University of California, Los Angeles
Department of Computer Science

Computer Science 143 **Prof. Ryan Rosario**

Homework 2

Solutions

Please remember the following:

1. Homework is mostly graded on completion. We may grade a few parts, but it will never be the majority of the grade on the assignment. So try your best, and focus on solving the problems. Consider homework (and studying the solutions) as practice for the midterm.
2. Homework must be submitted digitally, on CCLE. We will not do any paper grading. You can use a text file, but if you use Word, a PDF is preferred rather than a DOC file.
3. If there are any exercises that are difficult to do digitally (such as diagrams or math), consider scanning your drawing or math, or using a graphics program (even a readable MS Paint is fine) or Equation Editor.
4. **For the sanity of the grader** we will ask you to run the queries and submit the result. You may lose points if you only provide a query.
5. Solutions will be posted.

Part 1: Text, Joins and Subqueries

For some reason, your instructor has been scraping the Caltrans website every 15 minutes or so, since 2015, to get road conditions on all of the highways within California. The data is written to MySQL. **Your version of the data is hourly, and only for 2017.**

A Caltrans highway conditions report looks like the following and contains conditions for individual stretches of highway ("area") typically representing a coarse area of the state: Northern, Southern, Central, Sierra Nevada etc.

```
SR 120
[IN THE CENTRAL CALIFORNIA AREA & SIERRA NEVADA]
IS CLOSED FROM CRANE FLAT TO 5 MI WEST OF THE JCT OF US 395 /TIOGA PASS/
(TUOLUMNE, MONO CO) - FOR THE WINTER - MOTORISTS ARE ADVISED TO USE AN
ALTERNATE ROUTE

[YOSEMITE NAT'L PARK]
FOR YOSEMITE NAT'L PARK ROAD INFORMATION CALL 209-372-0200
```

The schema for the `caltrans` table looks like the following:

```
CREATE TABLE caltrans (
    reported      timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
    highway       varchar(6) NOT NULL,
    area          varchar(255),
    condition     text NOT NULL,
    hash          varchar(32) NOT NULL
);
```

`reported` is the time the data was scraped, `highway` is the highway the status pertains to prefixed by its type (i.e. US101, SR1, I405), area refers to a particular part of the state or highway, and text is the update itself. Since we cannot use `text` as a primary key, a `hash` column was added.

US Highway (US)

California State Route (SR)

Interstate (I)

Exercises

(a) Write a query that returns a list of all the highway stretches in 2017 that were closed due to snow at any point of the year, or were closed for the winter. Order them by `highway` and `area` and give us the top 20 results, both columns in descending order.
Hint 1: You don't need to do anything with dates to answer this question.
Hint 2: Before writing a query, look at the data.

First, look at the data. Run some exploratory queries before you write your final query. We want (stretches of) highway that were closed (first condition), due to snow or were closed entirely for the winter. So we should look for stretches of highway that have "closed" in the text and also contain either "snow" or "for the winter" (this is where data intuition and exploration is important). If you look at the data, you will notice that Caltrans road conditions follow a very standardized format (it is not obvious at first though).

One preferred query:

```
SELECT
    DISTINCT
    highway,
    area AS stretch
FROM hw2.caltrans
WHERE condition like '%CLOSED%' AND (
    condition LIKE '%FOR THE WINTER%' OR condition LIKE '%DUE TO SNOW%'
) ORDER BY highway DESC, area DESC
LIMIT 20;
```

This exact query yields the following exact results, but variations may give different results. It is critical that your pattern has high *coverage*, which you can only get if you look at the data. As someone who has spent considerable time living in the Eastern Sierra and also in Southern California, any different answers should contain the following important highways:

(a) Eastern Sierra: US 395, SR 89 (Monitor Pass), SR 88 (Carson Pass), SR 4 (Ebbetts Pass), SR 270 (Bodie), SR 267 (Lake Tahoe), SR 203 (Mammoth Mountain), SR 168 (Aspendell), SR 158 (June Lake Loop). SR 108 (Sonora Pass) may be further down your list.

(b) Southern California, Angeles Crest National Forest, Big Bear: SR 330, SR 38, SR 18

(c) Yosemite: SR 120 (Tioga Pass, East Yosemite) may be further down your list.

```
+---------+-------------------------------------------------+
| highway | stretch                                         |
+---------+-------------------------------------------------+
| US395   | IN THE CENTRAL CALIFORNIA AREA & SIERRA NEVADA  |
| SR89    | IN THE NORTHERN CALIFORNIA AREA & SIERRA NEVADA |
| SR89    | IN THE CENTRAL CALIFORNIA AREA & SIERRA NEVADA  |
| SR88    | IN THE CENTRAL CALIFORNIA & SIERRA NEVADA       |
| SR4     | IN THE CENTRAL CALIFORNIA AREA                  |
| SR38    | IN THE SOUTHERN CALIFORNIA AREA                 |
| SR330   | IN THE SOUTHERN CALIFORNIA AREA                 |
| SR33    | IN THE SOUTHERN CALIFORNIA AREA                 |
| SR3     | IN THE NORTHERN CALIFORNIA AREA                 |
| SR270   | IN THE CENTRAL CALIFORNIA AREA & SIERRA NEVADA  |
| SR267   | IN THE NORTHERN CALIFORNIA AREA                 |
| SR203   | IN THE CENTRAL CALIFORNIA AREA & SIERRA NEVADA  |
| SR20    | IN THE NORTHERN CALIFORNIA AREA                 |
| SR2     | IN THE SOUTHERN CALIFORNIA AREA                 |
| SR18    | IN THE SOUTHERN CALIFORNIA AREA                 |
| SR172   | IN THE NORTHERN CALIFORNIA AREA                 |
| SR168   | IN THE CENTRAL CALIFORNIA AREA & SIERRA NEVADA  |
| SR158   | IN THE CENTRAL CALIFORNIA AREA & SIERRA NEVADA  |
| SR138   | IN THE SOUTHERN CALIFORNIA AREA                 |
| SR130   | IN THE CENTRAL CALIFORNIA AREA                  |
| SR120   | IN THE CENTRAL CALIFORNIA AREA & SIERRA NEVADA  |
| SR108   | IN THE CENTRAL CALIFORNIA AREA & SIERRA NEVADA  |
| I5      | IN THE NORTHERN CALIFORNIA AREA                 |
+---------+-------------------------------------------------+
```

(b) For each highway stretch found in part (a), compute the percentage of days out of the year that it was closed. If a highway stretch was closed for only a partial day, it counts as a full day. There are at least three ways to solve this problem. Try to use a method that involves a join, and a method that does not. Report the highway, area/stretch, and the percentage of days it was closed in descending order by percentage, and only gives us the 5 highest percentages and the highways and areas they belong to. You may hardcode the number of days in the year (see the note below).

Method 0: No Subquery with `DISTINCT`

I typically use `GROUP BY/COUNT` instead of `COUNT DISTINCT` because it is more explicit to me; however, it is perfectly valid to use `COUNT DISTINCT` for this problem and as a bonus, you do not need a subquery!

```
SELECT
    highway,
    area,
    COUNT(DISTINCT EXTRACT(DOY FROM reported)) * 100 / 365 AS percentage_of_days_closed_365,
    COUNT(DISTINCT EXTRACT(DOY FROM reported)) * 100 / 353 AS percentage_of_days_closed_353
FROM hw2.caltrans
WHERE condition LIKE '%CLOSED%DUE TO SNOW%' OR condition LIKE '%CLOSED%FOR THE WINTER%'
GROUP BY highway, area
ORDER BY percentage_of_days_closed_365 DESC;
```

This method is 78.7ms, which is actually slower than Method 1, but it works fine for our purposes.

Method 1: Select within a Select

```
SELECT
    highway,
    stretch,
    COUNT(1) AS days_closed,
    100 * COUNT(1) / 365 AS pct_closed_365,
    100 * COUNT(1) / 353 AS pct_closed_353
FROM (
    SELECT
        highway AS highway,
        area AS stretch,
        DATE(reported) AS closure
    FROM hw2.caltrans
    WHERE condition LIKE '%CLOSED%' AND (
        condition LIKE '%FOR THE WINTER%' OR condition LIKE '%DUE TO SNOW%')
    GROUP BY highway, stretch, closure
) result GROUP BY highway, stretch ORDER BY pct_closed_365 DESC;
```

We get the following results. The percentage depends on if we used 365 or 353. This method is comparable at 66.3ms.

```
+---------+-----------------------------------------------+-------------+----------------+----------------+
| highway | stretch                                       | days_closed | pct_closed_365 | pct_closed_353 |
+---------+-----------------------------------------------+-------------+----------------+----------------+
| SR89    | IN THE NORTHERN CALIFORNIA AREA & SIERRA NEVADA |         242 |        66.3014 |        68.5552 |  -- Lassen National Park (Tehama/Shasta Co) & Lake Tahoe
| SR120   | IN THE CENTRAL CALIFORNIA AREA & SIERRA NEVADA  |         225 |        61.6438 |        63.7394 |  -- Tioga Pass (Yosemite East Entrance, Mono County)
| SR203   | IN THE CENTRAL CALIFORNIA AREA & SIERRA NEVADA  |         224 |        61.3699 |        63.4561 |  -- Mammoth Mountain (Mono County)
| SR108   | IN THE CENTRAL CALIFORNIA AREA & SIERRA NEVADA  |         203 |        55.6164 |        57.5071 |  -- Sonora Pass (Mono/Tuolumne/Alpine County Line)
| SR4     | IN THE CENTRAL CALIFORNIA AREA                  |         200 |        54.7945 |        56.6572 |  -- Ebbetts Pass (Alpine County)
| SR168   | IN THE CENTRAL CALIFORNIA AREA & SIERRA NEVADA  |         149 |        40.8219 |        42.2096 |  -- Aspendell/Bishop Creek (Inyo County)
| SR270   | IN THE CENTRAL CALIFORNIA AREA & SIERRA NEVADA  |         145 |        39.7260 |        41.0765 |  -- Bodie Ghosttown (Mono County)
| SR89    | IN THE CENTRAL CALIFORNIA AREA & SIERRA NEVADA  |         127 |        34.7945 |        35.9773 |  -- Monitor Pass (Alpine County)
| SR2     | IN THE SOUTHERN CALIFORNIA AREA                 |         117 |        32.0548 |        33.1445 |  -- Angeles Crest Highway (LA/San Bernardino Co)
| SR158   | IN THE CENTRAL CALIFORNIA AREA & SIERRA NEVADA  |          91 |        24.9315 |        25.7790 |  -- June Lake Loop (Mono County)
| SR172   | IN THE NORTHERN CALIFORNIA AREA                 |          66 |        18.0822 |        18.6969 |  -- Lassen National Forest (Tehama County)
| SR88    | IN THE CENTRAL CALIFORNIA & SIERRA NEVADA       |          15 |         4.1096 |         4.2493 |  -- Carson Pass (Alpine County)
| SR3     | IN THE NORTHERN CALIFORNIA AREA                 |          13 |         3.5616 |         3.6827 |  -- Trinity Alps Wilderness (Trinity County)
| SR130   | IN THE CENTRAL CALIFORNIA AREA                  |           7 |         1.9178 |         1.9830 |  -- Diablo Range/Mt. Hamilton (Santa Clara County)
| SR33    | IN THE SOUTHERN CALIFORNIA AREA                 |           4 |         1.0959 |         1.1331 |  -- Ojai (Ventura/Santa Barbara Co)
| I5      | IN THE NORTHERN CALIFORNIA AREA                 |           3 |         0.8219 |         0.8499 |  -- Grapevine (Los Angeles/Kern County Line)
| US395   | IN THE CENTRAL CALIFORNIA AREA & SIERRA NEVADA  |           3 |         0.8219 |         0.8499 |  -- Eastern Sierra (Lee Vining to Bridgeport, Mono County)
| SR18    | IN THE SOUTHERN CALIFORNIA AREA                 |           2 |         0.5479 |         0.5666 |  -- Big Bear (San Bernardino County)
| SR267   | IN THE NORTHERN CALIFORNIA AREA                 |           1 |         0.2740 |         0.2833 |  -- Truckee/Lake Tahoe (Placer County)
| SR330   | IN THE SOUTHERN CALIFORNIA AREA                 |           1 |         0.2740 |         0.2833 |  -- Big Bear/Running Springs (San Bernardino County)
| SR20    | IN THE NORTHERN CALIFORNIA AREA                 |           1 |         0.2740 |         0.2833 |  -- Yuba Pass/Emigrant Gap (Placer County)
| SR138   | IN THE SOUTHERN CALIFORNIA AREA                 |           1 |         0.2740 |         0.2833 |  -- Crestline (San Bernardino County)
| SR38    | IN THE SOUTHERN CALIFORNIA AREA                 |           1 |         0.2740 |         0.2833 |  -- Big Bear (San Bernardino County)
+---------+-----------------------------------------------+-------------+----------------+----------------+
```

We could add another outermost query to reduce some redundancy, but it is unnecessary. If we do not want to hardcode the number of days, we can use a correlated subquery by replacing 365 or 353 with the scalar subquery, but there are a host of other issues we need to deal with so we use our best judgment and hardcode using 365, or the number of days represented in the data (353). If each highway was recorded on a different number of days, we would even need to use a join. Let's not go there.

Method 2: Join as a Filter

We can use a join. We join the `caltrans` table with a subset of itself to filter out highway stretches that were never closed for snow or for the winter.

```
SELECT
    closures.highway,
    stretch,
    COUNT(1) AS days_closed,
    100 * COUNT(1) / 365 AS pct_closed_365,
    100 * COUNT(1) / 353 AS pct_closed_353
FROM (
    SELECT
        c.highway AS highway,
        c.area AS stretch,
        DATE(c.reported) AS closure
    FROM hw2.caltrans c
    JOIN (
        SELECT
            DISTINCT
            highway,
            area
        FROM hw2.caltrans
        WHERE condition like '%CLOSED%' AND (
            condition LIKE '%FOR THE WINTER%' OR condition LIKE '%DUE TO NOW%')
    ) snow_highways ON c.highway = snow_highways.highway
    WHERE condition like '%CLOSED%' AND (
        condition LIKE '%FOR THE WINTER%' OR condition LIKE '%DUE TO SNOW%')
    GROUP BY c.highway, stretch, closure
) closures
GROUP BY closures.highway, closures.stretch
ORDER BY pct_closed_365 DESC;
```

This query is slower (128ms) than method 1 because of using both a join and a subquery. This mimics a *left semijoin* except that a semijoin technically uses a `WHERE EXISTS some_subquery` instead of the `JOIN` syntax. We did not even need to know it was called a semijoin! A left semijoin is actually an inner join where matches are dictated by the table on the lefthand side of the join. Usually `LEFT` implies `OUTER` but this is not the case with the semijoin. One reason we do not cover it: it's too confusing.

Method 3: Using an `IN` Subquery as a Filter

We replace the join with a `WHERE` clause that tests whether or not the stretch of highway is in the set of highways with snow or winter closures. Again this simulates left semijoin.

```
SELECT
    highway,
    stretch,
    COUNT(1) AS days_closed,
    100 * COUNT(1) / 365 AS pct_closed_365,
    100 * COUNT(1) / 353 AS pct_closed_353
FROM (
    SELECT
        c.highway AS highway,
        c.area AS stretch,
                DATE(c.reported) AS closure
    FROM hw2.caltrans c
    WHERE (highway, area) IN (
        SELECT
        DISTINCT
            highway,
            area
        FROM hw2.caltrans
        WHERE condition like '%CLOSED%' AND (
            condition LIKE '%FOR THE WINTER%' OR condition LIKE '%DUE TO SNOW%')
    ) AND condition LIKE '%CLOSED%' AND (condition LIKE '%FOR THE WINTER%' OR condition LIKE '%DUE TO SNOW%')
    GROUP BY highway, stretch, closure
) closures
GROUP BY highway, stretch
ORDER BY pct_closed_365 DESC;
```

This is slightly the slowest (128s) in MySQL because a bunch of irrelevant rows are produced, which are then filtered out using the `IN`. The set containment also may require more time to process. In PostgreSQL, run time is comparable to Method 2. This is likely due to a difference in indexing or the optimizer.

Important Note: The results should match exactly, but when there are ties, the query processor seems to slightly alter the ordering. This only affects the last few records, so we will ignore these when grading.

Method 4: Formal Left Semijoin

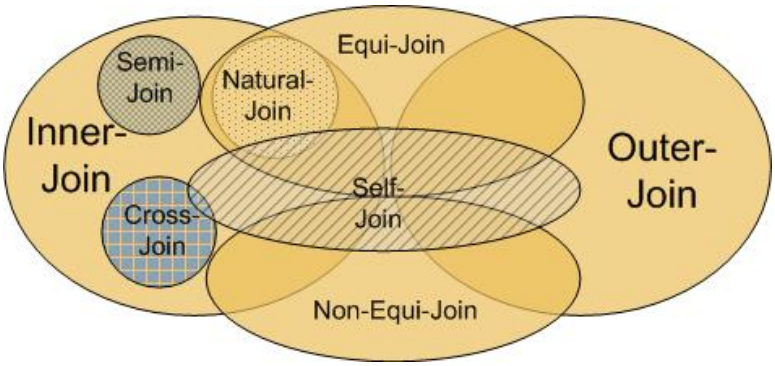
We can replace methods 2 and 3 with the formal left semijoin syntax. A left semijoin is actually an inner join where matches are dictated by the table on the lefthand side of the join. Usually `LEFT` implies `LEFT OUTER` but this is not the case with the semijoin. This is one reason we do not cover it – it is too confusing.

```
SELECT
    highway,
    stretch,
    COUNT(1) AS days_closed,
    100 * COUNT(1) / 365 AS pct_closed_365,
    100 * COUNT(1) / 353 AS pct_closed_353
FROM (
    SELECT
        c.highway AS highway,
        c.area AS stretch,
        DATE(c.reported) AS closure
    FROM hw2.caltrans c
    WHERE EXISTS (
        SELECT
            1
        FROM hw2.caltrans
        WHERE condition like '%CLOSED%' AND (
            condition LIKE '%FOR THE WINTER%' OR condition LIKE '%DUE TO SNOW%')
    ) AND condition LIKE '%CLOSED%' AND (condition LIKE '%FOR THE WINTER%' OR condition LIKE '%DUE TO SNOW%')
    GROUP BY highway, stretch, closure
) closures
GROUP BY highway, stretch
ORDER BY pct_closed_365 DESC;
66ms
```

If we replace `EXISTS` with `NOT EXISTS` (or `NOT IN` in method 3) we get an *antijoin*.

This is the **fastest** method along with Method 1. **Thus, for filtering records, we can use a subquery, or a semijoin. While a normal join and set containment works, they are slower.** The `COUNT DISTINCT` method is also not the fastest, but it is darn simple to write, so there are tradeoffs to consider.

Part 2: Join Definitions



Exercises

(a) Your instructor almost included the above Venn Diagram in his lecture slides to show how different types of joins are related, but he noticed that it was wrong in at least one way. Explain at least one thing that is wrong about the diagram.

A natural join is not an inner join!!! It is not a subset of inner joins either! Yes, in PostgreSQL the syntax `NATURAL JOIN` yields an `INNER JOIN` if there are columns in common (and a `CROSS JOIN` otherwise), but the concepts are not the same!

It is also unclear why the author classified a `CROSS JOIN` as an `INNER JOIN`.

Finally, outer join and inner join should overlap because an `OUTER` join keeps the same matches that an `INNER` join does, except it handles *mismatches* differently.

Overall, this Venn diagram is a mess.

Part 3: More Joins and Subqueries

In Homework 1, we did several things with the Bird Scooter use case, but we did not have any data to practice writing queries with. Suppose we now have trip data in the following two tables:

1. `trip_starts`;
2. `trip_ends`;

Exercises

(a) Write a query that computes the elapsed time of each trip. If something happened and a trip end was not recorded, the elapsed time shall be 24 hours, per Bird's policy. Print your results as `trip_id`, `user_id`, and `trip_length`. Only show the first 5, without any special ordering.

```
SELECT
  l.trip_id,
  l.user_id,
  COALESCE(CEILING(((EXTRACT(EPOCH FROM r.time) -
      EXTRACT(EPOCH FROM l.time)) / 60)::numeric), 1440) AS trip_length
FROM hw2.trip_start l
LEFT JOIN hw2.trip_end r
ON l.trip_id = r.trip_id and l.user_id = r.user_id
LIMIT 5;
```

```
+---------+---------+-------------+
| trip_id | user_id | trip_length |
+---------+---------+-------------+
|       0 |   20685 |           2 |
|       2 |   34808 |           3 |
|       3 |   25463 |        1440 |
|       4 |   26965 |           2 |
|       5 |     836 |           1 |
+---------+---------+-------------+
```

(b) Write a query that computes the charge to the user for each trip. The charge is calculated as follows: $1 flat rate per trip plus 15 cents per minute. All fractional minutes are rounded up to the next minute. Assume we did not store the results of the query from part (a). Print the first 5 results (no ordering) as `trip_id`, `user_id` and `trip_charge`.

```
SELECT
  trip_id,
  user_id,
  trip_length * 0.15 + 1.00 AS trip_charge
FROM (
  SELECT
      l.trip_id AS trip_id,
      l.user_id AS user_id,
      COALESCE(CEILING(((EXTRACT(EPOCH FROM r.time) -
          EXTRACT(EPOCH FROM l.time)) / 60)::numeric), 1440) AS trip_length
  FROM hw2.trip_start l
  LEFT JOIN hw2.trip_end r
  ON l.trip_id = r.trip_id and l.user_id = r.user_id
) durations
LIMIT 5;
```

This could also be written without the subquery, but the query becomes unreadable.

In the next problem, we cap the maximum daily charge is $100. We also make the assumption that the user is billed monthly, not by ride. Unfortunately, this is not clear until reading the next part. So, for this part, it is fine to have charges above $100. We get the following results:

```
+---------+---------+-------------+
| trip_id | user_id | trip_charge |
+---------+---------+-------------+
|       0 |   20685 |        1.30 |
|       2 |   34808 |        1.45 |
|       3 |   25463 |      217.00 |
|       4 |   26965 |        1.30 |
|       5 |     836 |        1.15 |
+---------+---------+-------------+
```

(c) Putting it all together: Suppose we bill the user at the end of the month rather than at the end of each trip. Write a query that computes the monthly charge for trips in March 2018 for each user assuming we did not store the results from parts (a) or (b). Print your results: `user_id` and `monthly_total` for the first five users (no ordering). In particular, how much does `user_id = 2` owe?

Answers will vary depending on where you chose to put the, well, `WHERE` clause. Typically, we want to filter filter filter as much as possible before doing an inner join, but we have to *very* careful when using outer joins, and it's best do filtering after the join. **Think about why.** MySQL's query processor creates a fairly efficient query plan regardless of where we put the `WHERE` in this case (as long as it is outside of the join), but the difference in execution time can be dramatic in systems like Hive. There are at least three different places we can put the `WHERE` clause.

Also, students were to cap the daily charge at $100, but this seems to have been inadvertently deleted somewhere in my edits, so it will not be graded.

Regardless of method, we get the following results:

```
+---------+---------------+
| user_id | monthly_total |
+---------+---------------+
|       0 |        105.50 |
|       1 |          4.05 |
|       2 |        314.05 |
|       3 |         11.90 |
|       4 |        210.55 |
+---------+---------------+
```

If we do not cap at $100, we may get the following which is also accepted. We can see the difference between the total and the capped total.

```
+---------+-------+---------------+--------------------+
| user_id | trips | monthly_total | month_total_capped |
+---------+-------+---------------+--------------------+
|       0 |     5 |        222.50 |             105.50 |
|       1 |     3 |          4.05 |               4.05 |
|       2 |    13 |        665.05 |             314.05 |
|       3 |     8 |         11.90 |              11.90 |
|       4 |    10 |        444.55 |             210.55 |
+---------+-------+---------------+--------------------+
```

Method 1: WHERE in Same Subquery as LEFT JOIN

```
SELECT
    user_id,
    SUM(daily_total) AS monthly_total
FROM (
    SELECT
        user_id,
        LEAST(100, SUM(trip_charge)) AS daily_total
    FROM (
        SELECT
            trip_id,
            user_id,
            time,
            trip_length * 0.15 + 1.00 AS trip_charge
        FROM (
            SELECT
                l.trip_id AS trip_id,
                l.user_id AS user_id,
                DATE(l.time) AS time,
                COALESCE(CEILING(((EXTRACT(EPOCH FROM r.time) -
                    EXTRACT(EPOCH FROM l.time)) / 60)::numeric), 1440) AS trip_length
            FROM hw2.trip_start l
            LEFT JOIN hw2.trip_end r
            ON l.trip_id = r.trip_id and l.user_id = r.user_id
            WHERE EXTRACT(MONTH FROM l.time) = 3 AND EXTRACT(YEAR FROM l.time) = 2018
            -- right side time is NULL, so don't use it to filter!
        ) durations
    ) charges
    GROUP BY user_id, DATE(time)
) daily
GROUP BY user_id
LIMIT 5;
3273ms
```

Method 2: WHERE in Subquery Outside LEFT JOIN

```
SELECT
    user_id,
    SUM(daily_total) AS monthly_total
FROM (
    SELECT
        user_id,
        LEAST(100, SUM(trip_charge)) AS daily_total
    FROM (
        SELECT
            trip_id,
            user_id,
            trip_length * 0.15 + 1.00 AS trip_charge,
            trip_date
        FROM (
            SELECT
                l.trip_id AS trip_id,
                l.user_id AS user_id,
                DATE(l.time) as trip_date,
                COALESCE(CEILING(((EXTRACT(EPOCH FROM r.time) -
                    EXTRACT(EPOCH FROM l.time)) / 60)::numeric), 1440) AS trip_length
            FROM hw2.trip_start l
            LEFT JOIN hw2.trip_end r
            ON l.trip_id = r.trip_id and l.user_id = r.user_id
        ) durations
        WHERE EXTRACT(MONTH FROM trip_date) = 3 AND EXTRACT(YEAR FROM trip_date) = 2018
    ) charges
    GROUP BY user_id, DATE(trip_date)
) daily
GROUP BY user_id
LIMIT 5;
3255ms
```

Method 3: WHERE in Outermost Query

```
SELECT
    user_id,
    COUNT(trip_id) AS trips,
    SUM(trip_charge) AS monthly_total
FROM (
    SELECT
        trip_id,
        user_id,
        start_time,
        LEAST(100.00, trip_length * 0.15 + 1.00) AS trip_charge
    FROM (
        SELECT
            l.trip_id AS trip_id,
            l.user_id AS user_id,
            DATE(l.time) AS start_time,
            COALESCE(CEILING(((EXTRACT(EPOCH FROM r.time) -
                EXTRACT(EPOCH FROM l.time)) / 60)::numeric), 1440) AS trip_length
        FROM hw2.trip_start l
        LEFT JOIN hw2.trip_end r
        ON l.trip_id = r.trip_id and l.user_id = r.user_id
    ) durations
) charges
WHERE EXTRACT(MONTH FROM start_time) = 3 AND EXTRACT(YEAR FROM start_time) = 2018
GROUP BY user_id
LIMIT 5;
2980ms
```

Wrong Method 1: Including the End Date in a Filter

If you got a charge of \$14.05 for `user_id = 2`, you likely did this, or used the wrong join type. Remember that when there is no end time, the attributes in the result from the left join related to the right hand table will be `NULL`. So, by applying a filter with month and year on `end_time` we remove all of the mismatches because `MONTH(NULL)` $\neq 3$! So, the user gets charged for the completed trips, but never gets charged for the trips that did not have an end time.

```
SELECT
    user_id,
    SUM(daily_total) AS monthly_total
FROM (
    SELECT
        user_id,
        LEAST(100, SUM(trip_charge)) AS daily_total
    FROM (
        SELECT
            trip_id,
            user_id,
            time,
            trip_length * 0.15 + 1.00 AS trip_charge
        FROM (
            SELECT
                l.trip_id AS trip_id,
                l.user_id AS user_id,
                DATE(l.time) AS time,
                COALESCE(CEILING(((EXTRACT(EPOCH FROM r.time) -
                    EXTRACT(EPOCH FROM l.time)) / 60)::numeric), 1440) AS trip_length
            FROM hw2.trip_start l
            LEFT JOIN hw2.trip_end r
            ON l.trip_id = r.trip_id and l.user_id = r.user_id
            WHERE EXTRACT(MONTH FROM l.time) = 3 AND EXTRACT(MONTH FROM r.time) = 3
                AND EXTRACT(YEAR FROM l.time) = 2018 AND EXTRACT(YEAR FROM r.time) = 2018
            -- right side time is NULL, so don't use it to filter!
        ) durations
    ) charges
    GROUP BY user_id, DATE(time)
) daily
GROUP BY user_id
LIMIT 5;
```

Wrong Method 2: Using `ON` Instead of `WHERE` in `OUTER` Join

We get the wrong answer if we do this. Why? Think about how the join will differ when the filter is carried out *before* the outer join. This should work fine for inner join though.

```
ON l.trip_id = r.trip_id AND l.user_id = r.user_id AND
    EXTRACT(MONTH FROM l.time) = 3 AND EXTRACT(YEAR FROM l.time) = 2018
```

Note that we can place some `WHERE` restrictions in the `ON` and in some systems this makes a difference for efficiency; however, it does not work well for an `OUTER` join.

(d) In the solution set for Homework 1, it was mentioned that another way we can record starts and ends of trips was to use one table, 2 rows per trip: one row representing the start and a second row representing the end of the trip. We would then have an `enum` or `bit(1)` that specifies whether the row refers to a start or an end. If we wanted to use this one single table as the basis to charge users, what type of join would we need to compute?

Self left [outer] equijoin.