COMP 421 Database Systems, Winter 2019

Project Deliverable 2

Group 64

Chris Li 260708306
Yuxiang Ma 260714506
Yudi Xie 260712639
Naxin Fang 260783488

1. Relational Schema

Users(userName, firstName, lastName, phone, email)

Passengers(userName, homeLocation, workLocation) (userName ref Users)

Drivers(userName, status, driverLicence) (userName ref Users)

Admins(userName) (userName ref Users)

Vehicles(vehicleId, licencePlate, model, color, owner)

Comments(commentId, rating, timestamp, content, userName, tripId) (userName ref Passenger) (tripId ref Trips)

Trips(tripId, title, price, numberOfSeatsAvailable, startLocation)

Stops(stopName, cityId) (cityName ref Cities)

Cities(cityId, cityName)

CreditCards(cardNumber, holderName, expiryDate, issuer)

Books(pname, tid,, cnum, bookTime) (pname ref Passengers) (tid ref Trips) (cnum ref CreditCards)

HasStops(stopLocation, tid) (tid ref Trips) (stopLocation ref Stops)

Leads(tid, dname, vid, postTime) (tid ref Trips) (dname ref Drivers) (vid ref Vehicles)

HoldCards(cnum, uname) (cnum ref CreditCards) (uname ref Users)

Manages(aname, cid) (aname ref Admins) (cid ref Cities)

2. Create Commands (part2_CREATE_table.sql)

```
-- create User table
CREATE TABLE Users (
    userName VARCHAR(20) PRIMARY KEY,
    lastName VARCHAR(20),
    firstName VARCHAR(20),
    email VARCHAR(30) UNIQUE,
    phone VARCHAR(15) UNIQUE
);

-- create Passengers table
CREATE TABLE Passengers (
    userName VARCHAR(20) UNIQUE,
    homelocation VARCHAR(30),
    workLocation VARCHAR(30)
) INHERITS (Users);

-- create Drivers table
CREATE TABLE Drivers(
  userName varchar(20) unique,
  driverLicense varchar(20),
  status varchar(20)
) INHERITS (Users);

-- create CreditCards table
CREATE TABLE CreditCards (
    cardNumber VARCHAR(20) PRIMARY KEY,
    holderName VARCHAR(30),
    issuer VARCHAR(20),
    expiryDate DATE
);

-- create HoldCards table
CREATE TABLE HoldCards (
```

```sql
    cardNumber VARCHAR(20) PRIMARY KEY ,
    userName VARCHAR(20) NOT NULL,
    FOREIGN KEY (cardNumber)
        REFERENCES CreditCards (cardNumber),
    FOREIGN KEY (userName)
        REFERENCES Passengers(userName)
);

-- create Vehicles table
CREATE TABLE Vehicles (
    vehicleID SERIAL PRIMARY KEY,
    model VARCHAR(15),
    color VARCHAR(15),
    licensePlate VARCHAR(15),
    owner VARCHAR(20),
    FOREIGN KEY (owner)
        REFERENCES Drivers (userName)
);

-- create Trips table
CREATE TABLE Trips (
    tripId SERIAL PRIMARY KEY,
    numberOfSeatsAvailable INTEGER,
    title TEXT,
    startTime TIMESTAMP,
    startLocation VARCHAR(20),
    price FLOAT
);

-- create Comments table
CREATE TABLE Comments (
    commentID SERIAL PRIMARY KEY,
    postTime TIMESTAMP,
    content TEXT,
    rating INT,
    uid VARCHAR(20) NOT NULL,
    tripId INT NOT NULL,
    FOREIGN KEY (uid)
        REFERENCES Passengers (userName),
    FOREIGN KEY (tripId)
        REFERENCES Trips (tripId)
```

```sql
);

-- create Cities table
CREATE TABLE Cities (
    cityId SERIAL PRIMARY KEY,
    cityName VARCHAR(20)
);

-- create Admins table
CREATE TABLE Admins (
    userName VARCHAR(20) UNIQUE
) INHERITS (Users);

-- create Stops table
CREATE TABLE Stops (
    cityId INT,
    stopName VARCHAR(30),
    PRIMARY KEY (cityId, stopName),
    FOREIGN KEY (cityId)
        REFERENCES Cities (cityId)
);

-- create HasStops table
CREATE TABLE HasStops (
    tripId INT,
    stopName VARCHAR(30),
    cityId INT,
    PRIMARY KEY (tripId , cityId , stopName),
    FOREIGN KEY (tripId)
        REFERENCES Trips (tripId),
    FOREIGN KEY (cityId, stopName)
        REFERENCES Stops (cityId, stopName)
);

-- create Leads table
CREATE TABLE Leads (
    postTime TIMESTAMP,
    vehicleId INT,
    uid VARCHAR(20),
    tripId INT,
    PRIMARY KEY (tripId),
    FOREIGN KEY (tripId)
        REFERENCES Trips (tripId),
    FOREIGN KEY (vehicleId)
        REFERENCES vehicles (vehicleId),
    FOREIGN KEY (uid)
        REFERENCES Drivers (userName)
);

-- create Books table
CREATE TABLE Books (
    bookTime TIMESTAMP,
    cardNumber VARCHAR(30),
    tripId INT,
    uid VARCHAR(20),
    PRIMARY KEY (uid , tripId , cardNumber),
    FOREIGN KEY (uid)
        REFERENCES Passengers (userName),
    FOREIGN KEY (tripId)
        REFERENCES Trips (tripId),
    FOREIGN KEY (cardNumber)
        REFERENCES creditcards (cardNumber)
);

-- create Manages table
CREATE TABLE Manages (
    adminName VARCHAR(20),
    cityId INT NOT NULL,
    PRIMARY KEY (adminName),
    FOREIGN KEY (adminName)
        REFERENCES Admins (userName),
    FOREIGN KEY (cityId)
        REFERENCES Cities (cityId)
);
```

```
               Table "cs421g64.admins"
  Column   |         Type          | Modifiers
-----------+-----------------------+-----------
 username  | character varying(20) | not null
 lastname  | character varying(20) |
 firstname | character varying(20) |
 email     | character varying(30) |
 phone     | character varying(15) |
Indexes:
    "admins_username_key" UNIQUE CONSTRAINT, btree (username)
Referenced by:
    TABLE "manages" CONSTRAINT "manages_adminname_fkey" FOREIGN KEY (adminname)
REFERENCES admins(username)
Inherits: users
```

```
                   Table "cs421g64.books"
   Column   |            Type             | Modifiers
------------+-----------------------------+-----------
 booktime   | timestamp without time zone |
 cardnumber | character varying(30)       | not null
 tripid     | integer                     | not null
 uid        | character varying(20)       | not null
Indexes:
    "books_pkey" PRIMARY KEY, btree (uid, tripid, cardnumber)
Foreign-key constraints:
    "books_cardnumber_fkey" FOREIGN KEY (cardnumber) REFERENCES
creditcards(cardnumber)
    "books_tripid_fkey" FOREIGN KEY (tripid) REFERENCES trips(tripid)
    "books_uid_fkey" FOREIGN KEY (uid) REFERENCES passengers(username)
```

```
                           Table "cs421g64.cities"
  Column  |         Type          |                        Modifiers
----------+-----------------------+-----------------------------------------------
 cityid   | integer               | not null default
nextval('cities_cityid_seq'::regclass)
 cityname | character varying(20) |
Indexes:
    "cities_pkey" PRIMARY KEY, btree (cityid)
Referenced by:
    TABLE "manages" CONSTRAINT "manages_cityid_fkey" FOREIGN KEY (cityid)
REFERENCES cities(cityid)
    TABLE "stops" CONSTRAINT "stops_cityid_fkey" FOREIGN KEY (cityid) REFERENCES
cities(cityid)
```

```
                           Table "cs421g64.comments"
  Column   |            Type             |                          Modifiers
-----------+-----------------------------+----------------------------------------
 commentid | integer                     | not null default
nextval('comments_commentid_seq'::regclass)
 posttime  | timestamp without time zone |
 content   | text                        |
 rating    | integer                     |
 uid       | character varying(20)       | not null
 tripid    | integer                     | not null
Indexes:
    "comments_pkey" PRIMARY KEY, btree (commentid)
Foreign-key constraints:
    "comments_tripid_fkey" FOREIGN KEY (tripid) REFERENCES trips(tripid)
    "comments_uid_fkey" FOREIGN KEY (uid) REFERENCES passengers(username)
```

Table "cs421g64.creditcards"

```
   Column   |         Type          | Modifiers
------------+-----------------------+-----------
 cardnumber | character varying(20) | not null
 holdername | character varying(30) |
 issuer     | character varying(20) |
 expirydate | date                  |
Indexes:
    "creditcards_pkey" PRIMARY KEY, btree (cardnumber)
Referenced by:
    TABLE "books" CONSTRAINT "books_cardnumber_fkey" FOREIGN KEY (cardnumber)
REFERENCES creditcards(cardnumber)
    TABLE "holdcards" CONSTRAINT "holdcards_cardnumber_fkey" FOREIGN KEY
(cardnumber) REFERENCES creditcards(cardnumber)
```

Table "cs421g64.creditcards"

```
   Column   |         Type          | Modifiers
------------+-----------------------+-----------
 cardnumber | character varying(20) | not null
 holdername | character varying(30) |
 issuer     | character varying(20) |
 expirydate | date                  |
Indexes:
    "creditcards_pkey" PRIMARY KEY, btree (cardnumber)
Referenced by:
    TABLE "books" CONSTRAINT "books_cardnumber_fkey" FOREIGN KEY (cardnumber)
REFERENCES creditcards(cardnumber)
    TABLE "holdcards" CONSTRAINT "holdcards_cardnumber_fkey" FOREIGN KEY
(cardnumber) REFERENCES creditcards(cardnumber)
```

Table "cs421g64.hasstops"

```
  Column  |         Type          | Modifiers
----------+-----------------------+-----------
 tripid   | integer               | not null
 stopname | character varying(30) | not null
 cityid   | integer               | not null
Indexes:
    "hasstops_pkey" PRIMARY KEY, btree (tripid, cityid, stopname)
Foreign-key constraints:
    "hasstops_cityid_fkey" FOREIGN KEY (cityid, stopname) REFERENCES stops(cityid,
stopname)
    "hasstops_tripid_fkey" FOREIGN KEY (tripid) REFERENCES trips(tripid)
```

```
                  Table "cs421g64.holdcards"
   Column   |         Type          | Modifiers
------------+-----------------------+-----------
 cardnumber | character varying(20) | not null
 username   | character varying(20) | not null
Indexes:
    "holdcards_pkey" PRIMARY KEY, btree (cardnumber)
Foreign-key constraints:
    "holdcards_cardnumber_fkey" FOREIGN KEY (cardnumber) REFERENCES
creditcards(cardnumber)
    "holdcards_username_fkey" FOREIGN KEY (username) REFERENCES
passengers(username)
```

```
                  Table "cs421g64.leads"
  Column   |            Type             | Modifiers
-----------+-----------------------------+-----------
 posttime  | timestamp without time zone |
 vehicleid | integer                     |
 uid       | character varying(20)       |
 tripid    | integer                     | not null
Indexes:
    "leads_pkey" PRIMARY KEY, btree (tripid)
Foreign-key constraints:
    "leads_tripid_fkey" FOREIGN KEY (tripid) REFERENCES trips(tripid)
    "leads_uid_fkey" FOREIGN KEY (uid) REFERENCES drivers(username)
    "leads_vehicleid_fkey" FOREIGN KEY (vehicleid) REFERENCES vehicles(vehicleid)
```

```
                  Table "cs421g64.manages"
  Column   |         Type          | Modifiers
-----------+-----------------------+-----------
 adminname | character varying(20) | not null
 cityid    | integer               | not null
Indexes:
    "manages_pkey" PRIMARY KEY, btree (adminname)
Foreign-key constraints:
    "manages_adminname_fkey" FOREIGN KEY (adminname) REFERENCES admins(username)
    "manages_cityid_fkey" FOREIGN KEY (cityid) REFERENCES cities(cityid)
```

```
                  Table "cs421g64.passengers"
    Column    |         Type          | Modifiers
--------------+-----------------------+-----------
 username     | character varying(20) | not null
 lastname     | character varying(20) |
 firstname    | character varying(20) |
 email        | character varying(30) |
 phone        | character varying(15) |
 homelocation | character varying(30) |
 worklocation | character varying(30) |
Indexes:
    "passengers_username_key" UNIQUE CONSTRAINT, btree (username)
Referenced by:
    TABLE "books" CONSTRAINT "books_uid_fkey" FOREIGN KEY (uid) REFERENCES
passengers(username)
    TABLE "comments" CONSTRAINT "comments_uid_fkey" FOREIGN KEY (uid) REFERENCES
passengers(username)
    TABLE "holdcards" CONSTRAINT "holdcards_username_fkey" FOREIGN KEY (username)
REFERENCES passengers(username)
Inherits: users
```

Table "cs421g64.stops"

```
  Column  |         Type          | Modifiers
----------+-----------------------+-----------
 cityid   | integer               | not null
 stopname | character varying(30) | not null
Indexes:
    "stops_pkey" PRIMARY KEY, btree (cityid, stopname)
Foreign-key constraints:
    "stops_cityid_fkey" FOREIGN KEY (cityid) REFERENCES cities(cityid)
Referenced by:
    TABLE "hasstops" CONSTRAINT "hasstops_cityid_fkey" FOREIGN KEY (cityid,
stopname) REFERENCES stops(cityid, stopname)
```

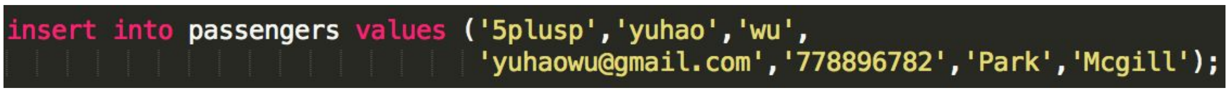
Table "cs421g64.trips"

```
         Column         |            Type             |
Modifiers
------------------------+-----------------------------+---------------------------
 tripid                 | integer                     | not null default
nextval('trips_tripid_seq'::regclass)
 numberofseatsavailable | integer                     |
 title                  | text                        |
 starttime              | timestamp without time zone |
 startlocation          | character varying(20)       |
 price                  | double precision            |
Indexes:
    "trips_pkey" PRIMARY KEY, btree (tripid)
Referenced by:
    TABLE "books" CONSTRAINT "books_tripid_fkey" FOREIGN KEY (tripid) REFERENCES
trips(tripid)
    TABLE "comments" CONSTRAINT "comments_tripid_fkey" FOREIGN KEY (tripid)
REFERENCES trips(tripid)
    TABLE "hasstops" CONSTRAINT "hasstops_tripid_fkey" FOREIGN KEY (tripid)
REFERENCES trips(tripid)
    TABLE "leads" CONSTRAINT "leads_tripid_fkey" FOREIGN KEY (tripid) REFERENCES
trips(tripid)
```

```
               Table "cs421g64.users"
  Column   |         Type          | Modifiers
-----------+-----------------------+-----------
 username  | character varying(20) | not null
 lastname  | character varying(20) |
 firstname | character varying(20) |
 email     | character varying(30) |
 phone     | character varying(15) |
Indexes:
    "users_pkey" PRIMARY KEY, btree (username)
    "users_email_key" UNIQUE CONSTRAINT, btree (email)
    "users_phone_key" UNIQUE CONSTRAINT, btree (phone)
Number of child tables: 3 (Use \d+ to list them.)
```

```
                         Table "cs421g64.vehicles"
   Column     |         Type          |                           Modifiers
--------------+-----------------------+-------------------------------------------
 vehicleid    | integer               | not null default
nextval('vehicles_vehicleid_seq'::regclass)
 model        | character varying(15) |
 color        | character varying(15) |
 licenseplate | character varying(15) |
 owner        | character varying(20) |
Indexes:
    "vehicles_pkey" PRIMARY KEY, btree (vehicleid)
Foreign-key constraints:
    "vehicles_owner_fkey" FOREIGN KEY (owner) REFERENCES drivers(username)
Referenced by:
    TABLE "leads" CONSTRAINT "leads_vehicleid_fkey" FOREIGN KEY (vehicleid)
REFERENCES vehicles(vehicleid)
```

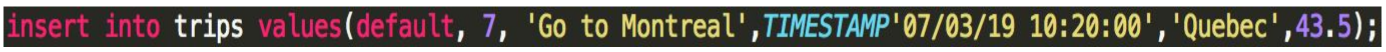
3. Insert Commands (part3-4_INSERT_data.sql)



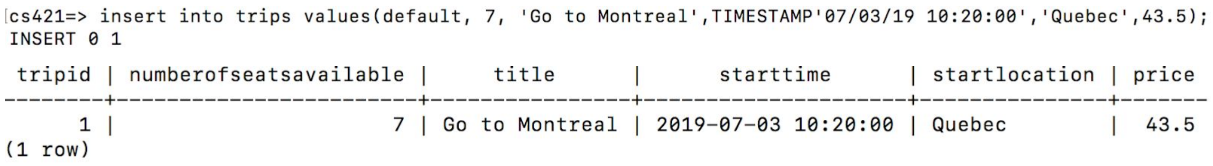
```
insert into passengers values ('5plusp','yuhao','wu',
                               'yuhaowu@gmail.com','778896782','Park','Mcgill');
```

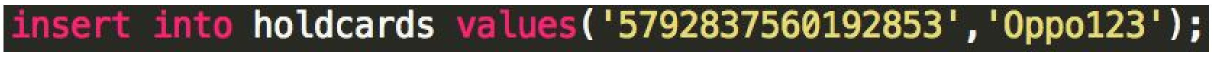
3.1 This query creates a general passenger user.

```
cs421=> insert into passengers values ('5plusp','yuhao','wu',
cs421(>                               'yuhaowu@gmail.com','778896782','Park','Mcgill');
INSERT 0 1

 username | lastname | firstname |       email       |   phone   | homelocation | worklocation
----------+----------+-----------+-------------------+-----------+--------------+--------------
 5plusp   | yuhao    | wu        | yuhaowu@gmail.com | 778896782 | Park         | Mcgill
(1 row)
```

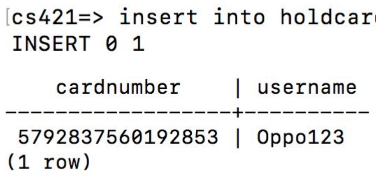

```
insert into trips values(default, 7, 'Go to Montreal',TIMESTAMP'07/03/19 10:20:00','Quebec',43.5);
```

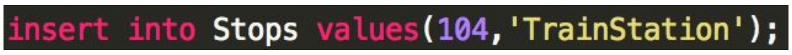
3.2 This query inserts a row into trips table, that goes from Quebec city to Montreal on July 3rd, and the price is $43.5


```
[cs421=> insert into trips values(default, 7, 'Go to Montreal',TIMESTAMP'07/03/19 10:20:00','Quebec',43.5);
 INSERT 0 1
 tripid | numberofseatsavailable |      title       |      starttime      | startlocation | price
--------+------------------------+------------------+---------------------+---------------+-------
      1 |                      7 | Go to Montreal | 2019-07-03 10:20:00 | Quebec        |  43.5
(1 row)
```

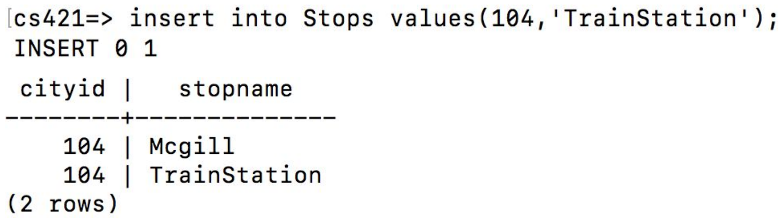

```
insert into holdcards values('5792837560192853','Oppo123');
```

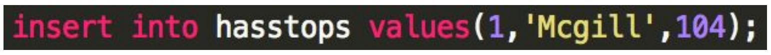
3.3 This query inserts a relation between credit card and a card holder ‘Oppo123’, that indicates this credit card is belong to the user.


```
[cs421=> insert into holdcards values('5792837560192853','Oppo123');
 INSERT 0 1
    cardnumber    | username
------------------+----------
 5792837560192853 | Oppo123
(1 row)
```

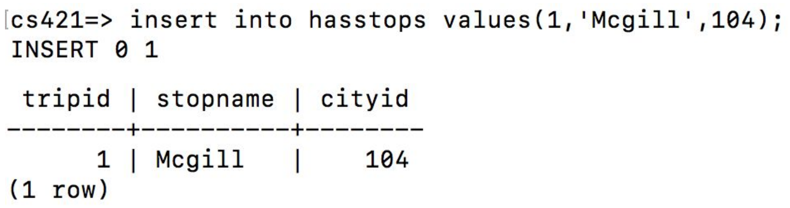

```
insert into Stops values(104,'TrainStation');
```

3.4 This query creates a Stop ‘TrainStation’ in city ID ‘104’, which is Montreal. In this database system, we can insert multiple row named ‘TrainStation’ into the Stops table as long as the cityID is different.


```
[cs421=> insert into Stops values(104,'TrainStation');
 INSERT 0 1
 cityid |   stopname
--------+--------------
    104 | Mcgill
    104 | TrainStation
(2 rows)
```


```
insert into hasstops values(1,'Mcgill',104);
```

3.5 This query creates a link between a trip and a city stop, trip #1 will be stop at McGill when arrived at Montreal.


```
[cs421=> insert into hasstops values(1,'Mcgill',104);
 INSERT 0 1
 tripid | stopname | cityid
--------+----------+--------
      1 | Mcgill   |    104
(1 row)
```

4. Populating Tables (part3-4_INSERT_data.sql)

Admins
```
   username    | lastname | firstname |           email            |   phone
---------------+----------+-----------+----------------------------+------------
 jyotiprakash  | jyoti    | prakash   | jyotiprakash@rideshare.io  | 7116633225
 manshishing   | manshi   | shing     | manshishing@rideshare.io   | 7520746847
 girishchand   | girish   | chand     | girishchand@rideshare.io   | 7506231410
 ravisingh     | ravi     | singh     | ravisingh@rideshare.io     | 0845478157
 harishchand   | harish   | chand     | harishchand@rideshare.io   | 3518504710
```

Books
```
      booktime       |    cardnumber    | tripid |    uid
---------------------+------------------+--------+------------
 2019-07-15 10:20:00 | 5792837560192853 |      1 | Oppo123
 2019-07-17 10:34:00 | 2847581029374814 |      2 | Xinyl
 2019-07-17 10:44:00 | 1984328437274304 |      3 | Huawei
 2019-07-19 10:55:00 | 1298498227465028 |      4 | xiaohuozhi
```

Cities
```
 cityid | cityname
--------+-----------
      4 | New Delhi
      5 | Tezpur
      6 | Jullundur
      7 | Navsari
      8 | Asansol
```

Comments
```
 commentid |      posttime       |           content         | rating | uid | tripid
-----------+---------------------+----------------------------+--------+---------
1 | 2019-07-21 10:20:00 | This is a very bad driver!  |      3 | Oppo123    |    1
2 | 2019-07-22 10:20:00 | This is a very bad driver!  |      4 | K9         |    1
3 | 2019-07-23 10:20:00 | This is a very bad driver!  |      4 | 5plusp     |    1
4 | 2019-07-25 10:20:00 | This is a very good driver! |      8 | Huawei     |    2
5 | 2019-07-25 10:20:00 | This is a very good driver! |      9 | xiaohuozhi |    2
```

Drivers
```
  username    | lastname | firstname |        email         |   phone    |
driverlicense | status
-------------+----------+-----------+----------------------+-----------+---------
 zhanlang123 | zhan     | lang      | a843172479@gmail.com | 778238129  |
123haskjd2    | working
 yss0755     | yao      | shenshun  | yaoyaole@gmail.com   | 514999238  |
123udemhec    | onbreak
```

```
 liudehua    | Liu      | Dehua     | dehua@gmail.com      | 5149992738 |
123sjdewbs    | working
 zhangxueyou | Zhang    | xueyou    | xueyou@gmail.com     | 5149999018 |
123sdwidcs    | working
 chenyixun   | chen     | eason     | yixun@gmail.com      | 5149958388 |
123xncfrjw    | onbreak
```

HasStops

```
 tripid |   stopname   | cityid
--------+--------------+--------
      1 | Mcgill       |    104
      1 | BellCentre   |    104
      1 | TrainStation |    104
      4 | UofT         |    105
      4 | Union        |    105
```

HoldCards

```
    cardnumber    | username
------------------+----------
 5792837560192853 | Oppo123
 2847581029374814 | Xinyl
 4028492840002847 | Mayx
 1827384927493029 | 5plusp
 9090334477889234 | Caige
```

Leads

```
      posttime       | vehicleid |     uid     | tripid
---------------------+-----------+-------------+--------
 2019-06-03 10:20:00 |         1 | zhangxueyou |      1
 2019-06-04 10:20:00 |         4 | chenyixun   |      2
 2019-06-05 10:20:00 |         3 | yss0755     |      3
 2019-06-06 10:20:00 |         5 | zhangwentao |      4
 2019-06-07 10:20:00 |         6 | liudehua    |      5
```

Manages

```
  adminname  | cityid
-------------+--------
 manshishing |    104
 mohdshubhan |    105
 shkurkhan   |    106
 devdutt     |    107
 gorvsharma  |    108
```

Passengers
```
 username | lastname |  firstname   |         email          |   phone    |
homelocation  | worklocation
----------+----------+--------------+------------------------+------------+-------
 Oppo123  | Fang     | Naxin        | 199888@163.com         | 5783218989 | ABC
appartment | Mcgill
 Mayx     | Ma       | Xinlaoshiren | newhonestman@gmail.com | 5143453455 | Lacite
| Cisco
 Xinyl    | Xinyu    | Li           | xinyuli@gmail.com      | 5143453723 | Lacite
| TUM
 5plusp   | yuhao    | wu           | yuhaowu@gmail.com      | 778896782  | Park
| Mcgill
 Caige    | Renjun   | Cai          | cairenjun@gmail.com    | 1391391887 |
3440durocher   | Mcgill
```

Stops
```
cityid |   stopname
--------+--------------
    104 | TrainStation
    104 | Mcgill
    104 | BellCentre
    105 | TrainStation
    105 | UofT
```

Trips
```
 tripid | numberofseatsavailable |       title       |      starttime      |
startlocation | price
--------+------------------------+-------------------+---------------------+------
      1 |                      7 | Go to Montreal    | 2019-07-03 10:20:00 |
Quebec        |  43.5
      2 |                      8 | A trip to NYC     | 2019-07-02 10:20:00 |
Kingston      |  69.5
      3 |                      2 | Go to Ottawa      | 2019-07-11 10:20:00 |
Kingston      |  79.5
      4 |                      3 | A trip to Toronto | 2019-07-12 10:20:00 |
Montreal      |  80.5
      5 |                      4 | Go to Vancouver   | 2019-07-13 10:20:00 |
Surrey        |  99.5
```

Users
```
 username | lastname |  firstname   |         email          |   phone
----------+----------+--------------+------------------------+------------
 Oppo123  | Fang     | Naxin        | 199888@163.com         | 5783218989
 Mayx     | Ma       | Xinlaoshiren | newhonestman@gmail.com | 5143453455
 Xinyl    | Xinyu    | Li           | xinyuli@gmail.com      | 5143453723
 5plusp   | yuhao    | wu           | yuhaowu@gmail.com      | 778896782
 Caige    | Renjun   | Cai          | cairenjun@gmail.com    | 1391391887
```

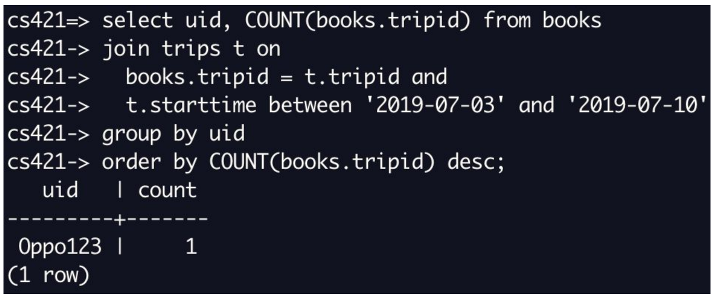
```
Vehicles
 vehicleid |    model    | color | licenseplate |    owner
-----------+-------------+-------+--------------+-------------
         1 | AudiA8      | Black | 768XU2       | zhangxueyou
         2 | AudiA4      | Black | 777969       | chenyixun
         3 | BMWX6       | White | 273YFR       | yss0755
         4 | ToyotaA7    | Black | H6F8B4       | chenyixun
         5 | HondaGMZ-8  | Red   | 2HGM92       | zhangwentao
```

5. Complex Queries (part5_SELECT_queries.sql)

5.1

```sql
select uid, COUNT(books.tripid) from books
join trips t on
 books.tripid = t.tripid and
 t.starttime between '2019-07-03' and '2019-07-10'
group by uid
order by COUNT(books.tripid) desc;
```

This query selects passenger rank and score from time period Jan 3 2018 to Jan 3 2019. Score was determined by count of his trip in a time period. This query first select trip from trips table between Jan 3 2018 and Jan 3 2019 and inner join with passenger table. Group by operations groups joined entries by passenger id then do a count of trips for each driver id group to compute score.
Output:

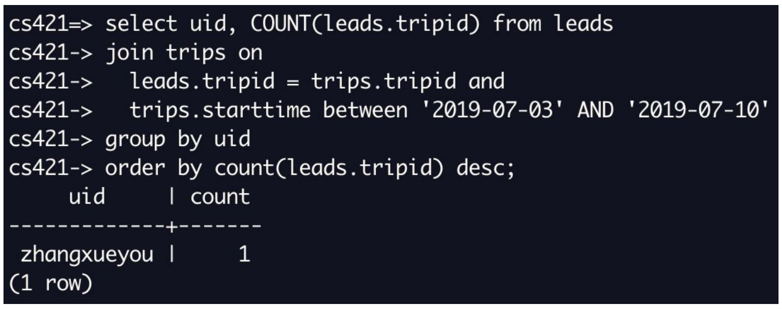


```
cs421=> select uid, COUNT(books.tripid) from books
cs421-> join trips t on
cs421->   books.tripid = t.tripid and
cs421->   t.starttime between '2019-07-03' and '2019-07-10'
cs421-> group by uid
cs421-> order by COUNT(books.tripid) desc;
   uid    | count
---------+-------
 Oppo123 |     1
(1 row)
```

5.2

```
select uid, COUNT(leads.tripid) from leads
join trips on
 leads.tripid = trips.tripid and
 trips.starttime between '2019-07-03' AND '2019-07-10'
group by uid
order by count(leads.tripid) desc;
```

This query selects driver rank and score from time period Jan 3 2018 to Jan 3 2019. Score was determined by count of his trip in a time period. This query first select trip from trips table between Jan 3 2018 and Jan 3 2019 and inner join with leads table on trip id. Group by operations groups joined entries by driver id then do a count of trips for each driver id group to compute score.
Output:

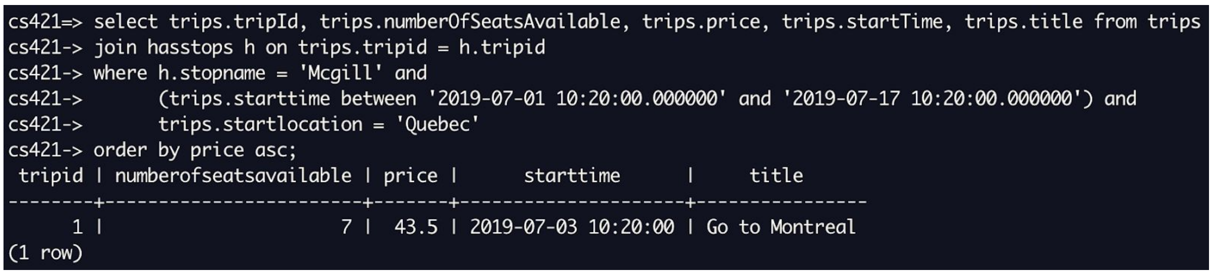


```
cs421=> select uid, COUNT(leads.tripid) from leads
cs421-> join trips on
cs421->   leads.tripid = trips.tripid and
cs421->   trips.starttime between '2019-07-03' AND '2019-07-10'
cs421-> group by uid
cs421-> order by count(leads.tripid) desc;
     uid     | count
-------------+-------
 zhangxueyou |     1
(1 row)
```

5.3

```
select trips.tripId, trips.numberOfSeatsAvailable, trips.price, trips.startTime,
trips.title from trips
join hasstops h on trips.tripid = h.tripid
where h.stopname = 'Mcgill' and
     (trips.starttime between '2019-07-01 10:20:00.000000' and '2019-07-17
10:20:00.000000') and
     trips.startlocation = 'Quebec'
order by price asc;
```

This query helps with searching a trip in a time period by start and end location. In this case, a trip start at “Quebec” and ends at “Mcgill”, time between July 1 2019 and July 17 2019 was searched. Final result was sorted by price to allow users to choose the most economical choice. This query selects trips starting at Ottawa and between Feb 14 2019 to Feb 17 2019, then selects stops with name Mcgill and inner join them on trip id. Output was sorted by price ascendingly.
Output:

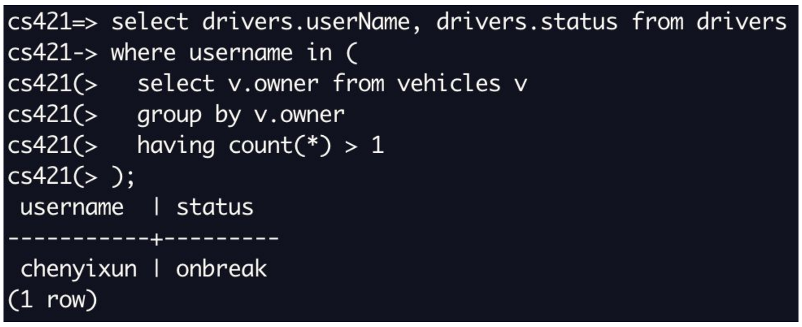


```
cs421=> select trips.tripId, trips.numberOfSeatsAvailable, trips.price, trips.startTime, trips.title from trips
cs421-> join hasstops h on trips.tripid = h.tripid
cs421-> where h.stopname = 'Mcgill' and
cs421->      (trips.starttime between '2019-07-01 10:20:00.000000' and '2019-07-17 10:20:00.000000') and
cs421->      trips.startlocation = 'Quebec'
cs421-> order by price asc;
 tripid | numberofseatsavailable | price |      starttime      |     title
--------+------------------------+-------+---------------------+----------------
      1 |                      7 |  43.5 | 2019-07-03 10:20:00 | Go to Montreal
(1 row)
```

5.4

```
select drivers.userName, drivers.status from drivers
where username in (
 select v.owner from vehicles v
 group by v.owner
 having count(*) > 1
);
```

This query searches all drivers with more than 1 cars and outputs their rating. It has a subquery and a parent query. Subquery groups all vehicles with owner (driver) and computer count of vehicles in each group, then outputs a set of driving ids (driver that has at least one vehicle) and driver status. Parent query iterates all driver and checks whether the driver is in the set, outputs the overall rating for each driver.
Output:

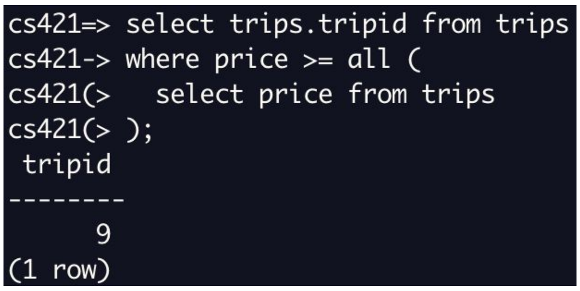


```
cs421=> select drivers.userName, drivers.status from drivers
cs421-> where username in (
cs421(>   select v.owner from vehicles v
cs421(>   group by v.owner
cs421(>   having count(*) > 1
cs421(> );
 username  | status
-----------+---------
 chenyixun | onbreak
(1 row)
```

5.5

```
select trips.tripid from trips
where price >= all (
 select price from trips
);
```

This query selects the most expensive trip. Subqueries selects all price, and parent query selects the one that larger or equal to all prices, which is the largest one.
Output:

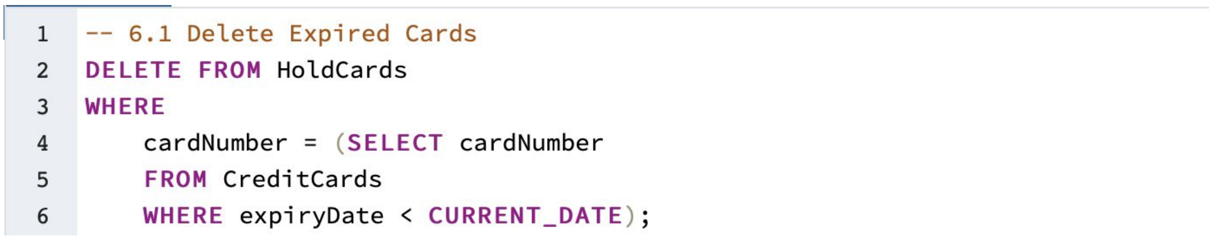


```
cs421=> select trips.tripid from trips
cs421-> where price >= all (
cs421(>   select price from trips
cs421(> );
 tripid
--------
      9
(1 row)
```

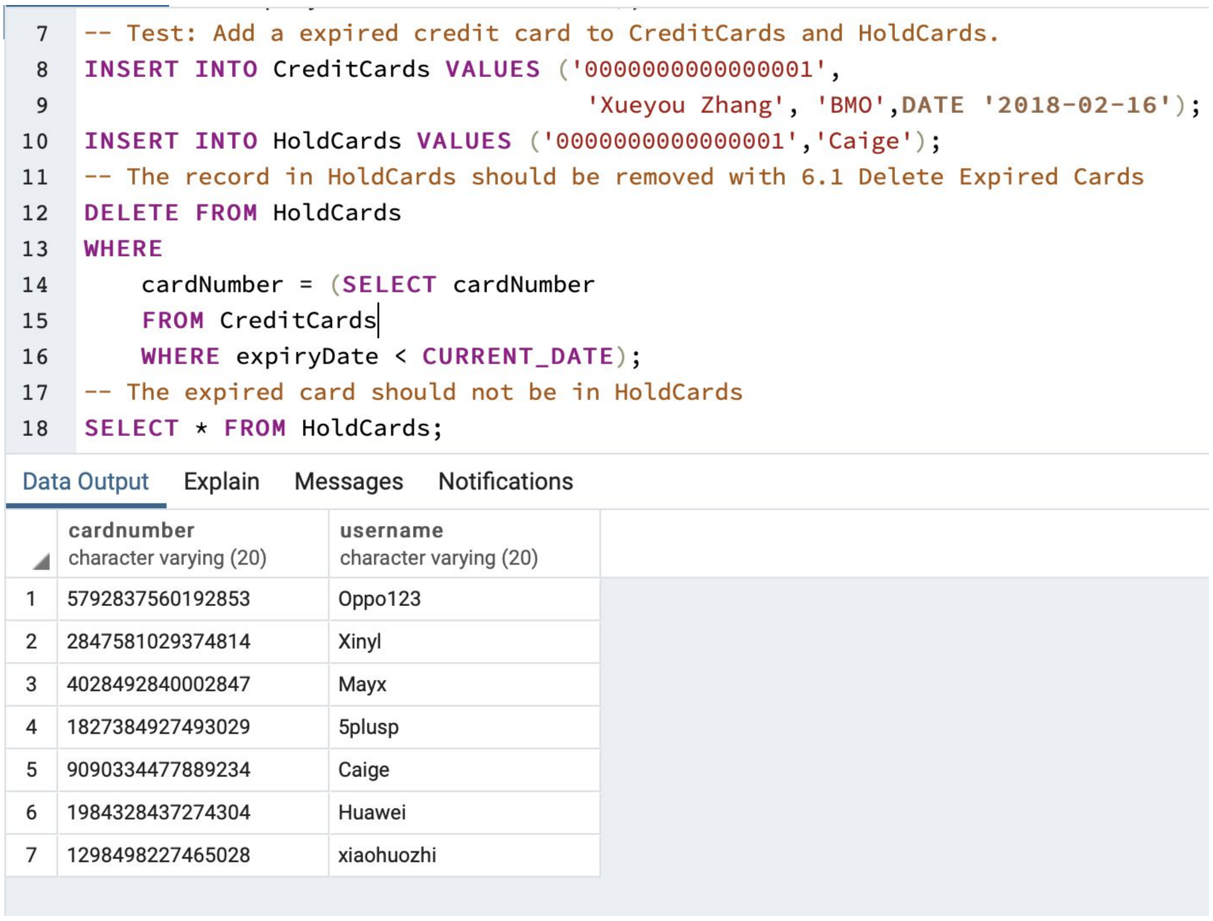
6. Data Modification (part6_data_modification.sql)

6.1 Delete Expired Cards

The wallet function is designed for passengers to manage their payment methods by adding, modifying or deleting credit cards. This command is to make sure all credit cards are valid and not expired. A expired credit card will be removed from HoldCards table, so it is not be shown in the cardholder's (passenger) wallet and cannot be used anymore. However, the credit card will still be in the CreditCards table and we can check payment history on this credit card.



```sql
-- 6.1 Delete Expired Cards
DELETE FROM HoldCards
WHERE
    cardNumber = (SELECT cardNumber
    FROM CreditCards
    WHERE expiryDate < CURRENT_DATE);
```

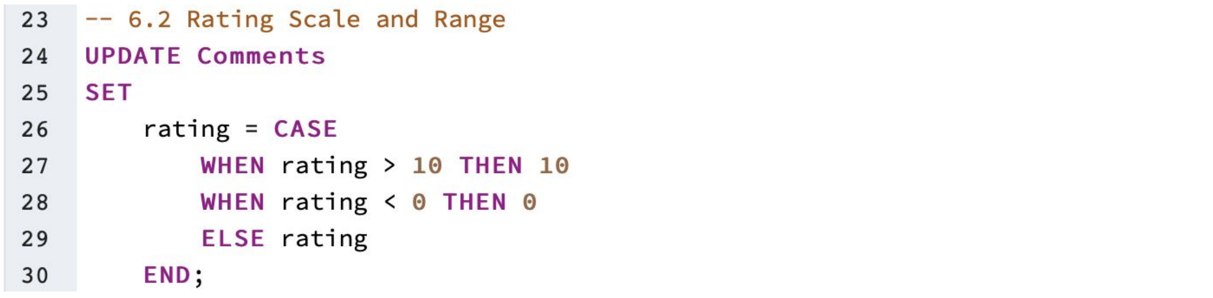


```sql
-- Test: Add a expired credit card to CreditCards and HoldCards.
INSERT INTO CreditCards VALUES ('0000000000000001',
                                  'Xueyou Zhang', 'BMO',DATE '2018-02-16');
INSERT INTO HoldCards VALUES ('0000000000000001','Caige');
-- The record in HoldCards should be removed with 6.1 Delete Expired Cards
DELETE FROM HoldCards
WHERE
    cardNumber = (SELECT cardNumber
    FROM CreditCards
    WHERE expiryDate < CURRENT_DATE);
-- The expired card should not be in HoldCards
SELECT * FROM HoldCards;
```

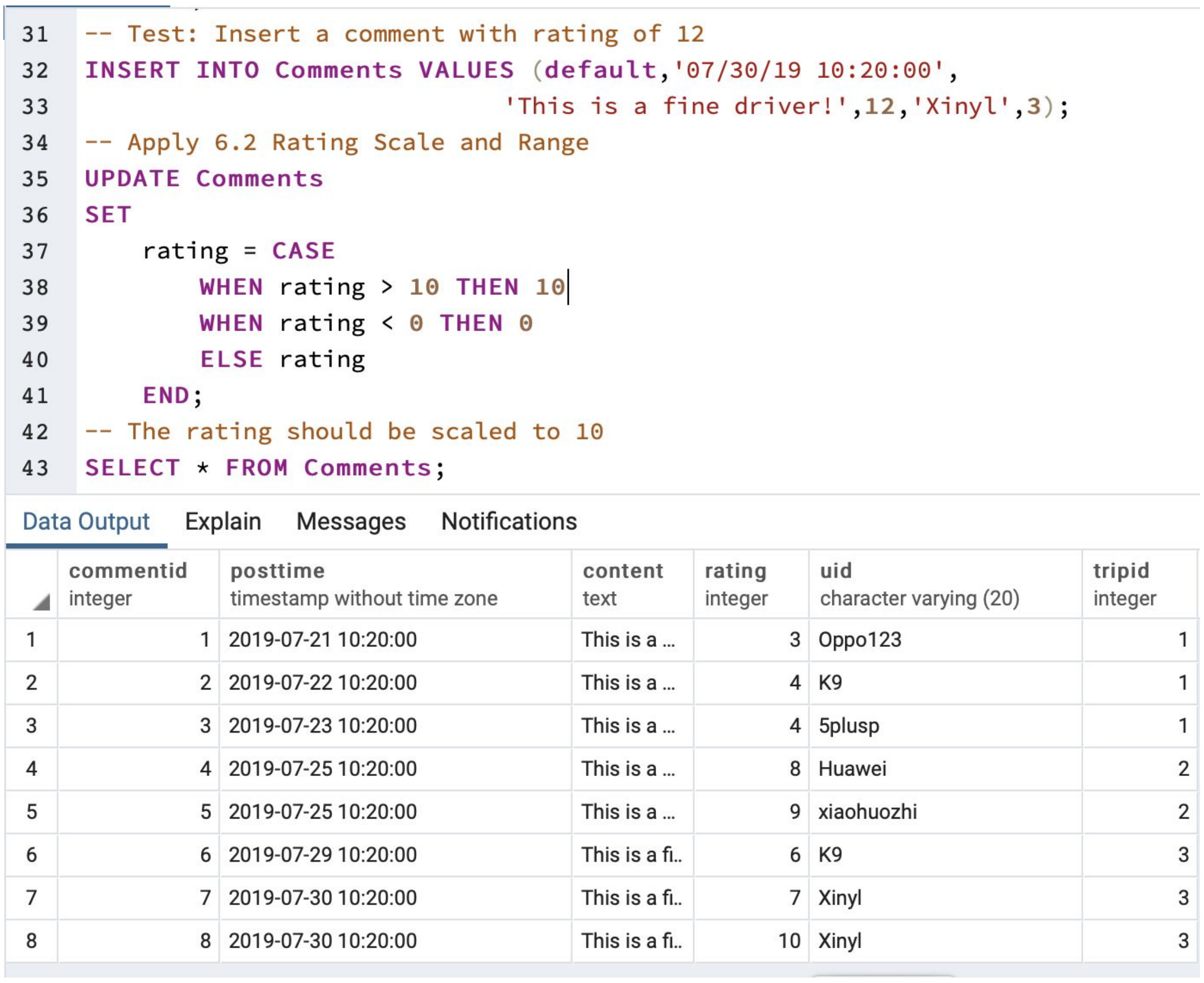
	cardnumber character varying (20)	username character varying (20)
1	5792837560192853	Oppo123
2	2847581029374814	Xinyl
3	4028492840002847	Mayx
4	1827384927493029	5plusp
5	9090334477889234	Caige
6	1984328437274304	Huawei
7	1298498227465028	xiaohuozhi

6.2 Rating Scale and Range

A passenger may leave a comment for a trip with a rating. The rating is an important for us to evaluate customer satisfaction, so we want to unify the rating scale to 0-10. Therefore, this command will adjust all ratings that are out of range: a rating lower than 0 will be adjusted to 0 and a rating greater than 10 will be adjusted to 10.



```sql
-- 6.2 Rating Scale and Range
UPDATE Comments
SET
    rating = CASE
        WHEN rating > 10 THEN 10
        WHEN rating < 0 THEN 0
        ELSE rating
    END;
```



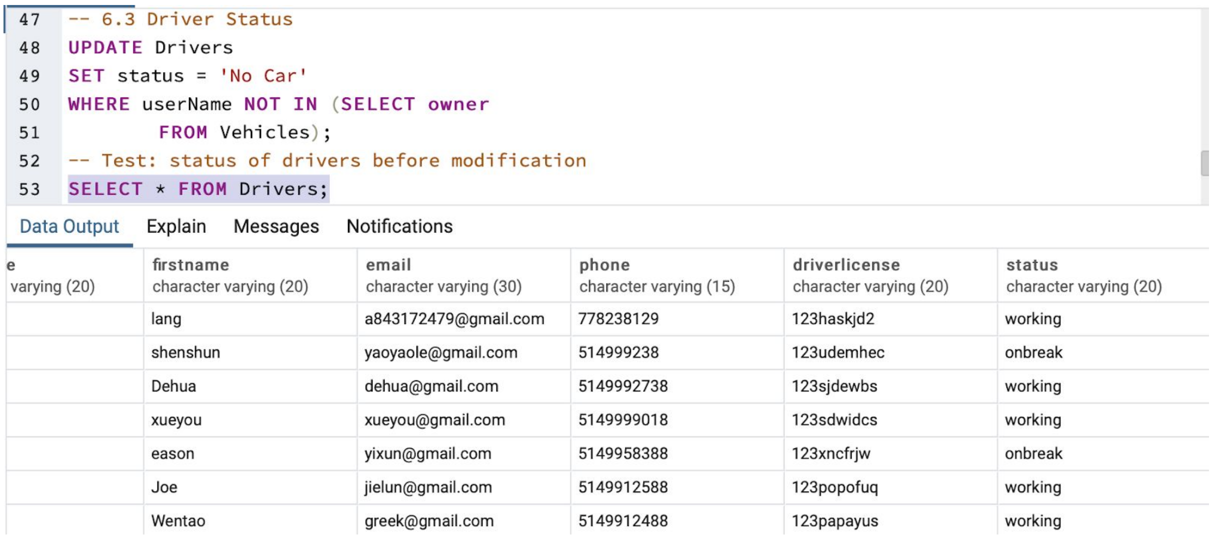
```sql
-- Test: Insert a comment with rating of 12
INSERT INTO Comments VALUES (default,'07/30/19 10:20:00',
                             'This is a fine driver!',12,'Xinyl',3);
-- Apply 6.2 Rating Scale and Range
UPDATE Comments
SET
    rating = CASE
        WHEN rating > 10 THEN 10
        WHEN rating < 0 THEN 0
        ELSE rating
    END;
-- The rating should be scaled to 10
SELECT * FROM Comments;
```

Data Output | Explain | Messages | Notifications

	commentid integer	posttime timestamp without time zone	content text	rating integer	uid character varying (20)	tripid integer
1	1	2019-07-21 10:20:00	This is a ...	3	Oppo123	1
2	2	2019-07-22 10:20:00	This is a ...	4	K9	1
3	3	2019-07-23 10:20:00	This is a ...	4	5plusp	1
4	4	2019-07-25 10:20:00	This is a ...	8	Huawei	2
5	5	2019-07-25 10:20:00	This is a ...	9	xiaohuozhi	2
6	6	2019-07-29 10:20:00	This is a fi..	6	K9	3
7	7	2019-07-30 10:20:00	This is a fi..	7	Xinyl	3
8	8	2019-07-30 10:20:00	This is a fi..	10	Xinyl	3

6.3 Driver Status

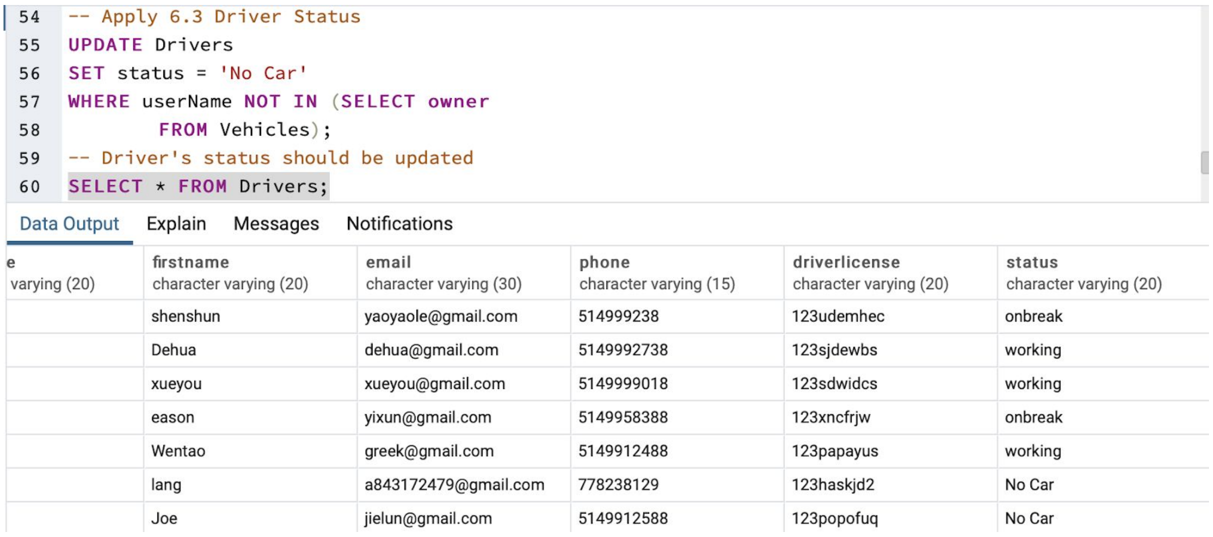
A driver must add a car before creating any trips, so we want to notify drivers to add cars by identifying their status. This command will change the status of all drivers without car to ‘No Car’ to notify them to add a car.



```
-- 6.3 Driver Status
UPDATE Drivers
SET status = 'No Car'
WHERE userName NOT IN (SELECT owner
            FROM Vehicles);
-- Test: status of drivers before modification
SELECT * FROM Drivers;
```

Data Output   Explain   Messages   Notifications

e	firstname	email	phone	driverlicense	status
varying (20)	character varying (20)	character varying (30)	character varying (15)	character varying (20)	character varying (20)
	lang	a843172479@gmail.com	778238129	123haskjd2	working
	shenshun	yaoyaole@gmail.com	514999238	123udemhec	onbreak
	Dehua	dehua@gmail.com	5149992738	123sjdewbs	working
	xueyou	xueyou@gmail.com	5149999018	123sdwidcs	working
	eason	yixun@gmail.com	5149958388	123xncfrjw	onbreak
	Joe	jielun@gmail.com	5149912588	123popofuq	working
	Wentao	greek@gmail.com	5149912488	123papayus	working



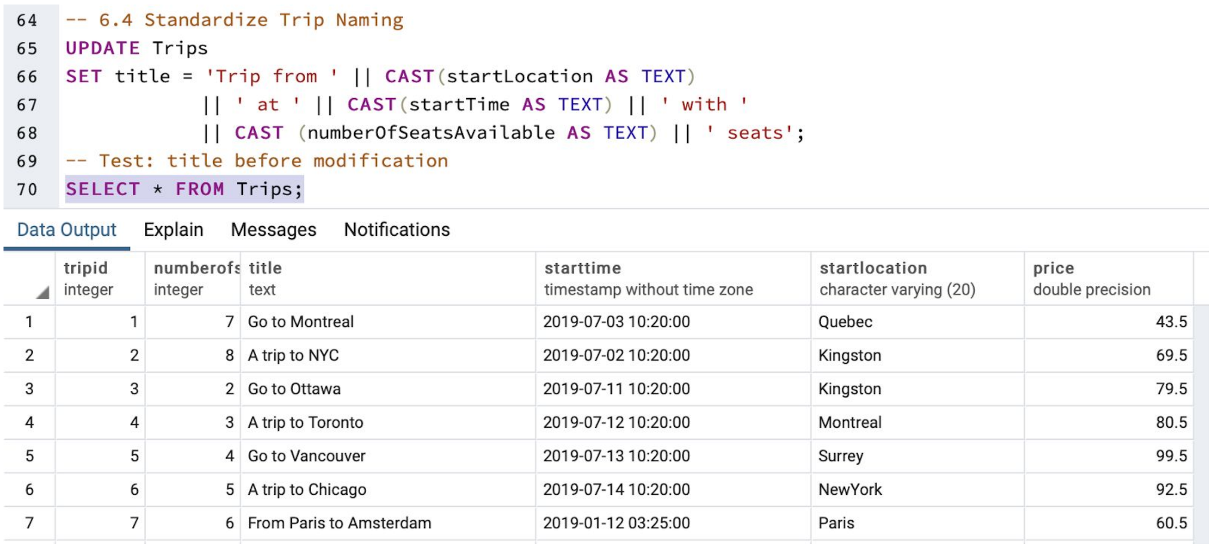
```
-- Apply 6.3 Driver Status
UPDATE Drivers
SET status = 'No Car'
WHERE userName NOT IN (SELECT owner
            FROM Vehicles);
-- Driver's status should be updated
SELECT * FROM Drivers;
```

Data Output   Explain   Messages   Notifications

e	firstname	email	phone	driverlicense	status
varying (20)	character varying (20)	character varying (30)	character varying (15)	character varying (20)	character varying (20)
	shenshun	yaoyaole@gmail.com	514999238	123udemhec	onbreak
	Dehua	dehua@gmail.com	5149992738	123sjdewbs	working
	xueyou	xueyou@gmail.com	5149999018	123sdwidcs	working
	eason	yixun@gmail.com	5149958388	123xncfrjw	onbreak
	Wentao	greek@gmail.com	5149912488	123papayus	working
	lang	a843172479@gmail.com	778238129	123haskjd2	No Car
	Joe	jielun@gmail.com	5149912588	123popofuq	No Car

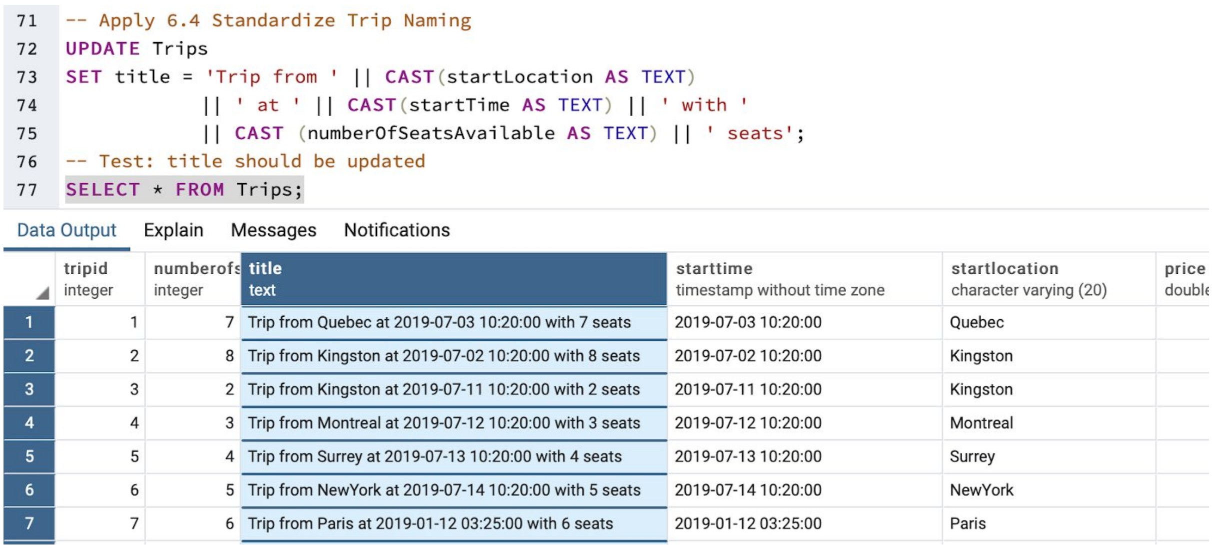
6.4 Standardize Trip Naming

Standardized and clear naming makes it easier for passengers to find a trip they want to join. Thus, this command can change trip names to a format of ‘Trip from [startLocation] at [startTime] with [numberOfSeatsAvailable] seats’ (content in ‘[]’ corresponds to the record).



```sql
-- 6.4 Standardize Trip Naming
UPDATE Trips
SET title = 'Trip from ' || CAST(startLocation AS TEXT)
            || ' at ' || CAST(startTime AS TEXT) || ' with '
            || CAST (numberOfSeatsAvailable AS TEXT) || ' seats';
-- Test: title before modification
SELECT * FROM Trips;
```

	tripid integer	numberofs integer	title text	starttime timestamp without time zone	startlocation character varying (20)	price double precision
1	1	7	Go to Montreal	2019-07-03 10:20:00	Quebec	43.5
2	2	8	A trip to NYC	2019-07-02 10:20:00	Kingston	69.5
3	3	2	Go to Ottawa	2019-07-11 10:20:00	Kingston	79.5
4	4	3	A trip to Toronto	2019-07-12 10:20:00	Montreal	80.5
5	5	4	Go to Vancouver	2019-07-13 10:20:00	Surrey	99.5
6	6	5	A trip to Chicago	2019-07-14 10:20:00	NewYork	92.5
7	7	6	From Paris to Amsterdam	2019-01-12 03:25:00	Paris	60.5



```sql
-- Apply 6.4 Standardize Trip Naming
UPDATE Trips
SET title = 'Trip from ' || CAST(startLocation AS TEXT)
            || ' at ' || CAST(startTime AS TEXT) || ' with '
            || CAST (numberOfSeatsAvailable AS TEXT) || ' seats';
-- Test: title should be updated
SELECT * FROM Trips;
```

	tripid integer	numberofs integer	title text	starttime timestamp without time zone	startlocation character varying (20)	price double
1	1	7	Trip from Quebec at 2019-07-03 10:20:00 with 7 seats	2019-07-03 10:20:00	Quebec	
2	2	8	Trip from Kingston at 2019-07-02 10:20:00 with 8 seats	2019-07-02 10:20:00	Kingston	
3	3	2	Trip from Kingston at 2019-07-11 10:20:00 with 2 seats	2019-07-11 10:20:00	Kingston	
4	4	3	Trip from Montreal at 2019-07-12 10:20:00 with 3 seats	2019-07-12 10:20:00	Montreal	
5	5	4	Trip from Surrey at 2019-07-13 10:20:00 with 4 seats	2019-07-13 10:20:00	Surrey	
6	6	5	Trip from NewYork at 2019-07-14 10:20:00 with 5 seats	2019-07-14 10:20:00	NewYork	
7	7	6	Trip from Paris at 2019-01-12 03:25:00 with 6 seats	2019-01-12 03:25:00	Paris	

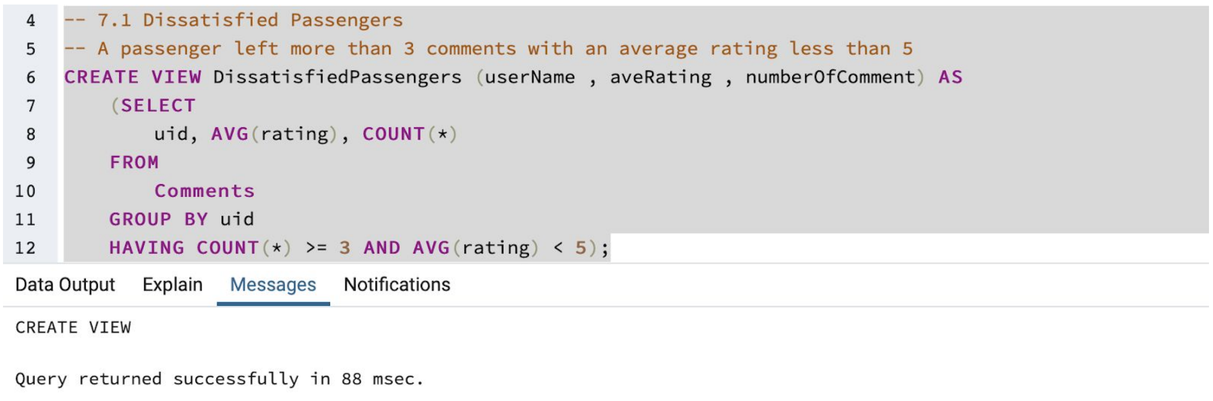
7. Views (part7_VIEW_create.sql)

A view is not datable in the following conditions:

1. The view must have exactly one entry (table or another view) in the FROM clause.
2. The defining query must not contain any one of the following clauses: GROUP BY, HAVING, LIMIT, OFFSET, DISTINCT, WITH, UNION, INTERSECT, and EXCEPT.
3. The selection list cannot contain any window function, set-returning function, or aggregate function.

7.1 Dissatisfied Passengers

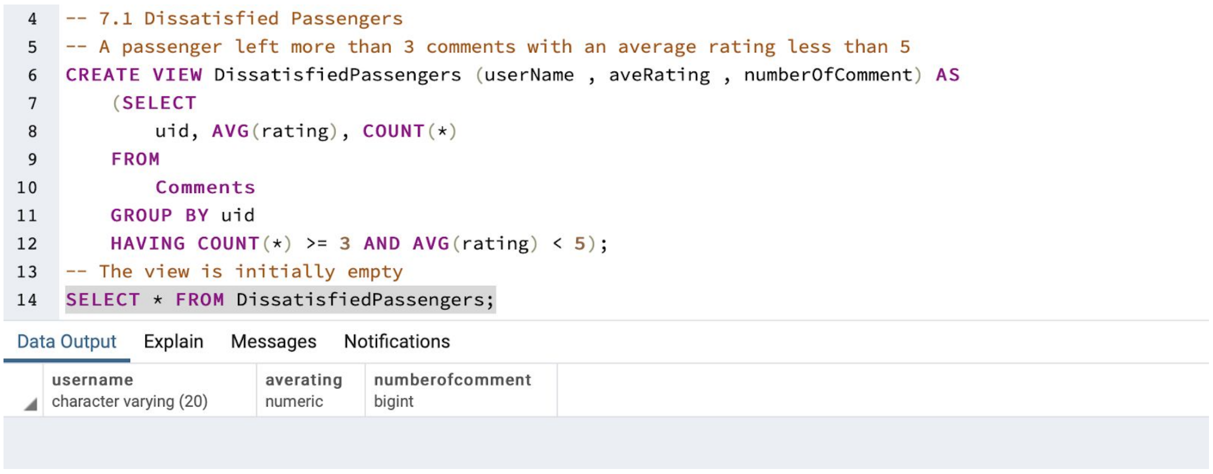
The feedback from passengers is valuable for us, so we want to know what we can do to improve our services. This view can track passengers who have left more than 3 comments with an average rating of less than 5. By combining the view with other queries, we can find out more about why they are dissatisfied and invite them to help us.



```
-- 7.1 Dissatisfied Passengers
-- A passenger left more than 3 comments with an average rating less than 5
CREATE VIEW DissatisfiedPassengers (userName , aveRating , numberOfComment) AS
    (SELECT
        uid, AVG(rating), COUNT(*)
    FROM
        Comments
    GROUP BY uid
    HAVING COUNT(*) >= 3 AND AVG(rating) < 5);
```

Data Output Explain Messages Notifications

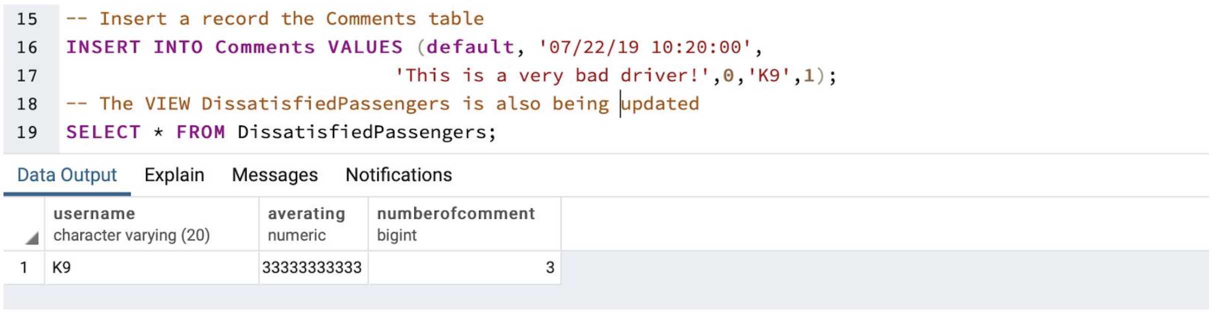
```
CREATE VIEW

Query returned successfully in 88 msec.
```



```
-- 7.1 Dissatisfied Passengers
-- A passenger left more than 3 comments with an average rating less than 5
CREATE VIEW DissatisfiedPassengers (userName , aveRating , numberOfComment) AS
    (SELECT
        uid, AVG(rating), COUNT(*)
    FROM
        Comments
    GROUP BY uid
    HAVING COUNT(*) >= 3 AND AVG(rating) < 5);
-- The view is initially empty
SELECT * FROM DissatisfiedPassengers;
```

Data Output Explain Messages Notifications

username character varying (20)	averating numeric	numberofcomment bigint

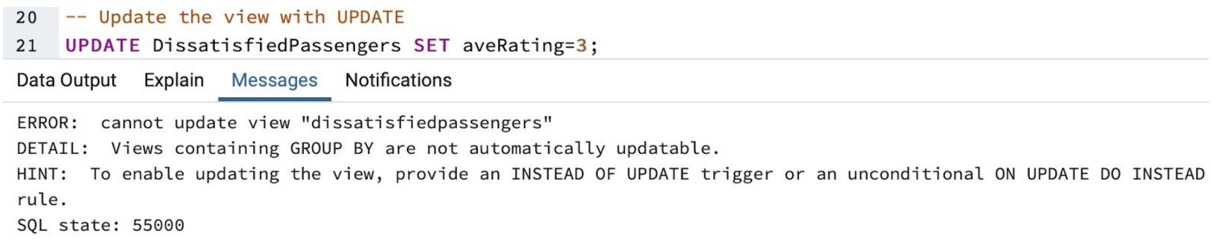


```
-- Insert a record the Comments table
INSERT INTO Comments VALUES (default, '07/22/19 10:20:00',
                               'This is a very bad driver!',0,'K9',1);
-- The VIEW DissatisfiedPassengers is also being updated
SELECT * FROM DissatisfiedPassengers;
```

Data Output | Explain | Messages | Notifications

	username character varying (20)	averating numeric	numberofcomment bigint
1	K9	33333333333	3

Note: This view cannot be updated with a UPDATE statement, but we can create a RULE to achieve data modification on this view. The reason is that GROUP BY clauses at the top level.



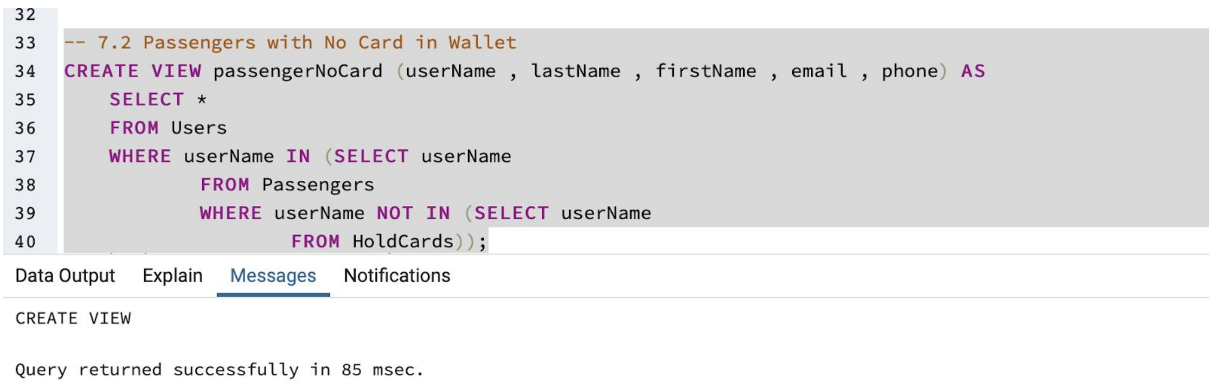
```
-- Update the view with UPDATE
UPDATE DissatisfiedPassengers SET aveRating=3;
```

Data Output | Explain | Messages | Notifications

```
ERROR:  cannot update view "dissatisfiedpassengers"
DETAIL:  Views containing GROUP BY are not automatically updatable.
HINT:  To enable updating the view, provide an INSTEAD OF UPDATE trigger or an unconditional ON UPDATE DO INSTEAD
rule.
SQL state: 55000
```

7.2 Passengers with No Card in Wallet

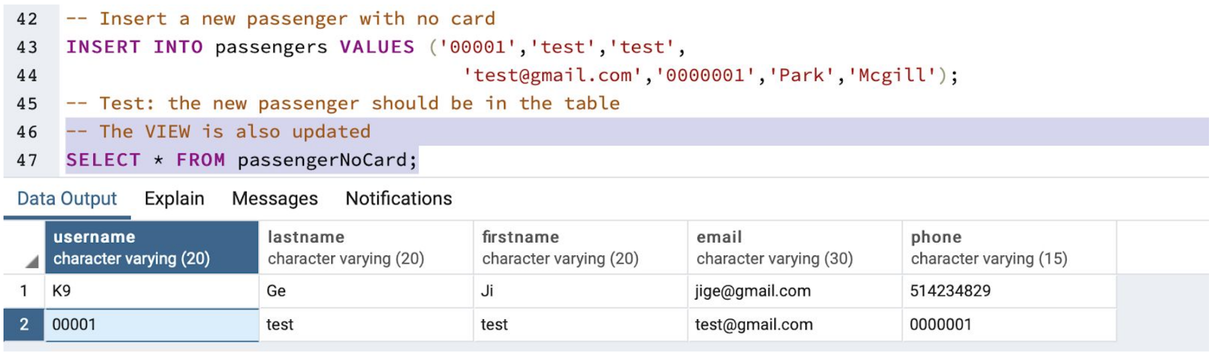
Since the credit card is the most convenient method of payment, we highly recommend passengers add credit cards to wallets. This view is used to track passengers who haven't added any credit cards, so we can address their concerns.



```

-- 7.2 Passengers with No Card in Wallet
CREATE VIEW passengerNoCard (userName , lastName , firstName , email , phone) AS
    SELECT *
    FROM Users
    WHERE userName IN (SELECT userName
            FROM Passengers
            WHERE userName NOT IN (SELECT userName
                    FROM HoldCards));
```

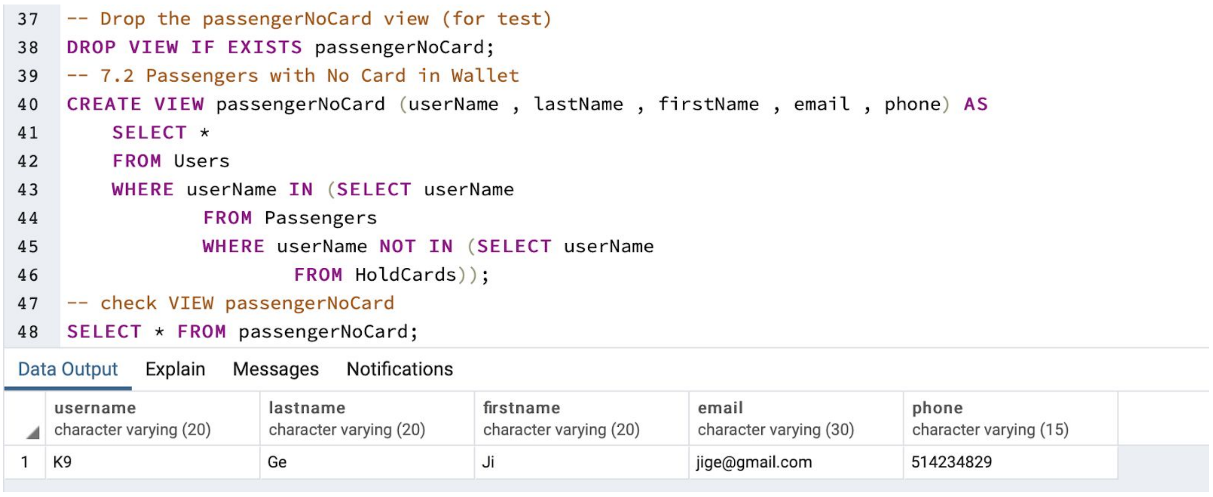
Data Output | Explain | Messages | Notifications

```
CREATE VIEW

Query returned successfully in 85 msec.
```



```sql
-- Insert a new passenger with no card
INSERT INTO passengers VALUES ('00001','test','test',
                                'test@gmail.com','0000001','Park','Mcgill');
-- Test: the new passenger should be in the table
-- The VIEW is also updated
SELECT * FROM passengerNoCard;
```

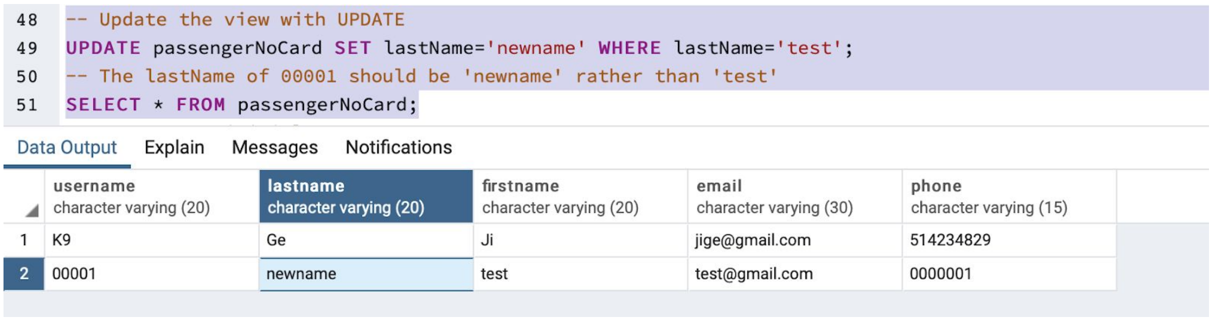
	username character varying (20)	lastname character varying (20)	firstname character varying (20)	email character varying (30)	phone character varying (15)
1	K9	Ge	Ji	jige@gmail.com	514234829
2	00001	test	test	test@gmail.com	0000001



```sql
-- Drop the passengerNoCard view (for test)
DROP VIEW IF EXISTS passengerNoCard;
-- 7.2 Passengers with No Card in Wallet
CREATE VIEW passengerNoCard (userName , lastName , firstName , email , phone) AS
    SELECT *
    FROM Users
    WHERE userName IN (SELECT userName
            FROM Passengers
            WHERE userName NOT IN (SELECT userName
                    FROM HoldCards));
-- check VIEW passengerNoCard
SELECT * FROM passengerNoCard;
```

	username character varying (20)	lastname character varying (20)	firstname character varying (20)	email character varying (30)	phone character varying (15)
1	K9	Ge	Ji	jige@gmail.com	514234829

Note: This view can be updated with a UPDATE statement. By executing the UPDATE query, the lastName of user ‘00001’ is changed. The Passenger table is also updated. The reason is that the view have exactly one entry (table or another view) in the FROM clause, not contain any one of the following clauses: GROUP BY, HAVING, LIMIT, OFFSET, DISTINCT, WITH, UNION, INTERSECT, and EXCEPT and not contain any window function, set-returning function, or aggregate function.



```sql
-- Update the view with UPDATE
UPDATE passengerNoCard SET lastName='newname' WHERE lastName='test';
-- The lastName of 00001 should be 'newname' rather than 'test'
SELECT * FROM passengerNoCard;
```

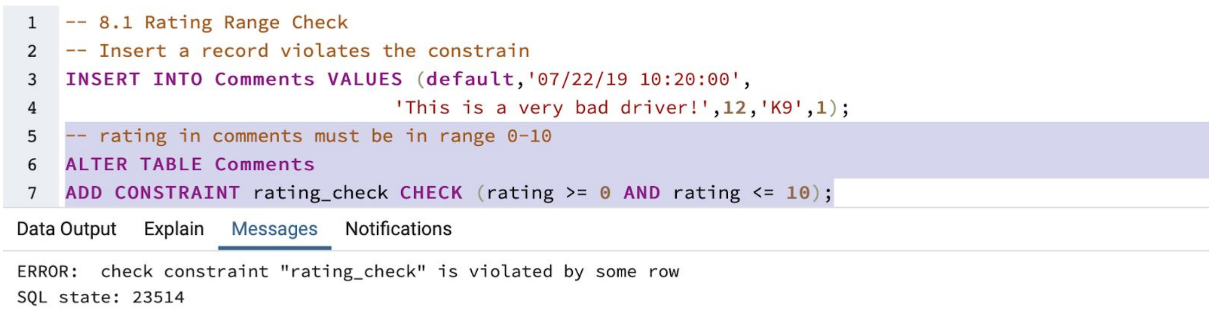
	username character varying (20)	lastname character varying (20)	firstname character varying (20)	email character varying (30)	phone character varying (15)
1	K9	Ge	Ji	jige@gmail.com	514234829
2	00001	newname	test	test@gmail.com	0000001

8. Check Constraints (part8_CHECK_constraints.sql)

8.1 Rating Range Check

As mentioned in 6.2, we need to unify the rating scale to 0-10 and keep the scale for new comments. Thus, this check ensures that all new comments with a rating in range 0-10.

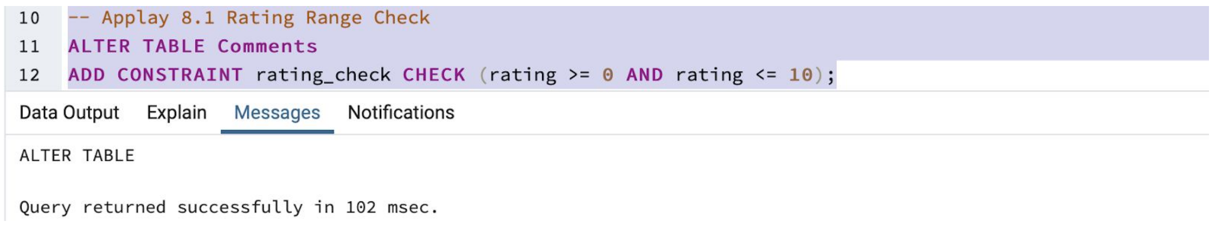
Note: Cannot add constraint if there is a violation.



```
-- 8.1 Rating Range Check
-- Insert a record violates the constrain
INSERT INTO Comments VALUES (default,'07/22/19 10:20:00',
                                'This is a very bad driver!',12,'K9',1);
-- rating in comments must be in range 0-10
ALTER TABLE Comments
ADD CONSTRAINT rating_check CHECK (rating >= 0 AND rating <= 10);
```

```
ERROR:  check constraint "rating_check" is violated by some row
SQL state: 23514
```

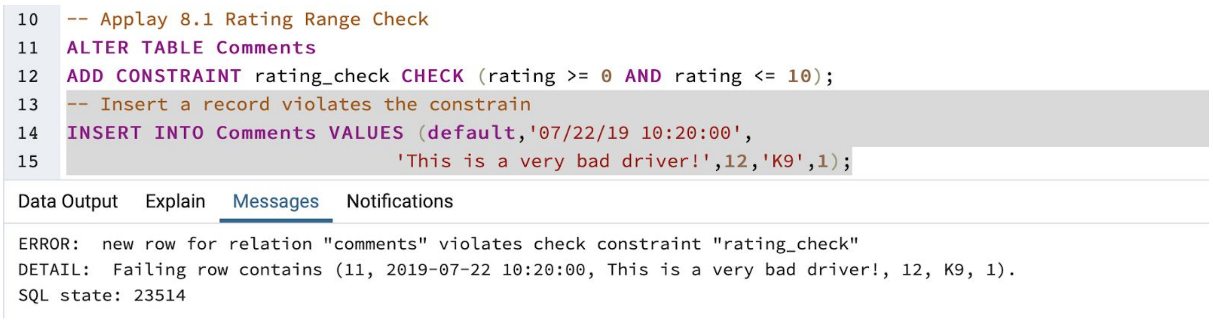
Note: Successful execution



```
-- Apply 8.1 Rating Range Check
ALTER TABLE Comments
ADD CONSTRAINT rating_check CHECK (rating >= 0 AND rating <= 10);
```

```
ALTER TABLE

Query returned successfully in 102 msec.
```

Note: After applying constraints, violation cannot be inserted.



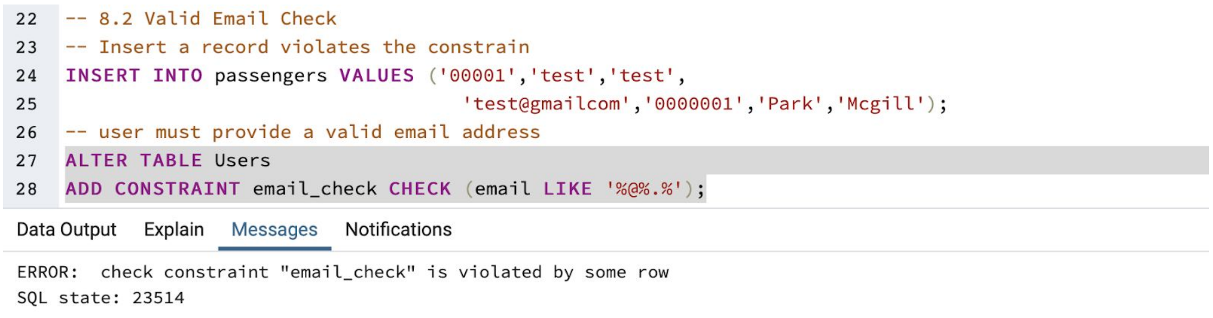
```
-- Apply 8.1 Rating Range Check
ALTER TABLE Comments
ADD CONSTRAINT rating_check CHECK (rating >= 0 AND rating <= 10);
-- Insert a record violates the constrain
INSERT INTO Comments VALUES (default,'07/22/19 10:20:00',
                                'This is a very bad driver!',12,'K9',1);
```

```
ERROR:  new row for relation "comments" violates check constraint "rating_check"
DETAIL:  Failing row contains (11, 2019-07-22 10:20:00, This is a very bad driver!, 12, K9, 1).
SQL state: 23514
```

8.2 Valid Email Check

The email address is important for us to contact customers, so we must ask customers register with valid email addresses. This check is to ensure that a provided email address has a format of ‘[1]@[2].[3]’ ([1], [2] and [3] can be any string, i.e. ‘example@mail.com’). The check can only perform basic format check; more complex verification requires verification code and email services.

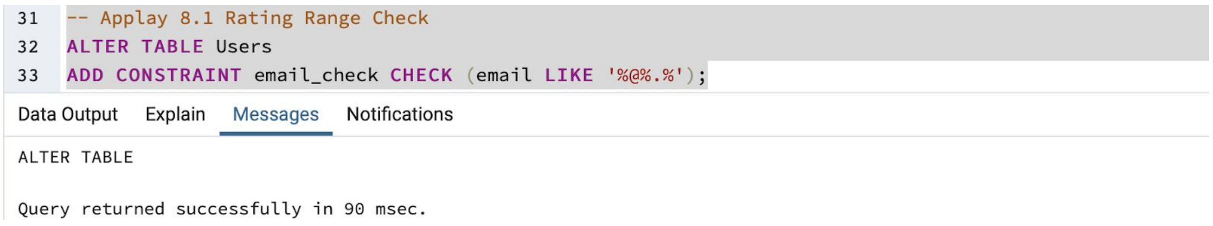
Note: Cannot add constraint if there is a violation.



```sql
-- 8.2 Valid Email Check
-- Insert a record violates the constrain
INSERT INTO passengers VALUES ('00001','test','test',
                                'test@gmailcom','0000001','Park','Mcgill');
-- user must provide a valid email address
ALTER TABLE Users
ADD CONSTRAINT email_check CHECK (email LIKE '%@%.%');
```

```
ERROR:  check constraint "email_check" is violated by some row
SQL state: 23514
```

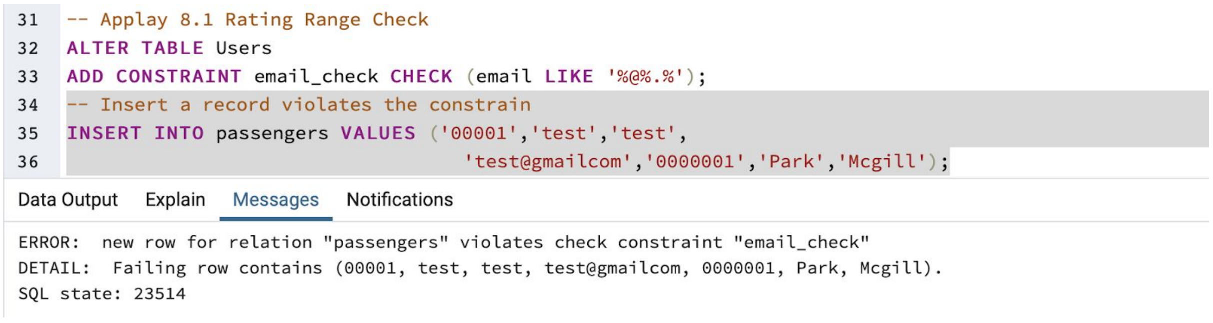
Note: Successful execution



```sql
-- Applay 8.1 Rating Range Check
ALTER TABLE Users
ADD CONSTRAINT email_check CHECK (email LIKE '%@%.%');
```

```
ALTER TABLE

Query returned successfully in 90 msec.
```

Note: After applying constraints, violation cannot be inserted.



```sql
-- Applay 8.1 Rating Range Check
ALTER TABLE Users
ADD CONSTRAINT email_check CHECK (email LIKE '%@%.%');
-- Insert a record violates the constrain
INSERT INTO passengers VALUES ('00001','test','test',
                                'test@gmailcom','0000001','Park','Mcgill');
```

```
ERROR:  new row for relation "passengers" violates check constraint "email_check"
DETAIL:  Failing row contains (00001, test, test, test@gmailcom, 0000001, Park, Mcgill).
SQL state: 23514
```

9. Creativity

9.1 Automated Data Generation and Real Data Set (part9_RealWorldDataset)

Approach for generating data was done for table Admin and Cities, for there are real city and real names from real world. Names are from a dataset of Indian male names (available from: https://gist.github.com/mbejda/7f86ca901fe41bc14a63) and cities are from dataset of World cities (available from: https://simplemaps.com/data/world-cities).

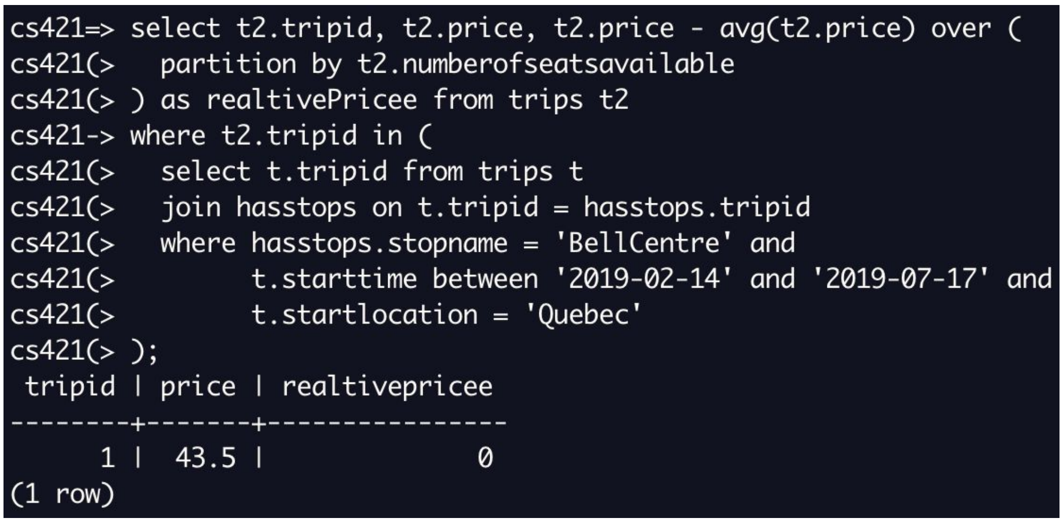
To simplify, only names with two english words are selected to generate name list, and set data structure was used to eliminate duplicate names. To make it real, we only select Indian cities (of course, only Indian Mayor can manage Indian city). For the phone number of the Admins, they are auto-generated random 10-digit numbers. Duplication is ignored, for there is 10^(-10) probability of getting the same phone number.

To generate data for 2 tables, we add real names, generated email and phone for Admin and real city names for Cities.
All of these was implemented using psycopg2, a python postgresql driver (http://initd.org/psycopg/docs/).

9.2 Cool SQL features (part9_coolSQL.sql)

```sql
select t2.tripid, t2.price, t2.price - avg(t2.price) over (
 partition by t2.numberofseatsavailable
) as realtivePricee from trips t2
where t2.tripid in (
 select t.tripid from trips t
 join hasstops on t.tripid = hasstops.tripid
 where hasstops.stopname = 'BellCentre' and
       t.starttime between '2019-02-14' and '2019-07-17' and
       t.startlocation = 'Quebec'
);
```

This query uses WINDOW FUNCTION that computes relative price of searched trips. In this case, a trip start at “Quebec” and ends at “BellCentre”, time between Feb 14 2019 and July 17 2019 was searched. Relative price was determined by the difference between price and the average price of the group (grouped by number of seats in a vehicle), for the price may be affected by size of vehicle.
Output:

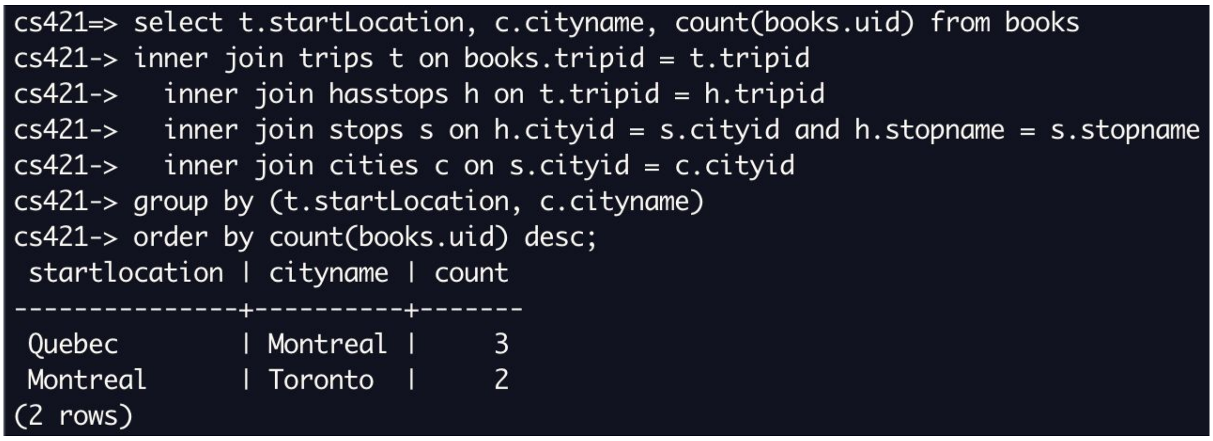


```
cs421=> select t2.tripid, t2.price, t2.price - avg(t2.price) over (
cs421(>   partition by t2.numberofseatsavailable
cs421(> ) as realtivePricee from trips t2
cs421-> where t2.tripid in (
cs421(>   select t.tripid from trips t
cs421(>   join hasstops on t.tripid = hasstops.tripid
cs421(>   where hasstops.stopname = 'BellCentre' and
cs421(>         t.starttime between '2019-02-14' and '2019-07-17' and
cs421(>         t.startlocation = 'Quebec'
cs421(> );
 tripid | price | realtivepricee
--------+-------+----------------
      1 |  43.5 |              0
(1 row)
```

9.3 Complex Analytical Query

```sql
select t.startLocation, c.cityname, count(books.uid) from books
inner join trips t on books.tripid = t.tripid
 inner join hasstops h on t.tripid = h.tripid
 inner join stops s on h.cityid = s.cityid and h.stopname = s.stopname
 inner join cities c on s.cityid = c.cityid
group by (t.startLocation, c.cityname)
order by count(books.uid) desc;
```

Business case
As an manager, I would like to find the best route (from start location to a city) based on count of passengers on that route.
This query computes the most popular route (from start location to a city), determined by count of passengers from a start location to a stop city.
Description
A inner join was done among trips, books, has-stops, stops, and cities. This multi way joined relation was then grouped by start location and stop city, and a count of bookings was done for each group to generate a popularity score for raking.
Output:



```
cs421=> select t.startLocation, c.cityname, count(books.uid) from books
cs421-> inner join trips t on books.tripid = t.tripid
cs421->   inner join hasstops h on t.tripid = h.tripid
cs421->   inner join stops s on h.cityid = s.cityid and h.stopname = s.stopname
cs421->   inner join cities c on s.cityid = c.cityid
cs421-> group by (t.startLocation, c.cityname)
cs421-> order by count(books.uid) desc;
 startlocation | cityname | count
---------------+----------+-------
 Quebec        | Montreal |     3
 Montreal      | Toronto  |     2
(2 rows)
```