University of California, Los Angeles
Department of Computer Science

Computer Science 143 **Prof. Ryan Rosario**

Homework 2

Solutions.

Part 1: SQL

We will again use the BART dataset. To remind yourself about BART, see Homework 1.

```
-- The schema isn't exactly set up this way in the HW2 tarball since we are not modifying data,
-- we don't need to worry about referential integrity. The proper schema is provided for your reference.
CREATE TABLE station(
    Abbreviation char(4) PRIMARY KEY,
    Location varchar(23) NOT NULL,
    Name varchar(50) NOT NULL
);

CREATE TABLE ridecount(
    Origin char(4) REFERENCES station(Abbreviation),
    Destination char(4) REFERENCES station(Abbreviation),
    Throughput integer,
    TStamp timestamp,
    PRIMARY KEY(Origin, Destination, TStamp)
);
```

Note that this schema is slightly different from Homework 1.

Exercises

(a) Write a query that computes the total number of trips that started at each BART station. Output the `Origin` and total Throughput as `total_throughput`. **To help our grader, also submit the first 10 rows of your output without changing the order of the rows.**

```
SELECT
    Origin,
    SUM(Throughput) AS total_throughput
 FROM ridecount
 GROUP BY Origin;

  origin | total_throughput
--------+------------------
 12TH   |          1323517
 16TH   |          1313113
 19TH   |          1322154
 24TH   |          1336186
 ASHB   |           558777
 BALB   |          1160154
 BAYF   |           586765
 CAST   |           289594
 CIVC   |          2539919
 COLM   |           426739
(10 rows)
```

(b) Write a query that computes the total number of trips that started at each BART station. Only keep stations that had more than 100,000 entries/boardings. Output the station `Name`, `Location` and total Throughput as `total_throughput`.

```
SELECT
    Name,
    Location,
    SUM(Throughput) AS total_throughput
 FROM ridecount L
 JOIN station R
 ON L.Origin = R.Abbreviation
 GROUP BY Name, Location, Origin
 HAVING SUM(Throughput) > 10000;

                 name                 |   location    | total_throughput
--------------------------------------+---------------+------------------
 12th St. Oakland City Center (12TH) | Oakland       |          1323517
 16th St. Mission (16TH)             | San Francisco |          1313113
 19th St. Oakland (19TH)             | Oakland       |          1322154
 24th St. Mission (24TH)             | San Francisco |          1336186
 Ashby (ASHB)                        | Berkeley      |           558777
 Balboa Park (BALB)                  | San Francisco |          1160154
 Bay Fair (BAYF)                     | San Leandro   |           586765
 Castro Valley (CAST)                | Castro Valley |           289594
 Civic Center/UN Plaza (CIVC)        | San Francisco |          2539919
 Coliseum/Oakland Airport (COLS)     | Oakland       |           684904
(10 rows)
```

(c) Write a query that computes movement among cities. That is, compute the total number of trips (`Throughput`) between city A and city B and call it `total_rides`. For example, we want to know how many trips started in Oakland and ended in Fremont, and vice verse. Output the city (`Location`) corresponding to `Origin`, the city `Location` corresponding to `Destination` and `total_rides`. Do not output `Origin` or `Destination`, as that would not make sense. Sort from largest to smallest `total_rides` and **report the first 10 rows without doing any additional ordering.** *Hint:* This is very similar to problem 2b in HW 1.

```
SELECT
    S.Location,
    T.Location,
    SUM(Throughput) AS total_rides
FROM ridecount R
JOIN station S
ON R.Origin = S.Abbreviation
JOIN station T
ON R.Destination = T.Abbreviation
GROUP BY S.Location, T.Location
ORDER BY total_rides DESC
LIMIT 10;

   location    |   location    | total_rides
---------------+---------------+-------------
 San Francisco | San Francisco |     6406085
 San Francisco | Oakland       |     3923381
 Oakland       | San Francisco |     3835810
 San Francisco | Berkeley      |     1113659
 Berkeley      | San Francisco |     1059120
 San Francisco | Walnut Creek  |      924245
 Walnut Creek  | San Francisco |      919046
 Oakland       | Oakland       |      869137
 San Francisco | El Cerrito    |      705177
 El Cerrito    | San Francisco |      650115
(10 rows)
```

(d) Write a query that finds the maximum `Throughput` ever recorded in this dataset, as well as the `Origin` and `Destination` for this trip, and the date/time (`Tstamp`). How can we do it **without** using `LIMIT` and `ORDER BY`? **In addition to your query, please submit the row.**

Use a scalar subquery!

```
SELECT
    Origin,
    Destination,
    Tstamp,
    Throughput
FROM ridecount
WHERE Throughput = (
    SELECT
        MAX(Throughput)
    FROM ridecount
);

 origin | destination |       tstamp        | throughput
--------+-------------+---------------------+------------
 24TH   | CIVC        | 2017-01-21 16:00:00 |       1826
(1 row)
```

Part 2: Relational Algebra

Exercises

(a) Consider the relation `RideCount` from Homework 1. Suppose we filter out all weekend traffic from `RideCount` and create a new relation called `RideCountWeekday` containing information about traffic that occurred on a weekday, since weekend traffic is drastically different the workweek traffic. Write a relational algebra expression that does the following: only keeps tuples from `RideCountWeekday` with non-zero `Throughput`, and with what's remaining, compute the average `Throughput` by `Hour` and route. The output would be 24 tuples per route. **Note:** While ridership may depend on both the day of week and the hour of the day, we are going to ignore the day of the week in this problem.

In this solution, let R be the relation `RideCountWeekday`, T is `Throughput`, O is `Origin`, D is `Destination` and H is `Hour`. First we only keep tuples where `Throughput` is non-zero. Thus:

$$\sigma_{T>0}\left(R\right)$$

Finally, we compute the average throughput by hour of day, to get the following:

$$_{H,O,D}\gamma_{\text{AVG(T)}}\sigma_{T>0}\left(R\right)$$

(b) Suppose the Mayor of San Francisco wants to create a public transit campaign. She wants `Throughput` data for all trips that start in San Francisco and end in San Francisco. The output should be `Origin`, `Destination`, `Date`, `Hour` and `Throughput`, but should only contain tuples where the `Origin` and `Destination` is in San Francisco. Write the relational algebra expression for this problem. You will need to explicitly specify conditions. *Hint:* You need to use both relations.

This problem was a bit more complex than I had intended, but it makes a good point that's important to know. In class we discussed attribute name clashes, and we definitely have that here.

`RideCount` only contains abbreviations for `Origin` and `Destination`. To get the name city location of these stations, we have to join with `Station`. We cannot use a natural join because there are no common attribute names, so we use a theta join, but we have to theta join **twice** once for `Origin` and once for `Destination`. The trick here is that there is a name clash in the attributes *and* in the relations (the latter is more rare), so I rename the second use of S to U (I skip T since that is an attribute name!). Without the rename, we do not know which S in the join each attribute comes from.

$$\Pi_{O,De,Da,H,T}\left(\sigma_{S.L=\text{“San Francisco”}\wedge U.L=\text{“San Francisco”}}\left(R\bowtie_{O=A}S\bowtie_{D=U.A}\rho_{U}\left(S\right)\right)\right)$$

Part 3: SQL Schemas

In Homework 1, we created a relational schema and diagram for the Bird Scooter example. In this problem, we will create a SQL schema using the `CREATE TABLE` syntax. This means we also need to pick the proper data types for each column. For a description of how Bird Scooter works, see Homework 1.

We need a table to represent a `scooter`. Each of the following statements is designed to give you a hint as to the proper data type.

1. Each scooter has an identifier `scooter_id`, a number. Since Bird is a startup, we assume that there are no more than 10,000 scooters.
2. Each scooter has a flag `status` that marks it as online, offline (broken etc.), and lost/stolen. Each scooter can have only one of these states at a time, and must have a state.

We need a table to represent a `customer` (`user` is a system keyword so I will not use it):

1. Each user has an identifier `user_id`, a number, and we assume that Bird has at most 500,000 users for now.
2. A user is just someone that installed the app, not necessarily someone that will use a scooter. Thus, they may, or may not have a credit card number `ccnum` (16 digits) and expiration date `expdate`. Expiration dates usually look like MM/YY, but to make this simpler so you can use a more apparent data type, it is safe to assume that the card expires at midnight (00:00) on the 1st of the month.
3. Each user must have an `email` address. Assume an email address length is at most 100.

We need a table to represent a `trip`. To keep it simpler, we will include start and end information in this table, but the end of trip information may be missing. Each trip is associated with:

1. a unique identifier `trip_id`, a number. Assume that the total number of rides is not small.
2. exactly one user `user_id` and exactly one scooter `scooter_id`.
3. a `start_time` and `end_time`, which includes the date.
4. a `pickup` and `dropoff` location as a GPS coordinate (a latitude/longitude pair). *Hint:* See the documentation here. Note that latitude and longitude together form a point on a Cartesian plane (actually a sphere, but we will assume Cartesian plane for this problem).

Exercise. Write the SQL schema for the tables discussed above using `CREATE TABLE`. Specify a primary key, or composite primary key using the correct syntax. Specify the proper foreign key relationship on each table (if one exists) using the proper syntax. Try to minimize storage space because we can always promote later.

```
CREATE TYPE scooter_status AS ENUM('online', 'offline', 'missing or stolen');
CREATE TABLE scooter (
    scooter_id    smallint PRIMARY KEY,
    status        scooter_status
);

CREATE TABLE customer (
    user_id        int PRIMARY KEY,
    ccnum          varchar(16),
    exp            timestamptz,
    email          varchar(100)
);

 CREATE TABLE trip (
    trip_id          int PRIMARY KEY,
    user_id          int NOT NULL REFERENCES customer(user_id),
    scooter_id       smallint NOT NULL REFERENCES scooter(scooter_id),
    start_time       timestamptz NOT NULL,
    end_time         timestamptz,
    pickup           point NOT NULL,
    dropoff          point
);
```