1 Project

The project has three deliverables. Each group will demo their application at the end of the semester. The deliverables are:

- ER-model: Each group should develop an ER-model for the application. This can be uploaded as any type of image file on gradescope. (please do not use esoteric formats). Deadline: April 11.
- Relational schema: The second deliverable is a translation of the ER-model into a relational schema implemented as an SQL script. The script should use Postgres's SQL dialect. Please upload the script as a simple .sql file on gradescope. Deadline: April 18
- The last deliverable is an online taxi rental management application that uses the relational schema defined in the first two deliverables. This application can be either a web or desktop application. Deadline: April 30

Each group should have three or four members. If the group has four members then the group should design an application with more requirements (as shown below). If you cannot find a group please send me on piazza or an email and I will assign you to a group.

2 Overview

The goal is to build a taxi rental management application. There will be three roles for the application: managers, clients, and drivers. Manager can manage drivers, add or delete a diver from the system, add or delete a car or model from the system and retrieve information about statistics of the drivers, rating, cars etc. Client can search for available cars or drivers and can book a rent. They can also add their addresses, and credit card information. Drivers can insert what car models they can drive and they can add their address.

3 Data Requirements

You should not use any n-ary relationship, for $n > 2$.

3.1 Managers

For every manager we store the name, the ssn (which is also the key), and the email.

3.2 Client

Every client should register with their name and email address. The email address is different for every client. Each client should have at least one but maybe multiple addresses. Each client should have at least one credit card. The client should be able to book a rent on a particular date (given that there are available models/drivers for that date. See below.) Finally a client might write an anonymous review to a driver.

3.3 Rent

The rent should store the date. We assume that a rent takes the entire day. Each rent is booked by exactly one client. The rent must have exactly one driver and exactly one car model. The rent should store the rentid which is unique.

3.4 Driver

Each driver has a name and an address. The name is different from each driver. A driver should have only one address. Each driver can work on zero or many rents. A driver can drive some car models. A driver might have received some reviews.

3.5 Car

For every car we store the brand of the car. Every car has a unique carid.

3.6 Model

For every car model we store the color, the construction year, and whether it has manual or automatic transmissions. A model is uniquely identified by the car and an attribute modelid.

3.7 Review

A review consists of a message and a rating (integer) from 0 to 5. A review is uniquely identified by a driver and an attribute reviewid.

3.8 Address

Every address consists of the name of the road, number, and city.

3.9 Credit card

A credit card belongs to one client. The credit card stores the credit card number. A credit card has exactly one payment address, which is an address that will be used for the payment.

4 Application requirements

The application should support the following actions for managers, drivers and clients.

4.1 Managers

1. Someone has the option to register as a manager adding their information (name, ssn, email). If someone is registered as a manager should be able to login using their ssn.
2. Managers should be able to insert or remove cars or models in the system.
3. Managers should be able to insert or remove drivers from the system along with their information such as name, and address.
4. Managers should be able to give as input a number k and the system should return the names and emails of the top-k clients with respect to the number of rents they have booked.
5. Managers should be able to generate a list containing every current car model and next to it the number of rents it has been used.
6. Similarly, they should be able to generate a list containing for every current driver in the system X: the name of X, total number of rents that X was the driver and the average rating of X.
7. A manager should be able to give as input a city C_1 and a city C_2 and the system should return the name and email of the clients who have at least one address from city C_1 and they have booked a rent having a driver with an address from city C_2.
8. (only groups of four) Managers should be able to report the names of (current) problematic local drivers. These are current drivers with average rating less than 2.5, whose address is in city 'Chicago', and drove in at least two rents booked by two clients who have at least one address from 'Chicago'.
9. (only groups of four) Managers should be able to report a list containing i) the brand of a car, ii) the average review ratings given to drivers who can drive at least one model by that brand, and iii) the number of rents that used a model by that brand.

4.2 Drivers

Drivers login with their name.

1. Drivers should be able to change their address if they want.
2. Drivers should be able to see the list of all car models.
3. They should also be able to declare what car models they can drive.

4.3 Clients

1. When a new client registers they should add their information (name, and email address). Furthermore, they insert their address(es) and credit card(s). Notice that a client might have an address X but some credit card with a payment address Y which is not the same as X. If a client is registered in the system, then the client can login to the system using their email.
2. A client should be able to give as input a date D, and see the list of each available (current) car models on D. A car model X is available on D if: i) X is not used at another rent on the same date D, ii) there exists at least one driver R who can drive X, and iii) driver R does not drive on another rent that date D.
3. A client should be able to book a rent with an available car model on a specific date. The system automatically assigns any arbitrary available driver who can drive the requested car model. If there is no available car model on thar date the system should return an error message to the user.
4. A client should be able to see a list of all rents that the client has booked, along with the car model and the assigned driver.
5. The user should be able to enter a review to a driver (that currently exists in the system). The system should check whether the driver has been assigned to a rent booked by the client. Otherwise, the system should not allow user to enter a review.
6. (only groups of 4) A client should have the option to book a rent with an available specific car model with the best driver. If the client chooses this option then the system automatically assigns the driver in the rent with the highest average rating among the available drivers who can drive the requested available car model.