CSCI 5800 – Big Data Systems

Time Series Analysis

Table of Contents

This assignment covers the topic of time series analysis using TimescaleDB. Make sure you have your system fully configured before starting this assignment. Refer to the Assignment Prelab document to do so.

TO-DO: *Run the example queries, understand what each example is doing before writing the assignment part (writing queries)*

Connect to PostgreSQL

This section includes a reminder of how to connect to PostgreSQL.

Open PostgreSQL and open the nyc_data database:

psql -U postgres -h localhost -d nyc_data

Then use the following command to see the list of tables:

\dt

```
shahab@shahab-VirtualBox:~$ psql -U postgres -h localhost -d nyc_data
Password for user postgres:
psql (9.6.3)
SSL connection (protocol: TLSv1.2, cipher: ECDHE-RSA-AES256-GCM-SHA384, bits: 25
6, compression: off)
Type "help" for help.

nyc_data=# \dt
             List of relations
 Schema |     Name      | Type  |  Owner
--------+---------------+-------+----------
 public | payment_types | table | postgres
 public | rates         | table | postgres
 public | rides         | table | postgres
(3 rows)
```

Now use the following command to see the columns of *rides* table (you can press *q* for going back to the terminal):

\d rides

```
                    Table "public.rides"
       Column         |            Type             | Modifiers
----------------------+-----------------------------+-----------
 vendor_id            | text                        |
 pickup_datetime      | timestamp without time zone | not null
 dropoff_datetime     | timestamp without time zone | not null
 passenger_count      | numeric                     |
 trip_distance        | numeric                     |
 pickup_longitude     | numeric                     |
 pickup_latitude      | numeric                     |
 rate_code            | integer                     |
 dropoff_longitude    | numeric                     |
 dropoff_latitude     | numeric                     |
 payment_type         | integer                     |
 fare_amount          | numeric                     |
 extra                | numeric                     |
 mta_tax              | numeric                     |
 tip_amount           | numeric                     |
 tolls_amount         | numeric                     |
 improvement_surcharge | numeric                    |
 total_amount         | numeric                     |
Indexes:
    "rides_passenger_count_pickup_datetime_idx" btree (passenger_count, pickup_d
atetime DESC)
    "rides_pickup_datetime_vendor_id_idx" btree (pickup_datetime DESC, vendor_id
)
    "rides_rate_code_pickup_datetime_idx" btree (rate_code, pickup_datetime DESC
)
    "rides_vendor_id_pickup_datetime_idx" btree (vendor_id, pickup_datetime DESC
)
Number of child tables: 4 (Use \d+ to list them.)
```

Run and Study Query Examples

Example 1

Find the average fare of rides with 2+ passengers per day until *'2016-01-08'*:

```
SELECT date_trunc('day', pickup_datetime) as day, avg(fare_amount)
FROM rides
WHERE passenger_count > 1 AND pickup_datetime < '2016-01-08'
GROUP BY day
ORDER BY day;
```

```
nyc_data=# SELECT date_trunc('day', pickup_datetime) as day, avg(fare_amount)
nyc_data-#   FROM rides
nyc_data-#   WHERE passenger_count > 1 AND pickup_datetime < '2016-01-08'
nyc_data-#   GROUP BY day ORDER BY day;
         day         |         avg
---------------------+--------------------
 2016-01-01 00:00:00 | 13.3990821679715529
 2016-01-02 00:00:00 | 13.0224687415181399
 2016-01-03 00:00:00 | 13.5382068607068607
 2016-01-04 00:00:00 | 12.9618895561740149
 2016-01-05 00:00:00 | 12.6614611935518309
 2016-01-06 00:00:00 | 12.5775245695086098
 2016-01-07 00:00:00 | 12.5868802584437019
(7 rows)
```

Example 2

Total number of rides by day for first 5 days

```
SELECT date_trunc('day', pickup_datetime) as day, COUNT(*) FROM rides
GROUP BY day
ORDER BY day
LIMIT 5;
```

```
nyc_data=# SELECT date_trunc('day', pickup_datetime) as day, COUNT(*) FROM rides
nyc_data-# GROUP BY day
nyc_data-# ORDER BY day
nyc_data-# LIMIT 5;
         day         | count
---------------------+--------
 2016-01-01 00:00:00 | 345037
 2016-01-02 00:00:00 | 312831
 2016-01-03 00:00:00 | 302878
 2016-01-04 00:00:00 | 316171
 2016-01-05 00:00:00 | 343251
(5 rows)

nyc_data=#
```

Note: This query is not supported by PostgreSQL itself and *time_bucket* is provided by timescaledb.

Example 3

Find the number of rides by 5-minute intervals on 2016-01-01 using the TimescaleDB "*time_bucket*" function

```
SELECT time_bucket('5 minute', pickup_datetime) as five_min, count(*)
FROM rides
WHERE pickup_datetime < '2016-01-01 02:00'
GROUP BY five_min
ORDER BY five_min;
```

```
nyc_data=# SELECT time_bucket('5 minute', pickup_datetime) as five_min, count(*)
nyc_data-# FROM rides
nyc_data-# WHERE pickup_datetime < '2016-01-01 02:00'
nyc_data-# GROUP BY five_min
nyc_data-# ORDER BY five_min;
      five_min       | count
---------------------+-------
 2016-01-01 00:00:00 |   703
 2016-01-01 00:05:00 |  1482
 2016-01-01 00:10:00 |  1959
 2016-01-01 00:15:00 |  2200
 2016-01-01 00:20:00 |  2285
 2016-01-01 00:25:00 |  2291
 2016-01-01 00:30:00 |  2349
 2016-01-01 00:35:00 |  2328
 2016-01-01 00:40:00 |  2440
 2016-01-01 00:45:00 |  2372
 2016-01-01 00:50:00 |  2388
 2016-01-01 00:55:00 |  2473
 2016-01-01 01:00:00 |  2395
 2016-01-01 01:05:00 |  2510
 2016-01-01 01:10:00 |  2412
 2016-01-01 01:15:00 |  2482
 2016-01-01 01:20:00 |  2428
 2016-01-01 01:25:00 |  2433
 2016-01-01 01:30:00 |  2337
 2016-01-01 01:35:00 |  2366
 2016-01-01 01:40:00 |  2325
 2016-01-01 01:45:00 |  2257
 2016-01-01 01:50:00 |  2316
 2016-01-01 01:55:00 |  2250
(24 rows)
```

Example 4

Analyze rides by rate type:

```
SELECT rate_code, COUNT(vendor_id) as num_trips FROM rides
WHERE pickup_datetime < '2016-01-08'
GROUP BY rate_code
ORDER BY rate_code;
```

```
nyc_data=#
nyc_data=# SELECT rate_code, COUNT(vendor_id) as num_trips FROM rides
nyc_data-# WHERE pickup_datetime < '2016-01-08'
nyc_data-# GROUP BY rate_code
nyc_data-# ORDER BY rate_code;

 rate_code | num_trips
-----------+-----------
         1 |   2266401
         2 |     54832
         3 |      4126
         4 |       967
         5 |      7193
         6 |        17
        99 |        42
(7 rows)
```

rate_code doesn't really tell us what these groups represent, and it doesn't look like there is any other info on rates in the rides table. However, there is a separate rates table and TimescaleDB supports JOINs between tables:

```
SELECT rates.description, COUNT(vendor_id) as num_trips
FROM rides JOIN rates on rides.rate_code = rates.rate_code
WHERE pickup_datetime < '2016-01-08'
GROUP BY rates.description ORDER BY rates.description;
```

```
nyc_data=# SELECT rates.description, COUNT(vendor_id) as num_trips FROM rides
nyc_data-# JOIN rates on rides.rate_code = rates.rate_code
nyc_data-# WHERE pickup_datetime < '2016-01-08'
nyc_data-# GROUP BY rates.description ORDER BY rates.description;

     description      | num_trips
----------------------+-----------
 group ride           |        17
 JFK                  |     54832
 Nassau or Westchester |       967
 negotiated fare      |      7193
 Newark               |      4126
 standard rate        |   2266401
(6 rows)
```

Now we have something that is human readable. In particular, two of these rate types correspond to local airports (JFK, Newark). Let's take a closer look at those two:

```
SELECT rates.description, COUNT(vendor_id) as num_trips,
       AVG(dropoff_datetime - pickup_datetime) as avg_trip_duration,
       AVG(total_amount) as avg_total,
       AVG(tip_amount) as avg_tip, MIN(trip_distance) as min_distance,
       AVG(trip_distance) as avg_distance, MAX(trip_distance) as
       max_distance,
       AVG(passenger_count) as avg_passengers
FROM rides JOIN rates on rides.rate_code = rates.rate_code
WHERE rides.rate_code in (2,3) AND pickup_datetime < '2016-02-01'
GROUP BY rates.description
ORDER BY rates.description;
```

Write Queries

Query 1

Write a query to find the total number of rides, total distance, and total fare for each day and *rate_code*. Order the result based on day and then *rate_code.* Do you see any unexpected number(s) in the result suggesting inaccuracy in the dataset?

Include your query and its result (copy and paste) into the text file. Do this for **Q2** and **Q3** as well.

Output format should be as follows:

day	rate_code	count	total_fare	total_distance
2016-01-01 00:00:00	1	121232	1451436.5	363822.74
2016-01-01 00:00:00	2	3090	160525.85	54291.90
2016-01-01 00:00:00	3	372	24005.5	6332.91
2016-01-01 00:00:00	4	59	3901.5	1058.75

Note that this result is computed over a different dataset. Your result would be different but should follow the same format

Query 2

Find the change in the number of rides for each *rate_code/hour* between 2016-01-01 and 2016-01-02. For example, if there are 10 rides for *rate_code* 99 between 2pm and 3pm on 2016-01-01 and there are 5 rides for the same time and *rate_code* on 2016-01-02, the result must contain 2, 99, -5.

You might find the following functions useful:

- extract(field from timestamp) extracts hour from date
  - extract(hour from timestamp '2001-02-16 20:38:40') returns 20
- date(datetime) extracts date from datetime
  - date('2001-02-16 20:38:40') = '2001-02-16'

Query 3

Write a query to find the most expensive *rate_code (i.e., the highest avg(fare_amount))* for each hour of the day on 2016-01-01. The result should contain the date and hour (e.g., 2016-01-01 01:00:00), rate_code, and the average fare for that rate_code. Please note that **only** the most expensive rate code for each hour must be included in the result. For example, if rate_code 1 is the most expensive between 1pm and 2pm, other rate_codes must not appear in the result for 1-2pm.

Deliverable

- Submit **a single PDF file with the requested three queries**.
- **Include the Query Number** before your query code.
- **Include your Name** in the pdf document.
- **Use Courier 11pt font for your document.**
- Assignment deadline is posted on Canvas.