CSI 410. Database Systems – Spring 2021

Homework Assignment II

The deadline for this assignment is **11:59 PM, March 10, 2021**. *Submissions after this deadline will not be accepted.* Each student is required to enter the UAlbany Blackboard system and then upload a .pdf file (in the form of [first name]_[last name].pdf) that contains answers to Problems 1-2. We strongly recommend you to evaluate your SQL queries using PostgreSQL or some other database management system and then include these queries in the file mentioned above.

The total grade for this assignment is 100 points. If you find any error or have questions or suggestions, please contact the instructor (`jhh@cs.albany.edu`).

Problem 1. (80 points) Consider the relational database defined below:

```
create table passenger (
 passenger_ID varchar(10),
 passenger_name varchar(30),
 passenger_city varchar(30),
 primary key (passenger_ID));

create table seat (
 train_number varchar(10),
 seat_number varchar(10),
 primary key (train_number, seat_number));

create table reservation (
 reservation_number varchar(10),
 passenger_ID varchar(10),
 train_number varchar(10),
 seat_number varchar(10),
 departure_station varchar(10),
 departure_time timestamp,
 arrival_station varchar(10),
 arrival_time timestamp,
 fare numeric(8, 2),
 primary key (reservation_number),
 foreign key (passenger_ID) references passenger,
 foreign key (train_number, seat_number) references seat);
```

Express in **SQL** each of the following queries:

(a) (10 points) Shows the number of passengers living in (i.e., whose `passenger_city` is) `'Albany'`.
(*Answer*)

```
select count(passenger_ID)
from passenger
where passenger_city = 'Albany';
```

(b) (10 points) Show the train number of each train that has more than 1000 seats.
(*Answer*)

```
select train_number
from seat
group by train_number
having count(seat_number) > 1000;
```

(c) (10 points) Find the ID of every passenger who lives in (i.e., whose `passenger_city` is) `'Albany'` and has never reserved a trip arriving at 'ALB'. For this query, use either an `in` clause or a `not in` clause.

(*Answer*)

```
select passenger_ID
from passenger
where passenger_city = 'Albany' and passenger_ID not in (select passenger_ID
 from reservation
 where arrival_station = 'ALB');
```

(d) (10 points) Find the ID of every passenger who lives in (i.e., whose `passenger_city` is) `'Albany'` and has never reserved a trip arriving at 'ALB'. For this query, use a set operation.

(*Answer*)

```
(select passenger_ID
 from passenger
 where passenger_city = 'Albany')
except
(select passenger_ID
 from reservation
 where arrival_station = 'ALB');
```

(e) (10 points) Find the ID of every passenger who lives in (i.e., whose `passenger_city` is) `'Albany'` and has never reserved any trip. For this query, use a left outer join.

(*Answer*)

```
select passenger_ID
from passenger natural left outer join reservation
where passenger_city = 'Albany' and arrival_station is null;
```

(f) (10 points) For each reservation whose departure time is between '2021-04-01 00:00:00' and '2021-04-02 23:59:59' (inclusive), update the fare as follows: If the current fare is greater than $100, decrease the fare by 5 percent. Otherwise, decrease the fare by 3 percent.

(*Answer*)

```
update reservation set fare = case
 when departure_time between '2021-04-01 00:00:00' and '2021-04-02 23:59:59'
 and fare > 100 then fare*0.95
 when departure_time between '2021-04-01 00:00:00' and '2021-04-02 23:59:59'
 and fare <= 100 then fare*0.97
end;
```

or

```
update reservation set fare = case
  when fare > 100 then fare*0.95
  else fare*0.97
end
where departure_time between '2021-04-01 00:00:00' and '2021-04-02 23:59:59';
```

or

```
update reservation set fare = fare*0.97
where departure_time between '2021-04-01 00:00:00' and '2021-04-02 23:59:59'
 and fare <= 100;
update reservation set fare = fare*0.95
where departure_time between '2021-04-01 00:00:00' and '2021-04-02 23:59:59'
 and fare > 100;
```

Note that the following is incorrect:

```
update reservation set fare = fare*0.95
where departure_time between '2021-04-01 00:00:00' and '2021-04-02 23:59:59'
 and fare > 100;
update reservation set fare = fare*0.97
where departure_time between '2021-04-01 00:00:00' and '2021-04-02 23:59:59'
 and fare <= 100;
```

In the above case, tuples whose `departure_time` is '2021-04-01 00:00:00' and `fare` is 101 will be updated twice.

(g) (10 points) Find the train(s) with the largest number of seats (i.e., trains with more seats or the same number of seats compared to every other train). For each of these trains, show the train number.

(*Answer*)

```
select train_number
from seat
group by train_number
having count(seat_number) >= all (select count(seat_number) from seat
 group by train_number);
```

Another one, which is equivalent to the answer to Problem 2 (b), is as follows:

```
with t(train_number, seat_count) as (
  select train_number, count(seat_number)
  from seat
  group by train_number),
t2(seat_count) as (
  select max(seat_count) as seat_count
  from t
)
select train_number
  from t natural join t2;
```

(h) (10 points) Find all of the trains whose seats were all reserved at least once on '2019-04-01'. For each of these trains, show the train number.

(*Answer*)

```
select distinct train_number
from seat as s1
where not exists (
(select seat_number from seat as s2 where s1.train_number = s2.train_number)
except
(select seat_number from reservation
 where reservation.train_number = s1.train_number
 and departure_time < '2019-04-02 00:00:00' and arrival_time >= '2019-04-01 00:00:00')
```

Problem 2. (20 points) Consider the relational database defined in Problem 1. Express in **relational algebra** each of the following queries:

(a) (10 points) Show the train number of each train that has more than 1000 seats.

(*Answer*)

$\Pi_{train\_number}(\sigma_{t>1000}(_{train\_number}\mathcal{G}_{count(seat\_number)\ as\ t}(seat)))$

(b) (10 points) Find the train(s) with the largest number of seats (i.e., trains with more seats or the same number of seats compared to every other train). For each of these trains, show the train number.

(*Answer*)

$t \leftarrow_{train\_number} \mathcal{G}_{count(seat\_number)\ as\ seat\_count}(seat)$

$\Pi_{train\_number}(t \bowtie \mathcal{G}_{max(seat\_count)\ as\ seat\_count}(t))$

After solving the above problems, please state the amount of time spent for this assignment. Feel free to add comments or suggestions if any.

Appendix A. Installing and Running PostgreSQL

1. Visit:

   `https://www.enterprisedb.com/downloads/postgres-postgresql-downloads`

2. On the EnterpriseDB webpage, download an installer for your computer (installers for Linux, Windows and Mac OS X are available). This instruction is based on version 10.16.

3. Run the PostgreSQL installer. An installation guide is available at:

   `https://www.postgresql.org/docs/10/static/index.html`.

   You don't have to include "Stack Builder".

4. Start "pgAdmin 4". Then, double click the "PostgreSQL 10" icon (Figure 1) and type the password (for the user `postgres`) to connect to the PostgreSQL server.

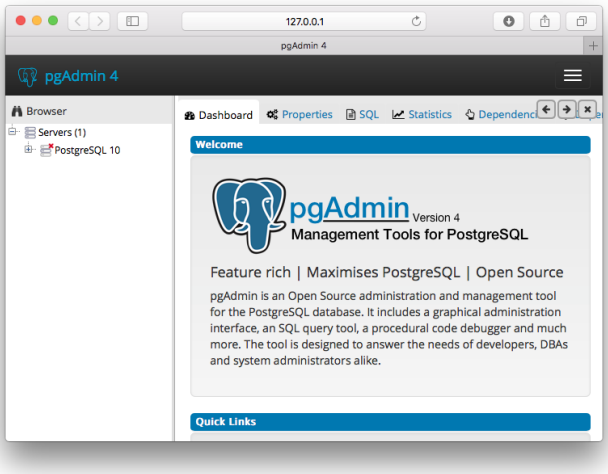

Figure 1: Connecting to the PostgresSQL server

5. As in Figure 2, choose the "`postgres`" database in the "Browser". In the menu bar, choose "Tools" and then "Query Tool". You can now type SQL commands in the SQL editor and run them.

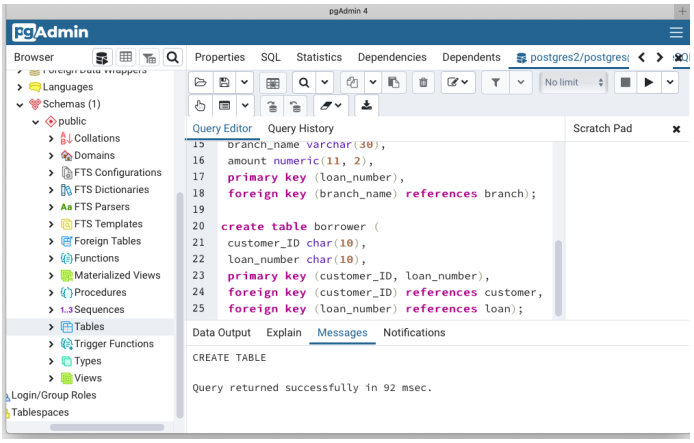

Figure 2: Selecting the "`postgres`" database

6. After creating tables, you can see these tables in pgAdmin 4 using the "Browser" (Figure 3).

Appendix B. Using the EXPLAIN command in Postgres

You can see the execution plan that Postgres constructs for each submitted SQL statement. Further details of this feature can be found at:

`https://www.postgresql.org/docs/10/static/sql-explain.html`.

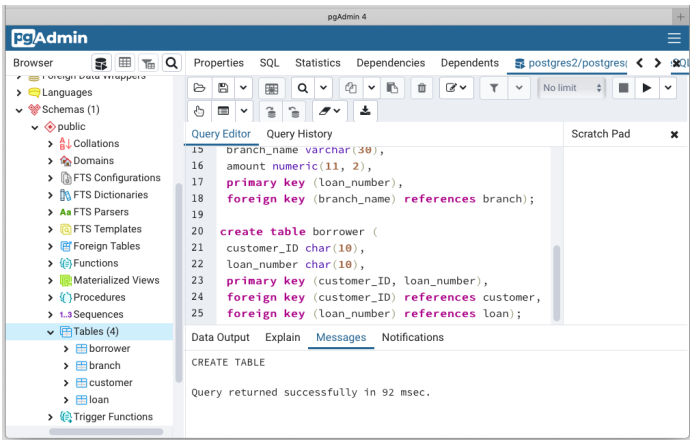

Figure 3: Created Tables