Project: Car Rental Database

Your company, Apasaja Pte Ltd, has been commissioned to design and implement a database application to record information about car rental operation for the PinjaMobil Pte Ltd. The application stores and manages historical information about the customers and cars.

The application stores and manages information about customers. A customer is identified by their NRIC number. Some customer may have driving license. The application also stores a name of the customer, the date of birth, as well as car preference. The car preference will be based on a brand and model of the car as stored in the database.

The application also stores and manages information about the cars. A specific car is identified by their license plate with the color of the car recorded. The current scheme for license plate starts with letter S followed by two letters, four digits, and the final checksum letter (see: Wikipedia). The brand and the model of each car are also recorded, which we call as the make of the car.

To accommodate future purchases, the application should allow recording of the make of the car even if we have currently no car with the given brand and model. This is also important as some customer may have a preference for a car make that the company does not have yet. There are also additional information about the capacity of the car that depends on the make of the car. As there is no modification of the car, all cars with the same make will have the same capacity.

The application tracks which customers rented which cars. The record of the start date and end date are recorded for each rental. We will use the notation [start, end] to indicate the interval from start to end *inclusive* of both dates. Given two interval [s1, e1] and [s2, e2], we will still treat the following two cases as overlapping interval: (i) e1 = s2 and (ii) e2 = s1. This is because the interval we are using is *inclusive* of both dates. Obviously, a car already rented in the interval [s1, e1] cannot be rented again in the interval [s2, e2] if the interval overlaps.

The application also tracks all passenger information of a car that are being rented. If a car is not currently being rented, there should be no passenger. For simplicity, the passenger will ride for the entire duration of the rented car. So if a car is being rented in the interval [s, e], then any passenger of this car will ride in the car for the entire interval [s, e].

Note that the customer renting need not be one of the passengers. In other words, a customer may initiate a rent for another passenger. The number of passenger must be smaller than or equal to the capacity of the car depending on their make. Furthermore, one *or more* of the passenger must have a driving license. Similar to how a car cannot be rented on overlapping dates, a customer cannot be a passenger on two different cars being rented with overlapping dates.

Finally, the application stores and manages only historical records and does not manage future events. For our purpose, given the new year of 2025, we will only test on dates of up to 2024. You do not have to make an explicit check for the date.

References

[1] Iso 8601 date and time format. `https://www.iso.org/iso-8601-date-and-time-format.html`. Visited on 31 January 2024.

[2] Iso week date. `https://en.wikipedia.org/wiki/ISO_week_date`. Visited on 31 January 2024.

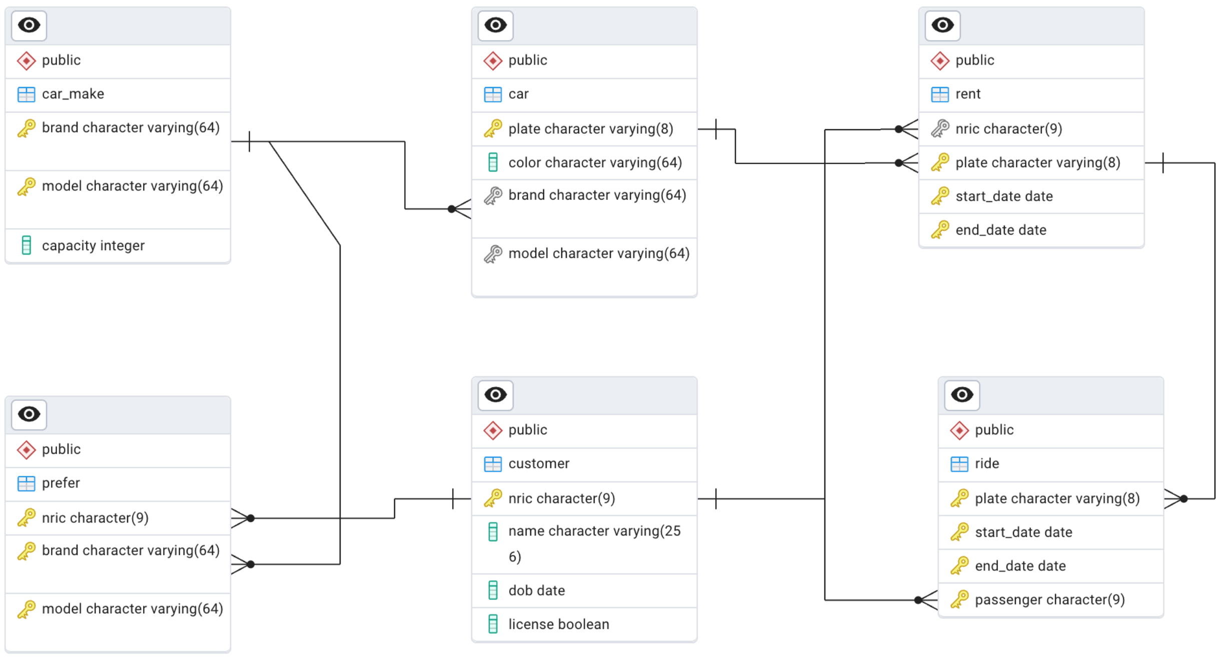
[3] Mockaroo - random data generator and api mocking tool. `www.mockaroo.com`. Visited on 31 January 2024.

[4] National registration identity card. `https://en.wikipedia.org/wiki/National_Registration_Identity_Card`. Visited on 31 January 2024.

[5] Vehicle registration plates of singapore. `https://en.wikipedia.org/wiki/Vehicle_registration_plates_of_Singapore`. Visited on 31 January 2024.

3. (6 points) Stored Procedures and Triggers

The initial schema can be found on Canvas Files > Project. Additionally, the logical diagram is shown below. Note that in our database, a passenger is a customer who is riding a rented car. In other words, it is based on the entry in the `ride` table.


public
car_make
brand character varying(64)
model character varying(64)
capacity integer
public
car
plate character varying(8)
color character varying(64)
brand character varying(64)
model character varying(64)
public
rent
nric character(9)
plate character varying(8)
start_date date
end_date date
public
prefer
nric character(9)
brand character varying(64)
model character varying(64)
public
customer
nric character(9)
name character varying(256)
dob date
license boolean
public
ride
plate character varying(8)
start_date date
end_date date
passenger character(9)

(a) (4 points) Triggers

During the design of our PinjaMobil Pte Ltd database, our goal was to capture as many application constraints as possible. Unfortunately, there are certain constraints we cannot enforce using only `PRIMARY KEY`, `UNIQUE`, `NOT NULL`, `FOREIGN KEY`, and `CHECK`. For instance, we cannot enforce the non-overlap constraint on car renting start and end date.

For this question, your task is to implement a series of triggers to capture additional application constraints. More specifically, your triggers should capture and preserve the following **4 constraints**.

(1) The same car cannot be rented on two different rent with overlapping dates. In other words, there is no overlapping start and end date for car rental. Note that in our schema, the rental date is *inclusive* of both start date and end date.

(2) A passenger (*i.e., a customer riding a rented car*) cannot be on two different cars rented with overlapping dates. Note that in our schema, the rental date is *inclusive* of both start date and end date.

(3) The number of passengers (*i.e., customers riding a rented car*) is less than or equal to the capacity of the car.

(4) One of the passengers (*i.e., a customer riding a rented car*) must be a driver but there can be more than one drivers. In other words, one or more of the passengers must have driver's licenses.

As our database is a historical database, you only need to ensure that insertion and update can be handled correctly. We will assume that no deletion is to be performed. Additionally, you may choose how you want to prevent the insertion and/or update operations (*e.g.,* raise exceptions on `AFTER` trigger or return `NULL` on `BEFORE` trigger).

However, your trigger should work for a transaction. In particular, if we have a transaction that performs several insertion on several tables, as long as the final state of the database is consistent, the entire transaction should be committed.

(b) (2 points) Procedures

Now that you have your triggers in place, your last task is to implement common actions as stored procedures. Here, we want to focus on the act of renting a car. The following two procedures should work even in the presence of your triggers.

(1) Implement the stored procedure `rent_solo` that attempt to rent the `car` (*i.e., license plate*) to the given `customer` (*i.e., the NRIC*) from the given `start_date` to the given `end_date`. The only passenger is the given `customer`.

You may assume that the car license plate and the customer NRIC number already exist in the database. For any other issues with the data, the operation should be rolled back.

```
CREATE OR REPLACE PROCEDURE rent_solo
  (IN car VARCHAR(8), IN customer CHAR(9),
   IN start_date DATE, IN end_date DATE)
AS
$$
  ...
$$ LANGUAGE plpgsql;
```

(2) Implement the stored procedure `rent_group` that attempt to rent the `car` (*i.e., license plate*) to the given `customer` (*i.e., the NRIC*) from the given `start_date` to the given `end_date`. For simplicity, you are also given **exactly 4 passengers** namely, `passenger1`, `passenger2`, `passenger3`, and `passenger4`. All passengers are given as NRIC number.

You may assume that the car license plate and the customer NRIC number already exist in the database. Additionally, you may assume all passengers already exist in the database. For any other issues with the data, the operation should be rolled back.

```
CREATE OR REPLACE PROCEDURE rent_group
  (IN car VARCHAR(8), IN customer CHAR(9),
   IN start_date DATE, IN end_date DATE,
   IN passenger1 CHAR(9), IN passenger2 CHAR(9),
   IN passenger3 CHAR(9), IN passenger4 CHAR(9))
AS
$$
  ...
$$ LANGUAGE plpgsql;
```

Submission

- Canvas Submission: "`Assignments > Questions 3`" (*one file per project group*)
- **Files**
  - Trigger: `triggers.sql` and `triggers-tests.sql`
    - The definition for your trigger and trigger functions should be in `triggers.sql`.
    - `triggers-tests.sql` should contain only your own test cases without definition for your trigger and trigger functions. This is *optional* and used to understand the behavior of your triggers.
  - Stored Procedures: `procedures.sql` and `procedures-tests.sql`
    - The definition for your stored procedures should be in `procedures.sql`. You may define more procedures that required as a good software engineering practice.
    - `procedures-tests.sql` should contain only your own test cases without definition for your stored procedures. This is *optional* and used to understand the behavior of your stored procedures.

Note: Submit four separate files. Ensure the name is correct before submission.

Testing

Having your test cases will help us understand your code and help us with the marking in case our test cases fail. For your test cases, you may assume the following steps which will be similar to how we will perform our test.

1. Start from a **fresh database** without any tables, procedures, and/or triggers.
2. Create all tables from the **given schema**.
3. Insert **valid rows** into the database as an initial data.
4. Create all procedures and triggers from `triggers.sql` and `procedures.sql`.
5. Perform tests (*e.g.,*) `triggers-tests.sql` and `procedures-tests.sql`.
    - For a transaction, as long as the final state of the database is consistent, the entire transaction should be committed regardless of the intermediate state.

As a good practice and to enhance your understanding of your implementation and/or problem description, you should prepare test cases that will satisfy the constraints as well as test cases that will not satisfy the constraints.