Homework 3

part 1: joins and subqueries

1a

```
select user_id, trip_id, trip_length, (1 + trip_length * 0.15)::decimal(10,2) as trip_cost
from (
    select ts.user_id, ts.trip_id,
    case
    when te.time is null then 1440
    else ceiling((extract(epoch from te.time) - extract(epoch from ts.time))::decimal / 60)
    end trip_length
    from trip_start ts
    left join trip_end te on ts.trip_id = te.trip_id
) trip_time
order by trip_id
limit 10;
```

```
user_id | trip_id | trip_length | trip_cost
---------+---------+-------------+-----------
   20685 |       0 |           2 |      1.30
   34808 |       2 |           3 |      1.45
   25463 |       3 |        1440 |    217.00
   26965 |       4 |           2 |      1.30
     836 |       5 |           1 |      1.15
    3260 |       6 |           5 |      1.75
   13850 |       7 |           3 |      1.45
   23528 |       9 |           4 |      1.60
   35829 |      11 |        1440 |    217.00
   18494 |      12 |        1440 |    217.00
(10 rows)
```

1b

```
select user_id, sum(trip_cost)::decimal(10,2) as total_spent
from (
select user_id, trip_id, trip_length, (1 + trip_length * 0.15)::decimal(10,2) as trip_cost
    from (
        select ts.user_id, ts.trip_id,
        case
        when te.time is null then 1440
        else ceiling((extract(epoch from te.time) - extract(epoch from ts.time))::decimal / 60)
        end trip_length
        from trip_start ts
        left join trip_end te on ts.trip_id = te.trip_id
    ) trip_time
) a
group by user_id
order by user_id
limit 10;
```

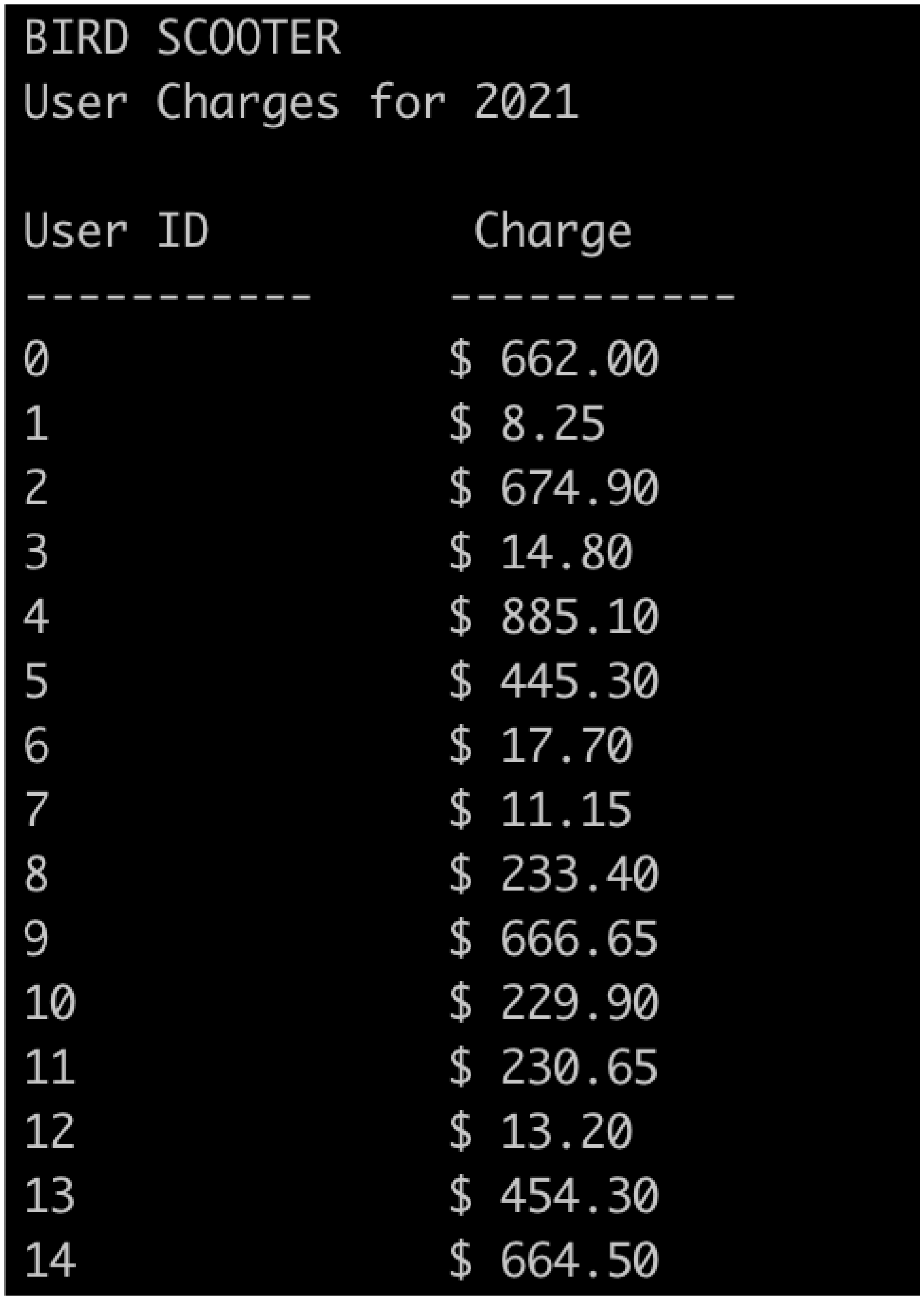
```
user_id | total_spent
---------+-------------
       0 |      662.00
       1 |        8.25
       2 |      674.90
       3 |       14.80
       4 |      885.10
       5 |      445.30
       6 |       17.70
       7 |       11.15
       8 |      233.40
       9 |      666.65
(10 rows)
```

1c

Left, Equi, Outer join

1d

```
import psycopg2

connection = psycopg2.connect("dbname=cs143 user=root password=cs143Rocks! host=localhost")
cur = connection.cursor()

query = '''
select user_id, sum(trip_cost)::decimal(10,2) as total_spend
from (
select user_id, trip_id, trip_length, (1 + trip_length * 0.15)::decimal(10,2) as trip_cost
    from (
        select ts.user_id, ts.trip_id,
        case
        when te.time is null then 1440
        else ceiling((extract(epoch from te.time) - extract(epoch from ts.time))::decimal / 60)
        end trip_length
        from trip_start ts
        left join trip_end te on ts.trip_id = te.trip_id
    ) trip_time
) a
group by user_id
order by user_id
'''

cur.execute(query)

rows = cur.fetchall()

print("BIRD SCOOTER")
print("User Charges for 2021\n")
print("User ID",'\t',"Charge")
print('-'*11,'-'*11,sep='\t')

for user_id, charge in rows:
  print(user_id, f'$ {charge}',sep='\t'*2)

connection.close()
```



```
BIRD SCOOTER
User Charges for 2021

User ID          Charge
----------      ----------
0               $ 662.00
1               $ 8.25
2               $ 674.90
3               $ 14.80
4               $ 885.10
5               $ 445.30
6               $ 17.70
7               $ 11.15
8               $ 233.40
9               $ 666.65
10              $ 229.90
11              $ 230.65
12              $ 13.20
13              $ 454.30
14              $ 664.50
```

part 2: views and authorization

2a

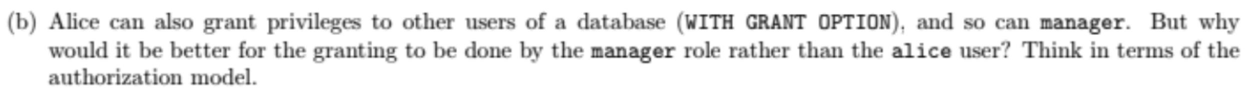
(a) In lecture, we discussed that RDBMS often grant authorizations based on users or *roles*, which are groups, and a user may be a member of zero or more groups. Suppose we have a role called **`manager`** and user **`alice`** has this role. First, how would **`alice`** and **`manager`** be represented in the authorization graph? How would the privileges **`INSERT`** and **`SELECT`** be represented?

`manager` would be a node.

`Alice` would be a node which grated the role manager through an edge.

`insert` and `delete` would be edges.

2b

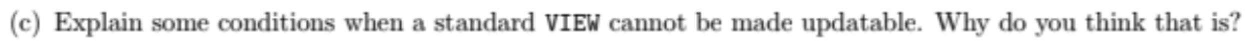


(b) Alice can also grant privileges to other users of a database (`WITH GRANT OPTION`), and so can **`manager`**. But why would it be better for the granting to be done by the **`manager`** role rather than the **`alice`** user? Think in terms of the authorization model.

If `Alice` granted roles then her permissions could only be removed with the `cascade` option and if `restrict` was the option then her role would not be removed.

Also the `manager` should grant the permissions so that they can remove privileges because it would not be able to remove privileges which `Alice` granted.

2c



(c) Explain some conditions when a standard `VIEW` cannot be made updatable. Why do you think that is?

If a standard view consists of `join` or if it has multiple tables in it's `from` or when its `select` uses aggregate functions.

If you tried to update when a view is multiple tables in its `from` or a `join` it would not be good to update it because it would be difficult to decide which original table to update.

You cannot update aggregate functions because they are not in the table, they are computed from the table so you can't change them.