CSCI3170 (2020-2021 1st term) Introduction to Database Systems Project– Ridesharing System

TA In Charge: Lai Ziliang (zllai@cse.cuhk.edu.hk), Zhang Mingxue (mxzhang@cse.cuhk.edu.hk)

Group Registration Deadline: 23:59 9th October 2020
Phase 1 Deadline: 23:59 23rd October 2020
Phase 2 Deadline: 23:59 4th December 2020

1. Introduction

You are required to implement a system for a ridesharing startup company. It manages data relevant to ridesharing, e.g., drivers, passengers, vehicles, requests, taxi stops, and trips. The system should provide an interactive interface to system administrators, passengers, managers, and drivers.

This project is divided into two phases:
In phase 1, you have to design the database (including an ER-diagram and a relational schema). A suggested solution will be provided after phase 1. You are required to use the suggested solution to complete phase 2.

In phase 2, you are required to implement the system as a Java command-line program.
Our tutors will give tutorials about connecting your program to a MySQL database system with JDBC API and deploying your work on the required platform.

This is a group project, and each group should have at most three members. ONLY one copy of the solution is required for each group. Please fill out the group registration form in the Blackboard system before the group registration deadline.

2. Milestones

Preparation

- Read the document thoroughly and make sure you understand all the assumptions and regulations stated in Section 4.

Phase 1 (20 %)

- According to the data specifications in Section 3, design an ER-diagram and transform it into a relational schema without any redundant fields and tables.

Phase 2 (80 %)

- According to the suggested solution of phase 1, implement a Java application that fulfills all requirements stated in Section 5.
- Debug your system with different datasets and user inputs.
- Write a readme file to describe the compilation and deployment of your system.

3. Data Files Specification

Every data file has a Unix line ending (i.e., the newline character is "**\n**") and is encoded in utf-8. The system should read every record stored in each file and put them into appropriate tables of the provided MySQL DBMS via JDBC API. There are five input files in total listed in the subsections. Each line of each input file is a sequence of attributes delimited by a comma (**,**). The definition of each attribute in each input file is defined in the corresponding subsections. The order of the attributes within each input file line follows that of the attribute in the corresponding subsection. A sample data set will be provided after the deadline of Phase 1.

3.1. Drivers – drivers.csv

Attribute Name	Format	Description
ID	A positive integer	A unique identifier for the driver.
Name	A non-empty string with at most 30 characters	The name of the driver.
Vehicle ID	A non-empty string with 6 characters	The ID of the vehicle belongs to the driver.
Driving years	A positive integer	Number of years the driver has been driving.

3.2. Vehicles – vehicles.csv

Attribute Name	Format	Description
ID	A non-empty string with 6 characters	The license plate number of the vehicle, which uniquely identifies a vehicle.
Model	A non-empty string with at most 30 characters	The model of the car, e.g., Toyota Prius, Tesla Model X.
Seats	A positive integer	The number of seats, excluding the driver seat, of the vehicle.

3.3. Passengers – passengers.csv

Attribute Name	Format	Description
ID	A positive integer	A unique identifier for the passenger.
Name	A non-empty string with at most 30 characters	The name of the passenger.

3.4. Trips – trips.csv

Attribute Name	Format	Description
ID	A positive integer	A unique identifier for the trip.
Driver ID	A positive integer	The ID of the driver of the trip.
Passenger ID	A positive integer	The ID of the passenger of the trip.
Start time	A time (format see 4.1)	The time when the trip starts.
End time	A time (format see 4.1)	The time when the trip ends.
Start location	A non-empty string with at most 20 characters	The starting taxi stop of the trip.
Destination	A non-empty string with at most 20 characters	The destination taxi stop of the trip.
Fee	A positive integer	The fee of the trip.

3.4. Taxi stops – taxi_stop.csv

Attribute Name	Format	Description
Name	A non-empty string with at most 20 characters	The unique name of the taxi stop.
Location x	An integer	The x coordinate of the taxi stop on the map.
Location y	An integer	The y coordinate of the taxi stop on the map.

4. Assumptions

4.1. System

- All numerical values will not be larger than the maximum integer value that can be handled by Java.
- Every time input should follow the format "YYYY-MM-DD HH:mm:ss", e.g., "2020-08-10 23:08:53", regardless of the time zone.
- Every date input should follow the format "YYYY-MM-DD", e.g. "2020-07-15".
- There is no duplicate row in any input files.
- Any user inputs to the system are **correct in format only**.
- Every data file is **correct in format and content**.
- The distance between two locations is measured by Manhattan distance, i.e., the distance between (x1, y1) and (x2, y2) is $|x1 - x2| + |y1 - y2|$.

4.2. Drivers

- Every driver should be uniquely identified by his/her ID.
- Every driver should have only one vehicle.

4.3. Vehicles

- Every vehicle should be uniquely identified by its ID, which is also its license plate number.
- Each vehicle should have 3 to 7 seats, excluding the driver seat.
- Every vehicle should belong to only one driver.

4.4. Passengers

- Every passenger should be uniquely identified by his/her ID.

4.5. Requests

- Every request should be uniquely identified by its ID.
- Each request should have 1 to 8 passengers.
- A request should be marked taken once a driver handles it.
- An open request is a request which is not taken by any driver yet.
- A passenger should not be allowed to place a request if there is an open request by him/her.
- The model criterion of every request should be at most 30 characters.
- The start location and the destination are the names of taxi stops, and they should be different.

4.6. Trips

- Every trip should be uniquely identified by its ID.
- A trip should be created once a driver took a request. Therefore, there may be unfinished trips.

4.7. Taxi Stops

- Every taxi stop should be uniquely identified by its name.

5. System Function Requirements

You are required to write a simple command-line application in Java. On startup, the system should display a list for the user to choose his/her role:

```
Welcome! Who are you?
1. An administrator
2. A passenager
3. A driver
4. A manager
5. None of the above
Please enter [1-4]
```

Figure 1: Choosing the role.

After an operation, the program should display the last appeared menu. The Java program should provide the following functions:

5.1. System Administrator

The system should let administrators perform the following operations:

- **Create tables**

  An administrator can create all the tables in the database based on the given relational schema.

```
Administrator, what would you like to do?
1. Create tables
2. Delete tables
3. Load data
4. Check data
5. Go back
Please enter [1-5]
1
Processing...Done! Tables are created!
```

Figure 2: Sample input and output of creating tables.

- **Delete tables**

  An administrator can delete all tables in the database created based on the given relational schema.

```
Administrator, what would you like to do?
1. Create tables
2. Delete tables
3. Load data
4. Check data
5. Go back
Please enter [1-5]
2
Processing...Done! Tables are deleted
```

Figure 3: Sample input and output of deleting tables.

- **Load data**

  An administrator can load the system with data. The system should let the administrator enter the path of the folder containing the data files, then read each data file and load it into a table in the database. You may assume the folder contains all 4 data files, namely drivers.csv, vehicles.csv, passengers.csv, and trips.csv. See section 3 for data file specifications.

```
Administrator, what would you like to do?
1. Create tables
2. Delete tables
3. Load data
4. Check data
5. Go back
Please enter [1-5]
3
Please enter the folder path
test_data
Processing...Data is loaded!
```

Figure 4: Sample input and output of loading data.

- **Check data**

  An administrator can check the data in the database. The system should display the number of records in each table in the database.

```
Administrator, what would you like to do?
1. Create tables
2. Delete tables
3. Load data
4. Check data
5. Go back
Please enter [1-5]
4
Numbers of records in each table:
Vehicle: 100
Passenger: 150
Driver: 100
Trip: 500
Request: 0
Taxi_Stop: 27
```

Figure 5: Sample output showing the number of records in each table.

5.2. Passenger

The system should let passengers perform the following operations:

- **Request a ride**

  A passenger can place requests with the criteria:

  1. Number of passengers
  2. The start location and the destination
  3. The partial model of the vehicle (optional)
  4. The minimum driving years of the driver (optional)

  The system should let the passenger enter his/her ID, the criteria, start location, and destination. He/she can press enter without any input to skip the optional criteria. The system should then search for any vehicles which are available (not in unfinished trips) and meets all the criteria: for criterion 1, all vehicles returned must have a number of seats which is greater than or equal to the request number of passengers; for criterion 2, only drivers that are sufficiently close to the start location can take the request (see Section 5.3); for criterion 3, all vehicles returned must have a model containing the keyword; for criterion 4, all the drivers returned must have driving years greater than or equal to the requested minimum driving years.

  The partial matching should be case-insensitive, and the input from the passenger is treated as a whole. For example, if a passenger enters “toyota”, the system may return a vehicle with the model “Toyota Prius”. If the passenger enters “toyota corolla”, the system should not split the input and should not return “Toyota Prius”.

  If any vehicles meet all the criteria, the system should create a new request and display a success message, including the possible number of drivers who will take the request, including drivers in unfinished trips. Otherwise, display a message telling the passenger to adjust the criteria.

```
Passenger,what would you like to do?
1. Request a ride
2. Check trip records
3. Go back
Please enter [1-3].
1
Please enter your ID.
1
Please enter the number of passengers.
3
Please enter the start location.
Sham Shui Po
Please enter the destination.
Jordan
Please enter the model. (Press enter to skip)
Honda
Please enter the minimum driving years of the driver. (Press enter to skip)
5
Your request is placed. 12 drivers are able to take the request.
```

Figure 6: Sample input and output of requesting a ride.

- **Check trip records**

  A passenger can check all his/her finished trips within a certain period. The system should let the passenger enter his/her ID, the start date, the end date (for date format, see section 4.1), and the destination, then display each matching trip by its ID, the driver's name, the vehicle ID, the vehicles' model, the start time, the end time, the fee, the start location and the destination, sorted in descending order of the start time.

```
Passenger,what would you like to do?
1. Request a ride
2. Check trip records
3. Go back
Please enter [1-3].
2
Please enter your ID.
1
Please enter the start date.
2020-01-01
Please enter the end date.
2020-12-31
Please enter the destination.
Sham Shui Po
Trip_id, Driver Name, Vehicle ID, Vehicle Model, Start, End, Fee, Start Location, Destination
500, Albert Chung, BP9474, Nissan Serena, 2020-10-30 18:01:02.0, 2020-10-30 19:05:54.0, 64, Diamond Hill, Sham Shui Po
```

Figure 7: Sample input and output of checking trip records.

5.3. Driver

- **Search requests**

  When a driver searches for requests, the system should ask the driver to enter his/her driver ID, his/her location (x and y coordinates separated a space), and the maximum distance between his/her location to the start location of the request. The system returns the requests that satisfy the following criteria:

  1. The request is still open (it has not been marked taken);
  2. The driver should be qualified for the request (see Section 5.2 *Request a ride*);
  3. The distance between the requested start location and the driver's location should be less than or equal to the specified distance.

  For each returned request, the system should display the request ID, the passenger's name, the number of passengers, the start location, and the destination.

```
Driver, what would you like to do?
1. Search requests
2. Take a request
3. Finish a trip
4. Go back
Please enter [1-4]
1
Please enter your ID.
8
Please enter the coordinates of your location.
100 200
Please enter the maximum distance from you to the passenger.
50
request ID, passenger name, num of passengers, start location, destination
1, Audrey Fu, 3, Sham Shui Po, Jordan
```

Figure 8: Sample input and output of searching requests.

- **Take a request**

  A driver who is not in an unfinished trip can take a request as long as his/her vehicle satisfies all the criteria of a request:

  1. The vehicle should have enough seats;
  2. The model of the vehicle should match the request;
  3. His/her driving years should be longer than or equal to that specified by the request.

  The system should let the driver enter his/her ID and the request ID, then mark the request taken, create a new trip, and display the trip by its ID, the passenger's name, and the start time.

```
Driver, what would you like to do?
1. Search requests
2. Take a request
3. Finish a trip
4. Go back
Please enter [1-4]
2
Please enter your ID.
8
Please enter the request ID.
1
Trip ID, Passenger name, Start
501, Audrey Fu, 2020-09-27 05:43:18
```

Figure 9: Sample input and output of taking a request.

- **Finish a trip**

  A driver who is in an unfinished trip can finish his/her current trip. The system should let the driver enter his/her ID, then look for an unfinished trip by this driver, display the trip by its ID, the passenger's id, and the start time. Then the system should let the driver confirm if he/she wants to finish the trip. If so, record the end time and calculate the fee for the trip. The fee should be the duration of the trip in minutes, rounded down to the nearest integer. The system should display the trip by its ID, the passenger's name, the start time, the end time, and the fee.

```
Driver, what would you like to do?
1. Search requests
2. Take a request
3. Finish a trip
4. Go back
Please enter [1-4]
3
Please enter your ID.
8
Trip ID, Passenger ID, Start
501, 1, 2020-09-27 05:43:18.0
Do you wish to finish the trip? [y/n]
y
Trip ID, Passenger name, Start, End, Fee
501, Audrey Fu, 2020-09-27 05:43:18, 2020-09-27 05:44:05, 0
```

Figure 10: Sample input and output of finishing a trip.

5.4. Manager

- **List all finished trips with traveling distances within a range.**
  The traveling distance is the distance between the start location and the end location.
  The system should ask the manager to enter the minimum traveling distance and the maximum traveling distance; and then display each finished trip by its ID, driver's name, passenger's name, start location, destination and duration of the trip in minutes (rounded down to the nearest integer). The displayed trips should have traveling distance within the specified range (both ends are included).

```
Manager, what would you like to do?
1. Find trips
2. Go back
Please enter [1-2]
1
Please enter the minimum traveling distance.
300
Please enter the maximum traveling distance.
330
trip id, driver name, passenger name, start location, destination, duration
111, Rita Au-Yeung, Albert Ma, Diamond Hill, Prince Edward, 42
488, Isabella Woo, Allen Mui, Kwun Tong, Lai Chi Kok, 68
408, Rita Shum, Denise Fung, Ngau Tau Kok, Tsim Sha Tsui, 84
307, Joe Chan, Sandy Tang, Prince Edward, Diamond Hill, 67
```

Figure 11: Sample input and output of finding trips.

`

5.5. Error Handling

Suppose there is a runtime error, including data not found, incorrect format, incorrect content, etc., the system should display an error message as shown below.

```
Passenger,what would you like to do?
1. Request a ride
2. Check trip records
3. Go back
Please enter [1-3].
4
[ERROR] Invalid input.
Please enter [1-3].
1
Please enter your ID.
1
Please enter the number of passengers.
1000
[ERROR] Invalid number of passengers.
Please enter the number of passengers.
3
Please enter the start location.
Mars
[ERROR] Start Location not found.
Please enter the start location.
Jordan
Please enter the destination.
Jordan
[ERROR] Destination and start location should be different.
```

Figure 12: Example of error handling.

6. Grading Policy

The mark distribution is as follows:

Phase	Content	Mark Distribution
1	ER-diagram	10%
	Relational schema (based on your ER-diagram)	10%
2	Java application	80%

- There will be a mark deduction if your application is terminated unexpectedly during the demonstration.
- You are not allowed to modify any source code during the demonstration.
- Every member of the same group will receive the same marks for the project. To encourage every student to participate in the project, a question about this project may be asked in the final examination.

7.Evaluation

- We will **compile** and **test** your system on a Linux 64-bit machine (linux1-16) of the CSE department with MySQL server provided by the department.
- We will release a test script on blackboard. Make sure your program can be compiled and run by the test script on the Linux 64-bit machine of the CSE department before submission. We will grade your implementation using the same script but different test data.
- Our script matches your output by keywords. So, you don't have to follow the strict formatting as long as your program returns correct answers to the queries. Nonetheless, we recommend you to follow the format shown in Figure 1-12.

8. Submission Methods

8.1. Phase 1

- Submit a PDF file (one copy for each group) to the Blackboard system.
- The PDF file should include an ER diagram, a relational schema, the group number, the names, and the student IDs of all group members of your group.

8.2. Phase 2

- Put all your source files into a single folder and name the file containing the main function as "Main.java".
- Also include a README file in the folder, which contains:
  - Your group number
  - The name and the student ID of each group member
- Compress the folder into a ZIP file and submit the ZIP file (one copy for each group) to the Blackboard system.