drivers who have trips to that city. The results must be sorted by the destination city in `alphabetical order` for consistent ordering.
Output: An array of `QueryResult.CityAndAverageDriverRating` objects containing `destination_city`, and `average_rating_of_drivers`.

5.15 Task 15: Find Total Bookings by Membership Status (10 pts)

Method: `getTotalBookingsEachMembershipStatus()`
For each membership status, count the total number of bookings made by passengers with that status. You should calculate total number of bookings for passengers within each membership status.
Note: The results should include the membership status and the total number of bookings for passengers with that membership status. The results must be sorted by membership status in ascending `alphabetical order` for consistent ordering.
Output: An array of `QueryResult.MembershipStatusAndTotalBookings` objects containing `membership_status`, and `total_number_of_bookings`.

5.16 Task 16: Update Driver Ratings (5 pts)

Method: `updateDriverRatings()`
For the drivers' ratings, if `rating` is smaller than 2.0 or equal to 2.0, update the rating by `adding 0.5`.
Note: If any driver's rating is bigger than 2.0, don't update.
Note 2: The `getAllDrivers` function within the `CarPoolingSystem` class is provided but initially commented out. It is disabled by default because there is no `Drivers` table initially, and attempting to call this function without the table will result in an error. After you have successfully initialized the `Drivers` table (which should be completed after insertion to the table), you should uncomment this function. This function is used to retrieve all the drivers from the database and is used for testing purposes after Task 16.
Output: The number of rows updated.

5.17 Task 17: Drop Database Tables (5 pts)

Method: `dropTables()`
You will drop all the database tables if they exist.
Output: The number of tables that are dropped successfully.

6 Extra Questions

Note that these **extra** questions **will not be graded**.

6.1 Task 18: Find Trips Departing from a Given City (Optional)

Method: `getTripsFromCity(String city)`
You will find all trips that depart from the specified city.
Note: The result should be sorted by the `TripID` field in ascending order to ensure the smallest TripID values appear first in the list.
Output: An array of `Trip` objects containing `TripID`, `CarID`, `date`, `departure`, `destination`, and `num_seats_available`.

6.2 Task 19: Find All Trips that Have Never Been Booked (Optional)

Method: `getTripsWithNoBooks()`
You will find all trips that have never been booked.
Note: The result should be sorted by the `TripID` field in ascending order to ensure the smallest TripID values appear first in the list.
Output: An array of `Trip` objects containing `TripID`, `CarID`, `date`, `departure`, `destination`, and `num_seats_available`.

6.3 Task 20: Find the most booked trip(s) per driver (Optional)

Method: `getTheMostBookedTripsPerDriver()`
You will retrieve the trip(s) with the highest number of bookings for each driver. If a driver has more than one trip with the same highest number of bookings, **all such trips should be included**. The result should be ordered by `DriverPIN` in ascending order. If a driver has multiple trips with the same highest number of bookings, those trips should be sorted by `TripID` in ascending order within the same `DriverPIN`.

Note:

- The result should include `DriverPIN`, `TripID`, and `NumberOfBookings`.
- If a driver has more than one trip with the highest number of bookings, all such trips will appear in the result, sorted by `TripID`.
- The final result list should be sorted by `DriverPIN` in ascending order.

Output: A list of `QueryResult.DriverPINandTripIDandNumberOfBookings` objects, where each object contains the `DriverPIN`, `TripID`, and `NumberOfBookings`. The list is ordered first by `DriverPIN` and then by `TripID` for trips with the highest number of bookings.

6.4 Task 21: Find Full Cars (Optional)

Method: `getFullCars()`
For each trip, if a car has 4 available seats and all of these 4 seats are booked with a "Confirmed" or "Pending" booking status, then the trip is considered to have a full car. You will find all full cars.
Note: The result should be sorted by the `TripID` field in ascending order to ensure the smallest TripID values appear first in the list.
Output: An array of `QueryResult.FullCars` objects containing `TripID`, `CarID`, `driver_name`, `color`, and `brand`.

7 Database Connection

To interact with the H2 database, you need to establish a connection. The database connection is handled in the `CarPoolingSystem` class and is initialized as follows:

```
public class CarPoolingSystem implements ICarPoolingSystem {

    private static String url =
        "jdbc:h2:mem:carpoolingdb;DB_CLOSE_DELAY=-1"; // In-
           memory database
    private static String user = "sa";            // H2 default
       username
    private static String password = "";          // H2 default
       password

    private Connection connection;

    public void initialize(Connection connection) {
        this.connection = connection;
    }
}
```

The `Evaluation.java` class handles the connection setup:

```
public static void connect() {
    try {
        connection = DriverManager.getConnection(url, user, password);
    } catch (SQLException e) {
        System.out.println("Cannot connect to database!");
        e.printStackTrace();
    }
}

// Usage
connect();

// Initialize CarPoolingSystem object
CarPoolingSystem carPoolingSystem = new CarPoolingSystem();
```

```
carPoolingSystem.initialize(connection); // Pass the H2 connection
```

This setup ensures that the database connection is properly established and passed to your implementation.

8 Statements and Prepared Statements

In Java, SQL queries can be executed using either `Statement` or `PreparedStatement` objects.

8.1 Statement

A `Statement` object is used to execute static SQL statements that don't require any parameters. It is suitable for executing simple SQL queries.

8.2 PreparedStatement

A `PreparedStatement` object is used for executing precompiled SQL statements that may contain parameters. It offers several advantages:

- Allows you to set parameters dynamically.
- Provides better performance as the SQL statement is precompiled.
- Enhances security by preventing SQL injection attacks.

8.3 Example Methods

Below are two example methods demonstrating the use of `Statement` and `PreparedStatement`.

8.3.1 Example 1: Get TripIDs of Trips Booked by "Jane Jones"

Using Statement:

```
public Integer[] getTripIDsOfJaneJones() {
    String query = "SELECT TripID FROM Bookings WHERE PIN =" +
                   "(SELECT PIN FROM Participants WHERE p_name =
                       'Jane Jones')";
    List<Integer> tripIDs = new ArrayList<>();

    try {
        Statement stmt = connection.createStatement();
        ResultSet rs = stmt.executeQuery(query);

        while (rs.next()) {
            tripIDs.add(rs.getInt("TripID"));
        }

        rs.close();
        stmt.close();
    } catch (SQLException e) {
```

```
        e.printStackTrace();
    }

    return tripIDs.toArray(new Integer[0]);
}
```

8.3.2 Example 2: Get TripIDs of Trips Booked by a Given Passenger

Using PreparedStatement:

```
public Integer[] getTripIDsOfGivenPassenger(String passenger) {
    String query = "SELECT TripID FROM Bookings WHERE PIN =" +
                   "(SELECT PIN FROM Participants WHERE p_name =
                      ?)";
    List<Integer> tripIDs = new ArrayList<>();

    try {
        PreparedStatement pstmt = connection.prepareStatement(
           query);
        pstmt.setString(1, passenger);
        ResultSet rs = pstmt.executeQuery();

        while (rs.next()) {
            tripIDs.add(rs.getInt("TripID"));
        }

        rs.close();
        pstmt.close();
    } catch (SQLException e) {
        e.printStackTrace();
    }

    return tripIDs.toArray(new Integer[0]);
}
```

Explanation: In the first example, since the passenger name is fixed ("Jane Jones"), we can use a `Statement`. However, this approach is not secure if the input can vary, as it opens up the possibility of SQL injection attacks.
In the second example, we use a `PreparedStatement` to safely incorporate the passenger name parameter into the query. This method is preferred when dealing with user-supplied input.

9 Regulations

- **Programming Language:** Java (version 14).
- **Database:** Embedded H2 Database.
- **Cheating:** We have a zero-tolerance policy for cheating. Individuals involved in cheating will be punished according to university regulations.

- **Evaluation:** Input files are guaranteed to be correctly formatted, and sample data will be provided. Similar (and larger) data will be used during evaluation. Your program will be evaluated automatically using a "black-box" technique, so ensure that you adhere strictly to the specifications. You must accomplish tasks using SQL queries within the Java methods, not with other Java programming facilities.

10 Submission Instructions

You are required to submit only the `CarPoolingSystem.java` file. Ensure that your implementations are within this file. Submissions will be evaluated automatically, so adhere strictly to the instructions and method signatures provided.

11 Grading

Your implementations will undergo black-box testing. For your submissions to receive credit, the results must **exactly match** the expected outputs. That is, adhering precisely to the output specifications, including the order of outputs, is critical.

Grading will be as follows:

- Tasks 1 to 12 and 16, 17: 5 points each
- Tasks 13 to 15: 10 points each

Please note that extra questions are intended for self-study and will not be graded.
Be aware that a different dataset will be used for evaluation purposes.