Week 3 Homework: Data Warehouse & BigQuery

Georgios Grigoriou
georgiosvgrigoriou@gmail.com
Deadline: 13 February (Monday), 22:00 CET

In this Homework, I familiarised myself with Data Warehouse and BigQuery.

Code (sql) can be found here

Instructions

Important Note:

You can load the data however you would like, but keep the files in .GZ Format. If you are using orchestration such as Airflow or Prefect do not load the data into Big Query using the orchestrator.

Stop with loading the files into a bucket.

NOTE: You can use the CSV option for the GZ files when creating an External Table

SETUP:

Create an external table using the fhv 2019 data.

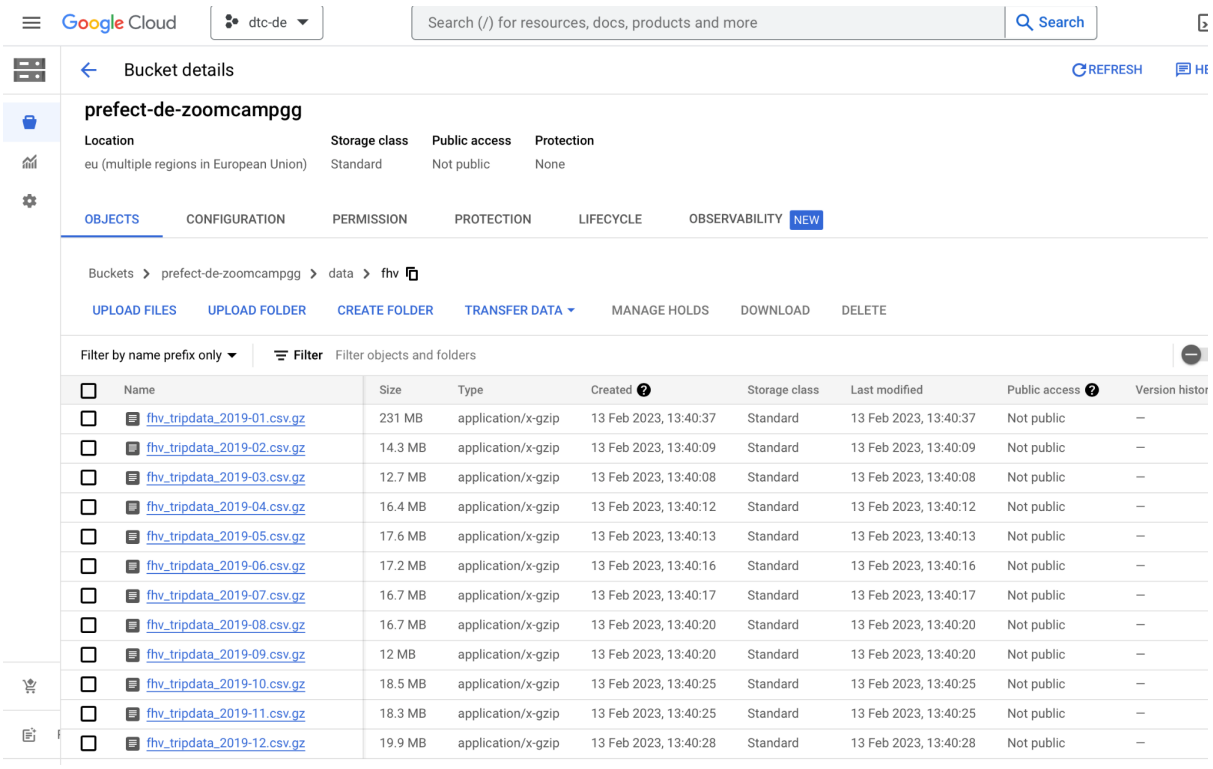
Create a table in BQ using the fhv 2019 data (do not partition or cluster this table).

Data can be found here:
https://github.com/DataTalksClub/nyc-tlc-data/releases/tag/fhv

In this homework, I familiarised myself with BigQuery orchestration tool as part of the Data Engineering Zoomcamp course. The code that was used can be found here

Preparation

Data were loaded in Google Cloud Storage

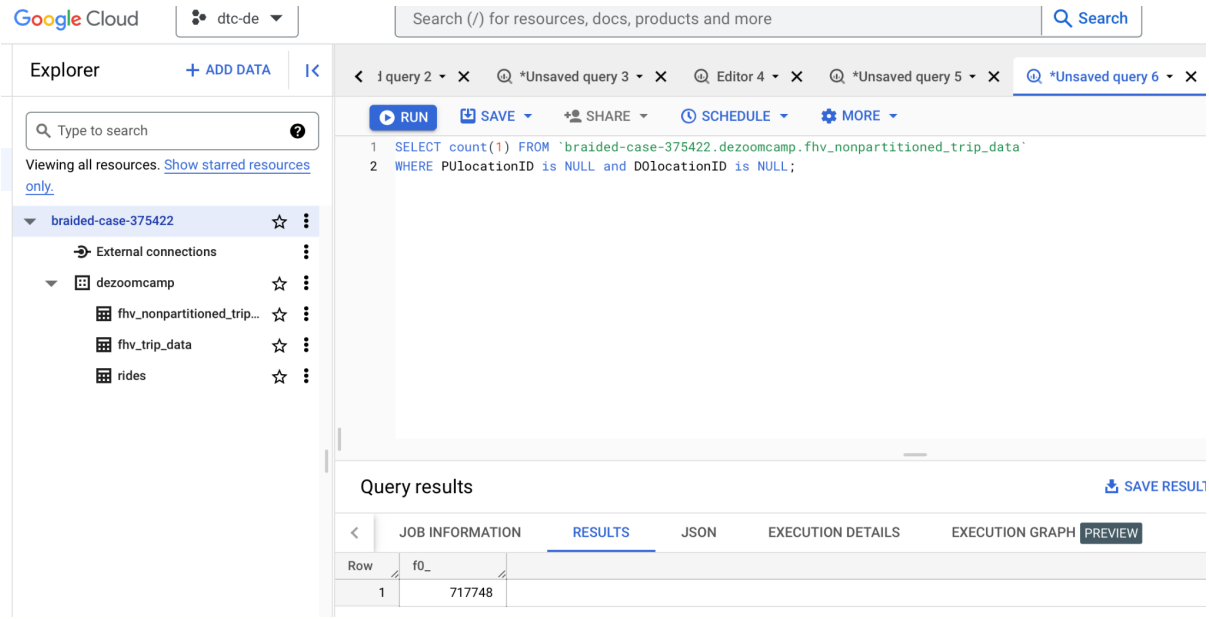


Create an external table using the fhv 2019 data.

```
-- Creating external table referring to gcs path

CREATE OR REPLACE EXTERNAL TABLE `<project id>.dezoomcamp.fhv_trip_data`

OPTIONS (

 format = 'CSV',

 uris = ['gs://prefect-de-zoomcampgg/data/fhv/fhv_tripdata_2019-*.csv.gz']

);
```

Create a table in BQ using the fhv 2019 data (do not partition or cluster this table).

```
CREATE OR REPLACE TABLE `<project id>.dezoomcamp.fhv_nonpartitioned_trip_data`

AS SELECT * FROM `<project id>.dezoomcamp.fhv_trip_data`;
```



Question 1: What is the count for fhv vehicle records for year 2019?

- 65,623,481
- 43,244,696
- 22,978,333
- 13,942,414

Question 1: Solution

The correct answer is **43,244,696.**

By running the following SQL Command in BigQuery

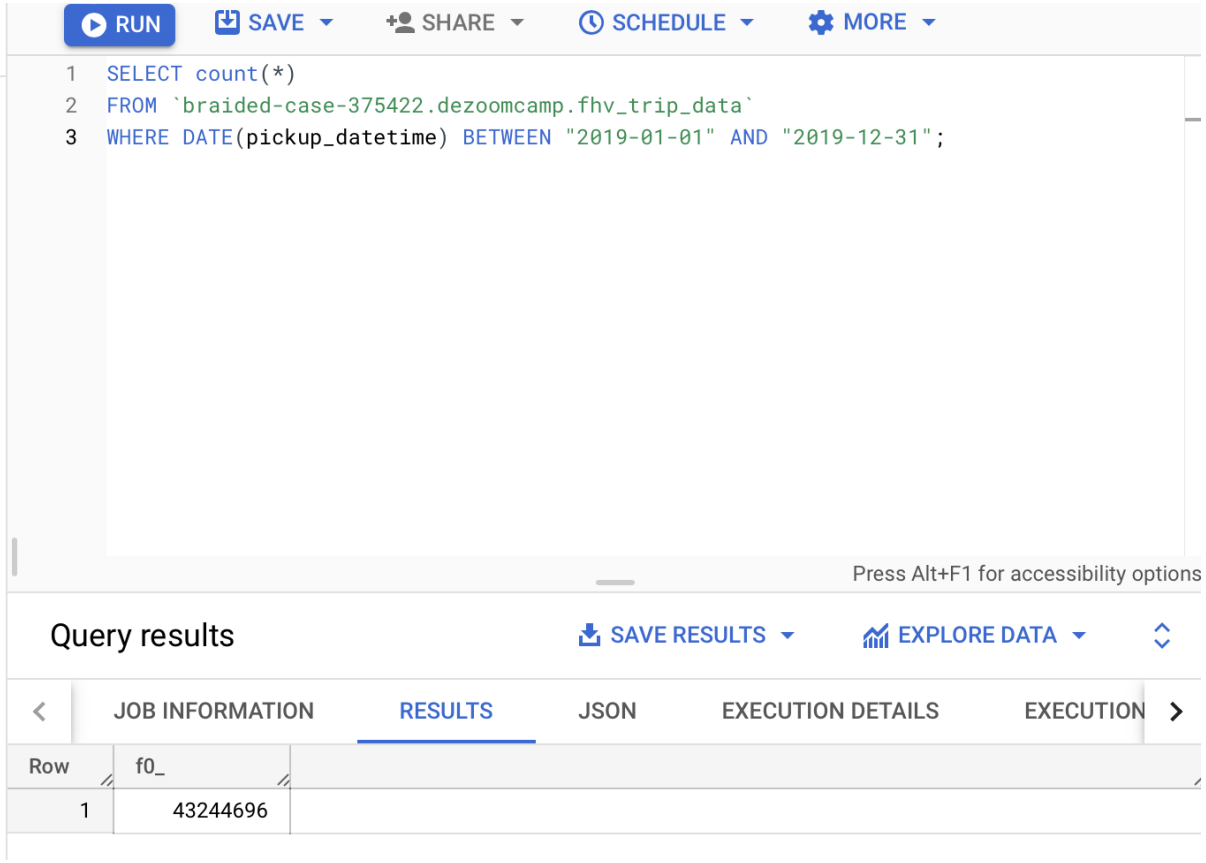
```
-- Count of fhv trips

SELECT count(*)

FROM `<project id>.dezoomcamp.fhv_trip_data`

WHERE DATE(pickup_datetime) BETWEEN "2019-01-01" AND "2019-12-31";
```

Or

```
-- Count of fhv trips
```

```
SELECT count(*) FROM `<project id>.dezoomcamp.fhv_nonpartitioned_trip_data`;
```

We get the result of 43,244,696



Question 2: Write a query to count the distinct number of affiliated_base_number for the entire dataset on both the tables. What is the estimated amount of data that will be read when this query is executed on the External Table and the Table?

- 25.2 MB for the External Table and 100.87MB for the BQ Table
- 225.82 MB for the External Table and 47.60MB for the BQ Table
- 0 MB for the External Table and 0MB for the BQ Table
- **0 MB for the External Table and 317.94MB for the BQ Table**

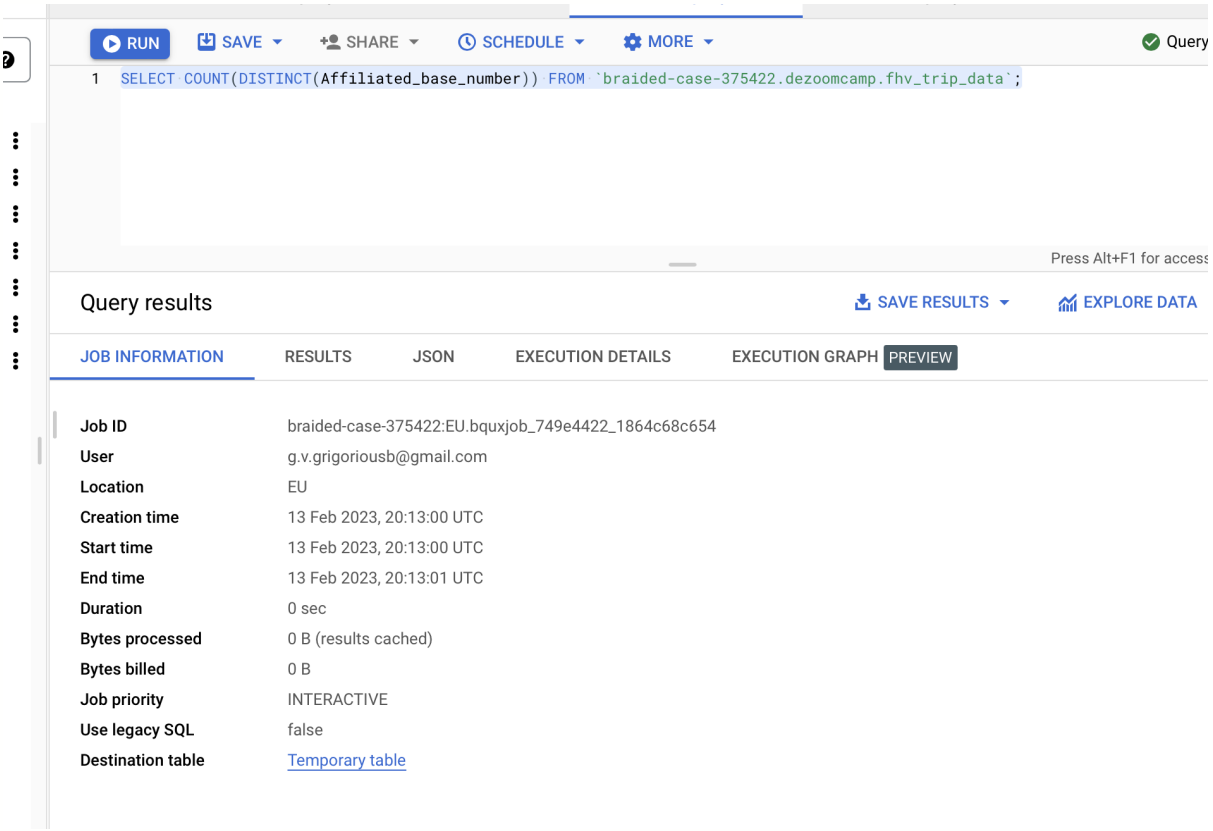
Question 2. Solution

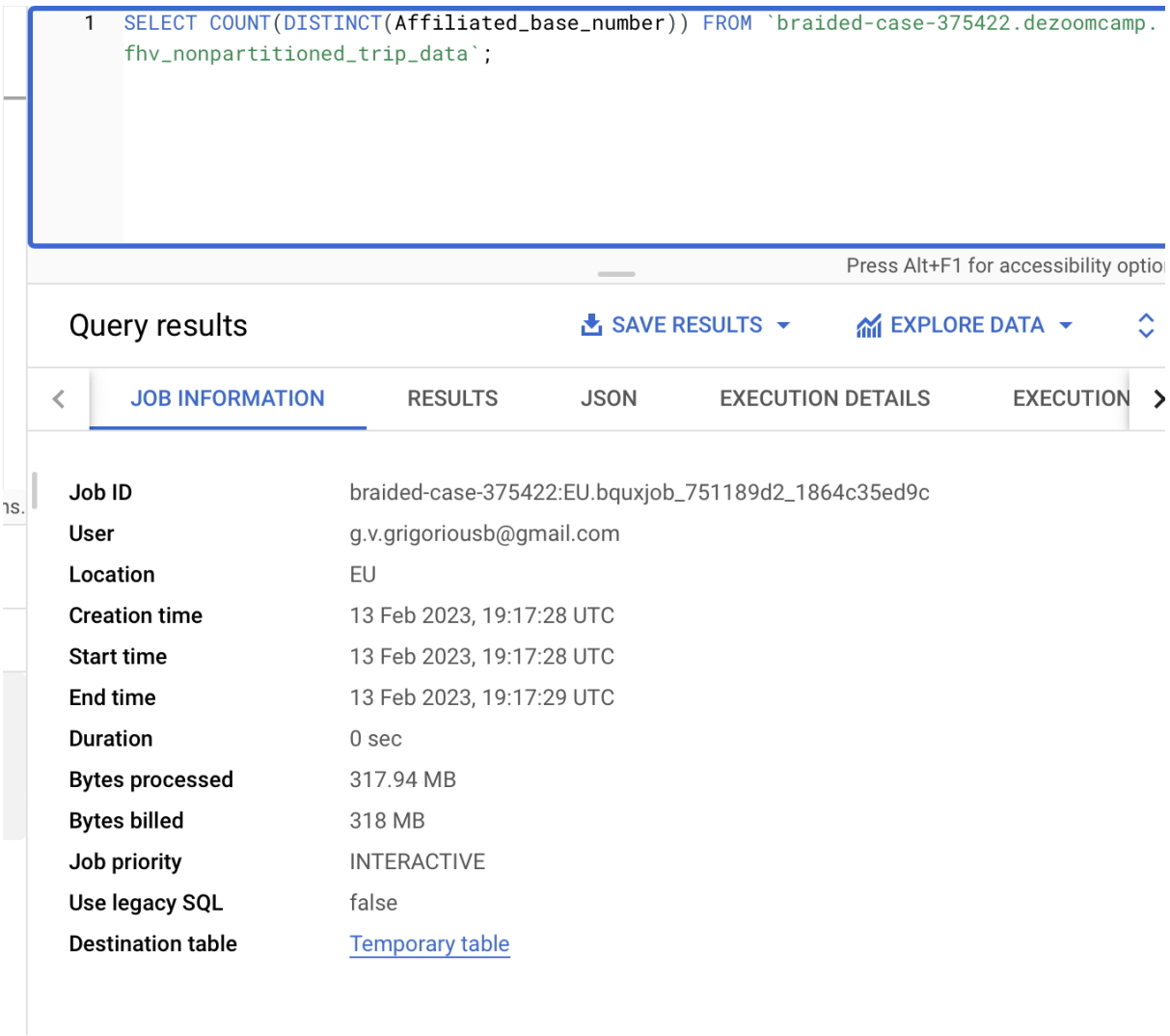
The correct answer is

- **0 MB for the External Table and 317.94MB for the BQ Table**

By running the following SQL Commands we get the above-mentioned answer

```
SELECT COUNT(DISTINCT(Affiliated_base_number)) FROM `<project
id>.dezoomcamp.fhv_trip_data`;
```





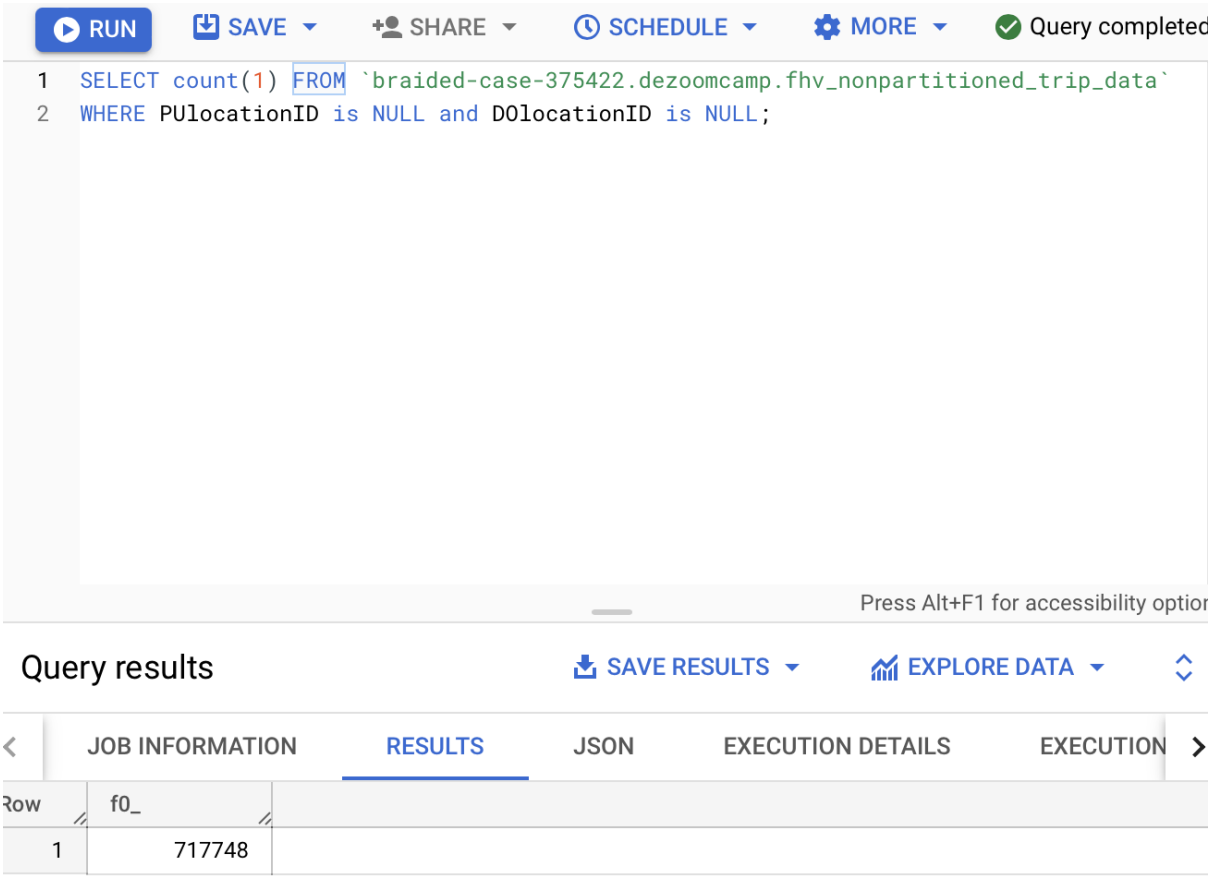
Question 3: How many records have both a blank (null) PUlocationID and DOlocationID in the entire dataset?

- **717,748**
- 1,215,687
- 5
- 20,332

Question 3. Solution

The correct answer is **717,748.** By running the SQL command

```
SELECT count(1) FROM `<project id>.dezoomcamp.fhv_nonpartitioned_trip_data`
WHERE PUlocationID is NULL and DOlocationID is NULL;
```



Question 4: What is the best strategy to optimize the table if query always filter by pickup_datetime and order by affiliated_base_number?

- Cluster on pickup_datetime Cluster on affiliated_base_number
- **Partition by pickup_datetime Cluster on affiliated_base_number**
- Partition by pickup_datetime Partition by affiliated_base_number
- Partition by affiliated_base_number Cluster on pickup_datetime

Question 4 Solution

The correct answer is **Partition by pickup_datetime and cluster on affiliated_base_number**

Partitioning my table by pickup_datetime will allow BigQuery to only scan the relevant partitions for my query, reducing the amount of data that needs to be read and improving query performance. This is particularly useful if data is time-sensitive and you run queries for specific time ranges.

Clustering on affiliated_base_number will physically store the data in a way that makes it more efficient to retrieve records that are sorted by affiliated_base_number. Since my queries are filtering by pickup_datetime and ordering by affiliated_base_number, this strategy will provide significant performance benefits by reducing the amount of data that needs to be read and returning the results in the order you need.

With this approach, I am optimising my table for the specific queries that I am running, making my queries faster and more efficient.

Question 5: Implement the optimized solution you chose for question 4. Write a query to retrieve the distinct affiliated_base_number between pickup_datetime 2019/03/01 and 2019/03/31 (inclusive). Use the BQ table you created earlier in your from clause and note the estimated bytes. Now change the table in the from clause to the partitioned table you created for question 4 and note the estimated bytes processed. What are these values? Choose the answer which most closely matches.

- 12.82 MB for non-partitioned table and 647.87 MB for the partitioned table
- **647.87 MB for non-partitioned table and 23.06 MB for the partitioned table**
- 582.63 MB for non-partitioned table and 0 MB for the partitioned table
- 646.25 MB for non-partitioned table and 646.25 MB for the partitioned table

Question 5 Solution

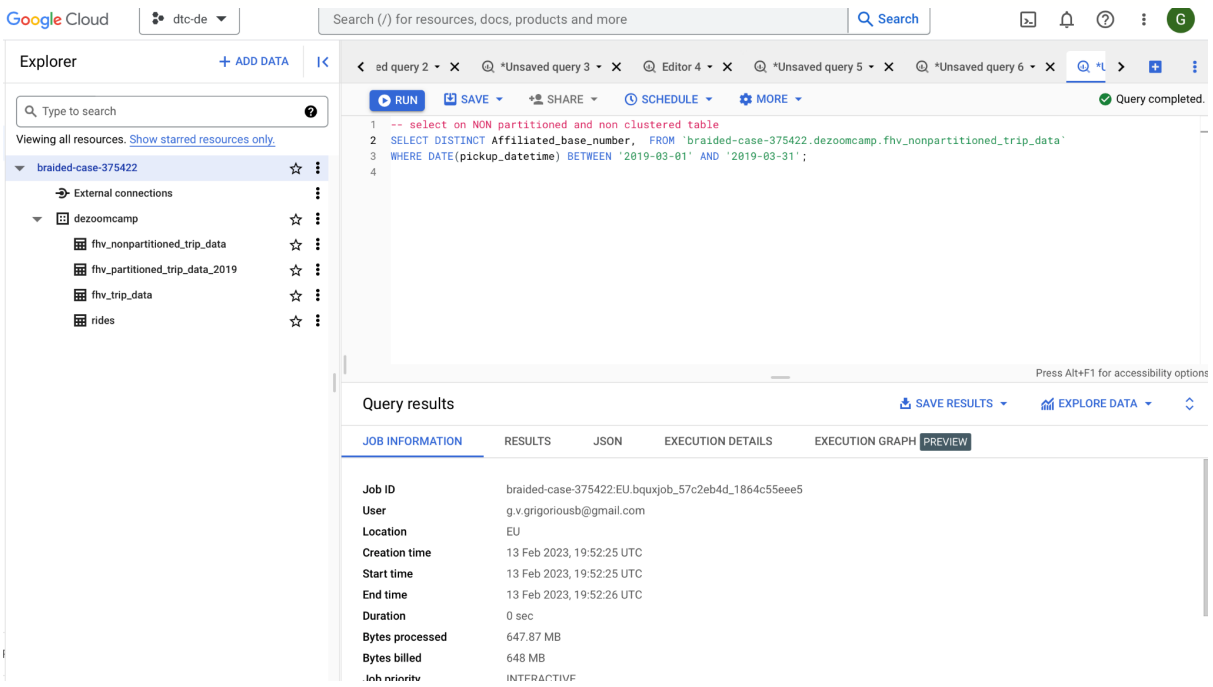
The correct answer is **647.87 MB for non-partitioned table and 23.06 MB for the partitioned table.**

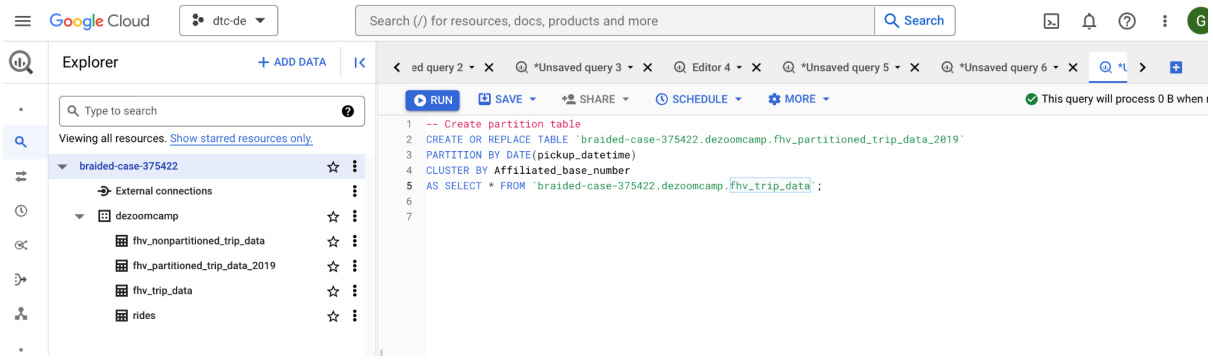
First, I created the partitioned table

```sql
-- Create partition table
CREATE OR REPLACE TABLE `<project id>.dezoomcamp.fhv_partitioned_trip_data_2019`
PARTITION BY DATE(pickup_datetime)
CLUSTER BY Affiliated_base_number
AS SELECT * FROM `<project id>.dezoomcamp.fhv_trip_data`;
```

Then, I selected the table that is neither partitioned nor clustered, which gives 647.87 MB Bytes processed

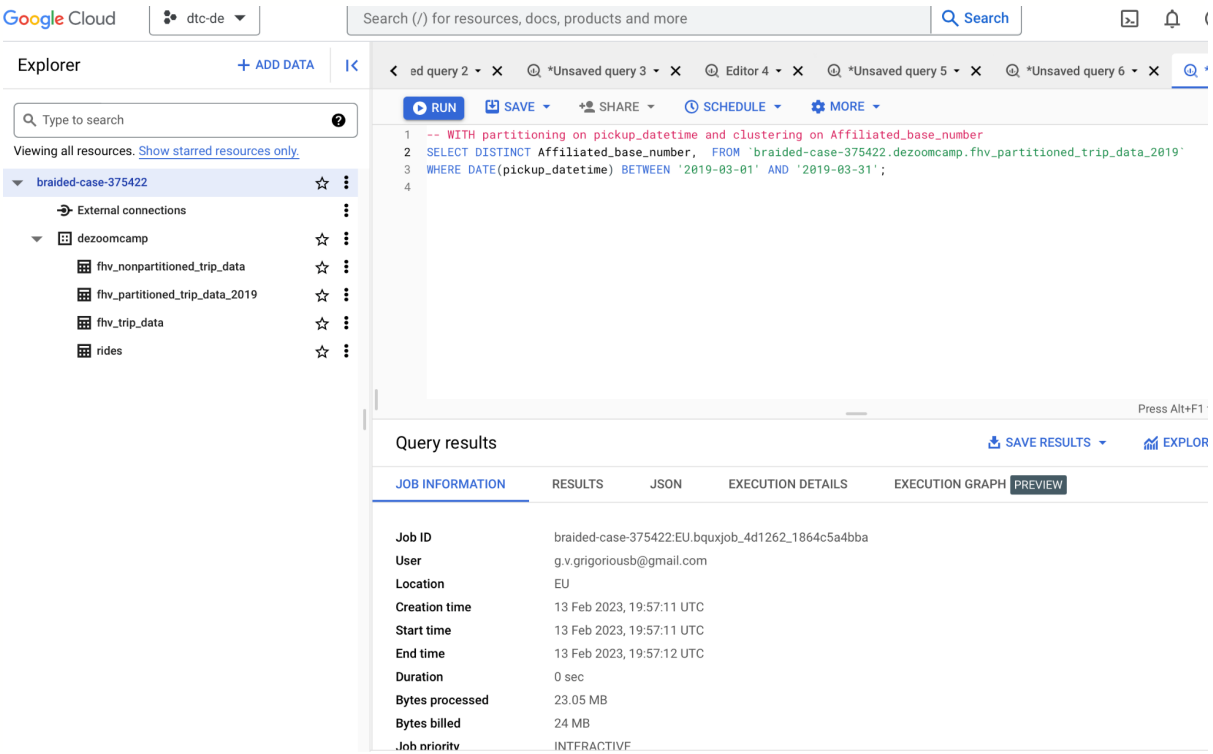
```sql
-- WITH partitioning on pickup_datetime and clustering on Affiliated_base_number
SELECT DISTINCT Affiliated_base_number,  FROM `<project id>.dezoomcamp.fhv_nonpartitioned_trip_data`
WHERE DATE(pickup_datetime) BETWEEN '2019-03-01' AND '2019-03-31';
```





Then, I selected the table with partitioning on pickup_datetime and clustering on Affiliated_base_number, which gives 23.05 MB Bytes processed

```
-- WITH partitioning on pickup_datetime and clustering on Affiliated_base_number
SELECT DISTINCT Affiliated_base_number,  FROM `<project id>.dezoomcamp.fhv_nonpartitioned_trip_data`
WHERE DATE(pickup_datetime) BETWEEN '2019-03-01' AND '2019-03-31';
```



Question 6:

Where is the data stored in the External Table you created?

- Big Query
- **GCP Bucket**
- Container Registry
- Big Table

Question 6 Solution

The correct answer is **GCP Bucket.** The data in an External Table in BigQuery is stored in a Google Cloud Storage (GCS) bucket or a Cloud Storage file system.

Question 7:

It is best practice in Big Query to always cluster your data:

- True
- **False**

Question 7 Solution

The correct answer is **False.** Clustering is a performance optimization in BigQuery that rearranges the physical storage of table data to optimize query performance by reducing the amount of data that needs to be read. While clustering can improve query performance, it is not always necessary, and it depends on the nature of your data and the types of queries you run.
For example, if your data is already sorted in the order that you need to query it, then clustering is not necessary because the data is already well organized for efficient querying. If your queries tend to filter data based on a specific set of columns, then clustering on those columns may provide performance benefits.
In general, whether or not clustering is necessary depends on the specific use case and data patterns. Before enabling clustering, it is important to understand the trade-offs and performance implications. If you're unsure, you can experiment with clustering to see if it provides performance benefits for your specific queries and data.