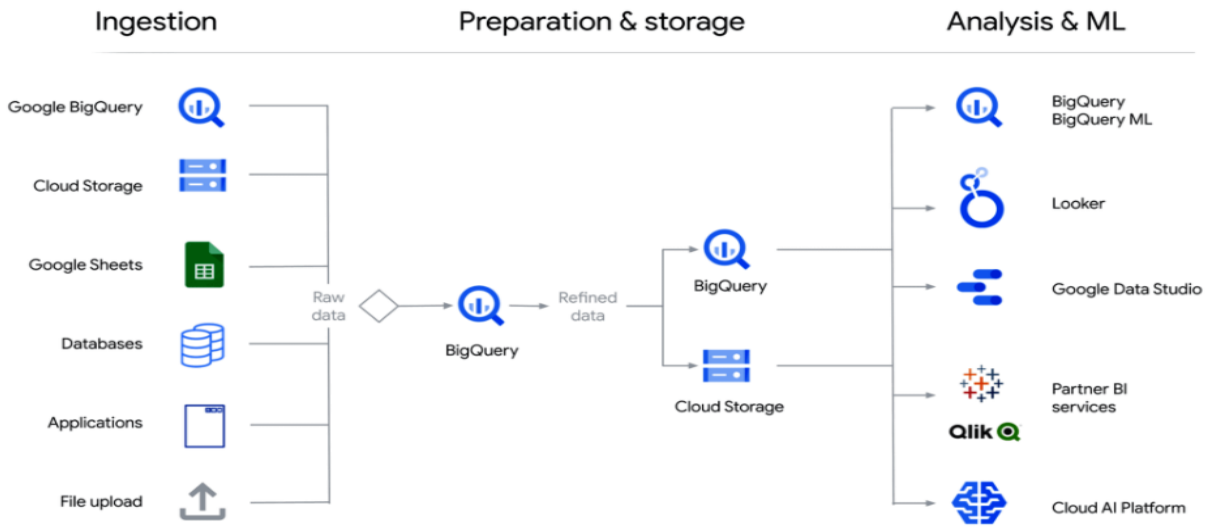
Google BigQuery

What is Google BigQuery?

BigQuery is a fully managed enterprise data warehouse that helps manage and analyse data with built-in features like machine learning, geospatial analysis and business intelligence. In layman terms, it is just a **BIG** database

BigQuery's serverless architecture lets us use SQL queries to answer organisation's biggest questions with zero infrastructure management. Its scalable, cost effective, distributed analysis engine lets us query terabytes in seconds and petabytes in minutes.

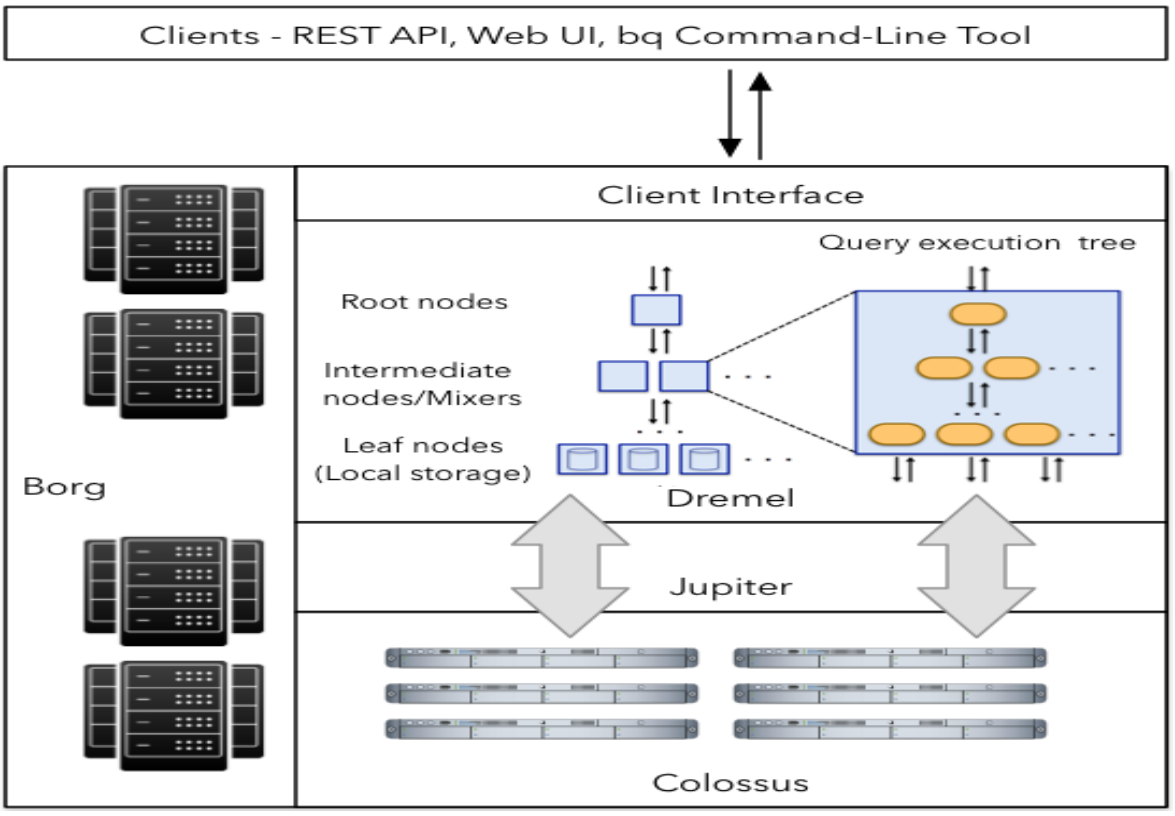


Working of BigQuery

BigQuery Architecture

Bigquery is built on Dremel technology (an interactive ad-hoc query system for analysing nested read only data). But bigquery is much more than that.Bigquery also has a serverless architecture that decouples storage and allows us to scale independantly.
This structure offers both flexibility and cost control.

It is different from traditional cloud data warehouse solutions or on-premise massively parallel processing systems. This approach allows customers of any size to bring their data into the data warehouse and start analysing their data using Standard SQL without worrying about database operations and system engineering.Dremel is just an execution engine for BigQuery. In fact, BigQuery service leverages Google's innovative technologies like Borg, Colossus, Capacitor, and Jupiter. As illustrated below, a BigQuery client (typically BigQuery Web UI or bg command-line tool or REST APIs) interacts with the Dremel engine via a client interface.

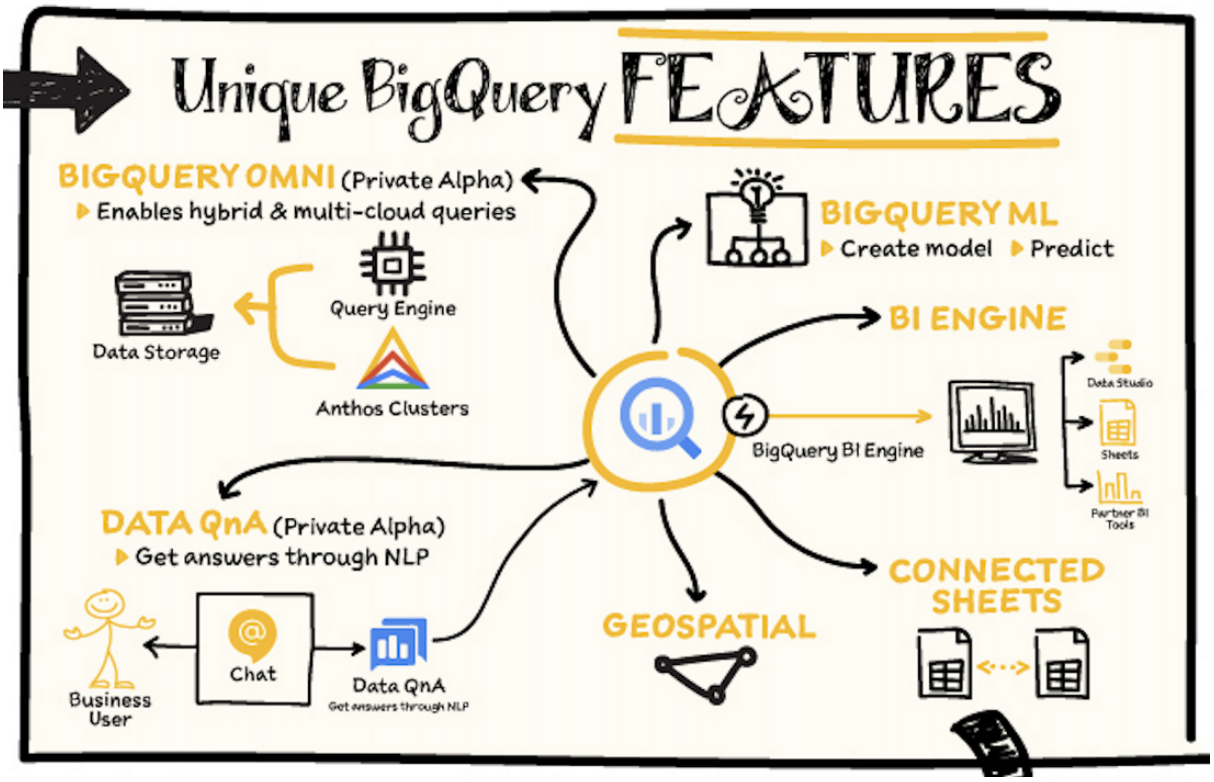


Architecture of BigQuery

Features of BigQuery

Bigquery offers a wide range of features that helps us get detailed insights from data in a very cost efficient manner.
Here is a list of bigquery features:

- Multi Cloud Functionality
- Built-in ML Integration
- Foundation for BI
- Geospatial Analysis
- Automated Data Transfer
- Free Access
- Partitioning and Clustering



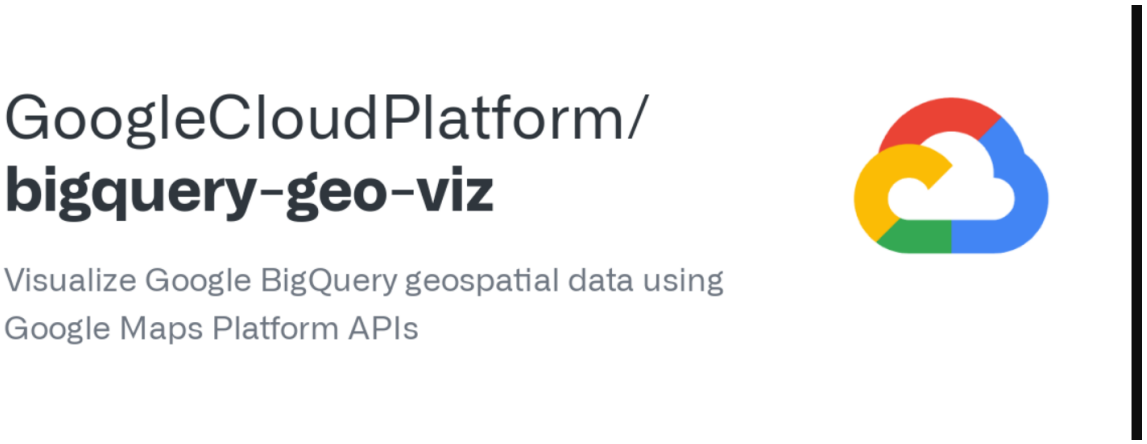
Unique features of BigQuery

Geospatial Analysis

What is Geospatial Analysis ?

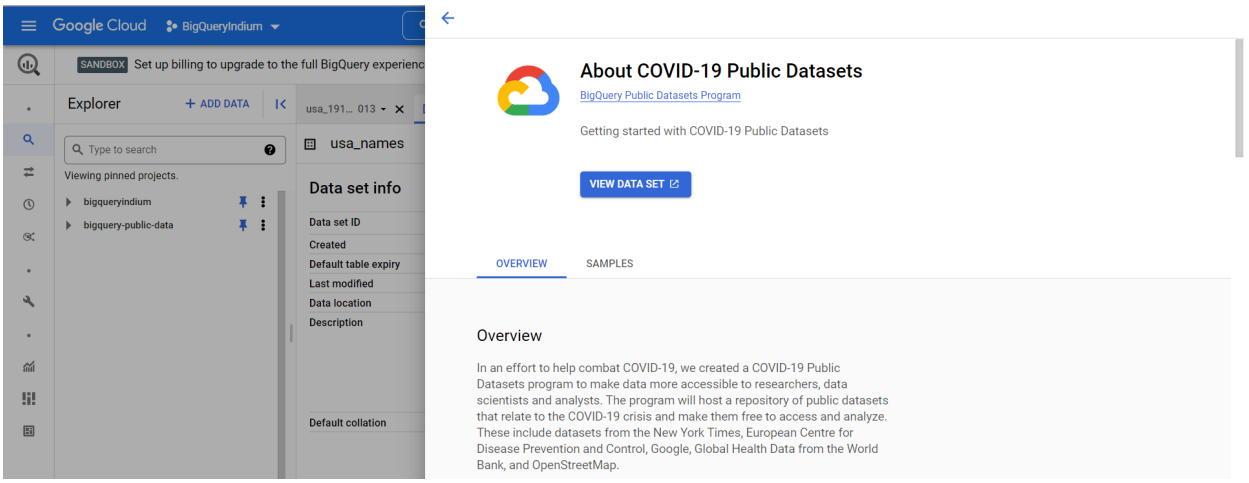
Geospatial Analytics highlights historical changes and current shifts by collecting, displaying, and manipulating Imagery and Geographic Information System (GIS) Data related to a specific location. GIS provides information about location and mapping. It functions by converting latitudes and longitudes columns into geographical points. The gathered information helps create Data Visualisations, such as Graphs, Maps, Stats, and Cartograms. These reports help the human brain to understand the distance, proximity, and contiguity not visible in large datasets.

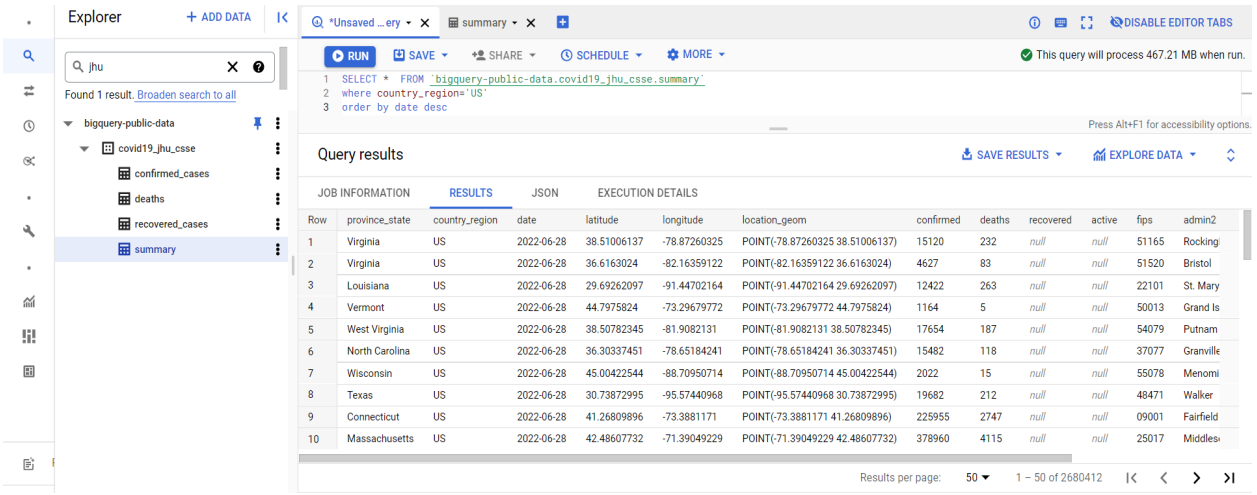
Tools Supported by BigQuery for Geospatial Analytics - BigQuery GeoViz, Google Data Studio, Google Earth Engine and Jupyter Notebooks.



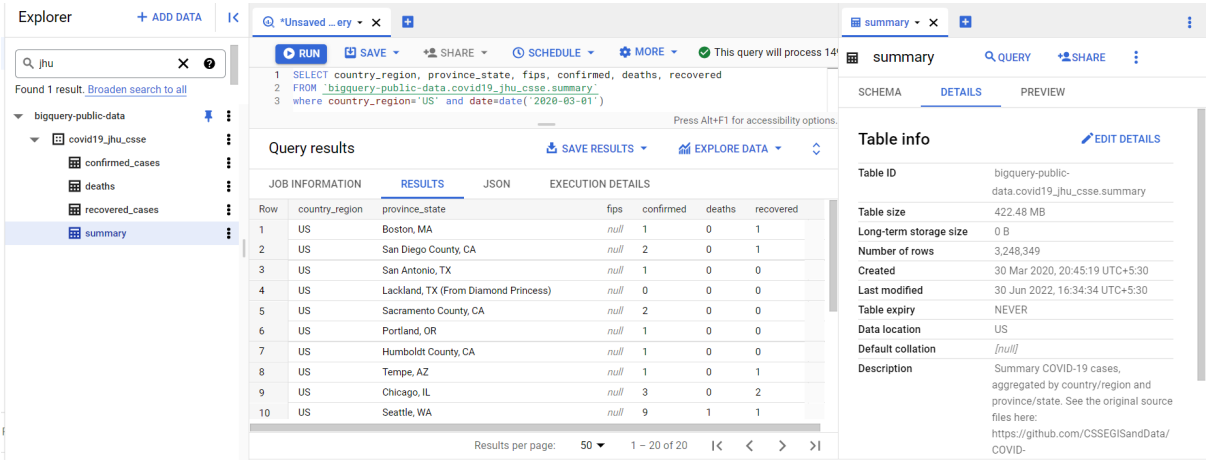
BigQuery Geospatial analysis

Now, let's try to explore and understand the covid-19 dataset available on bigquery and later visualise that on a graph.



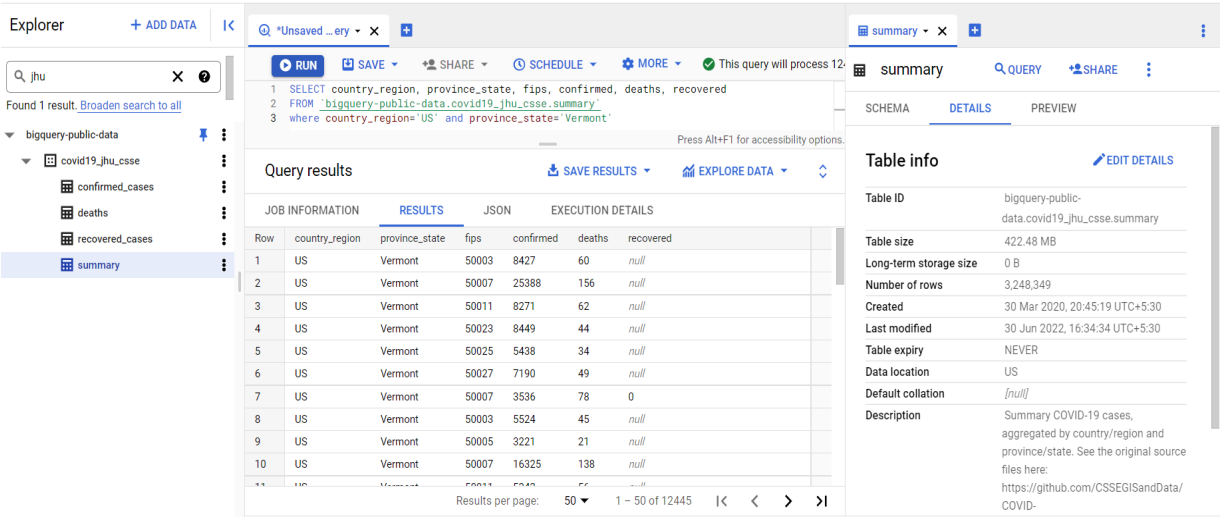


This query selects only rows that are in US country_region and orders it by most recent data.

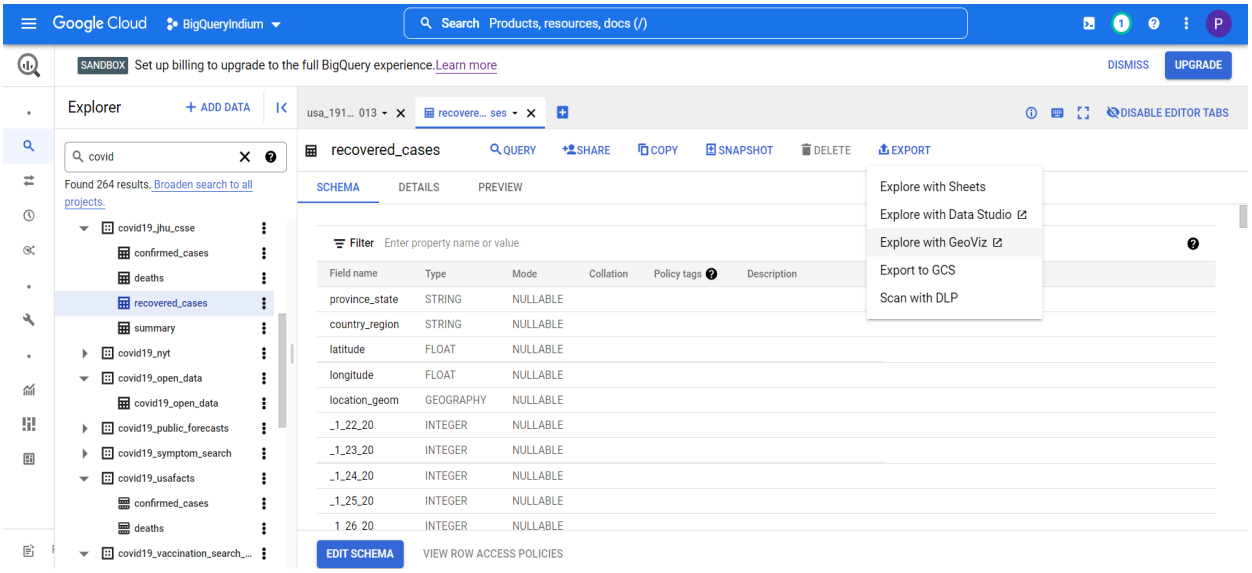


NOTE -: **fips** is an id made with first 2 digits of province state code and last 3 digits with counties code.

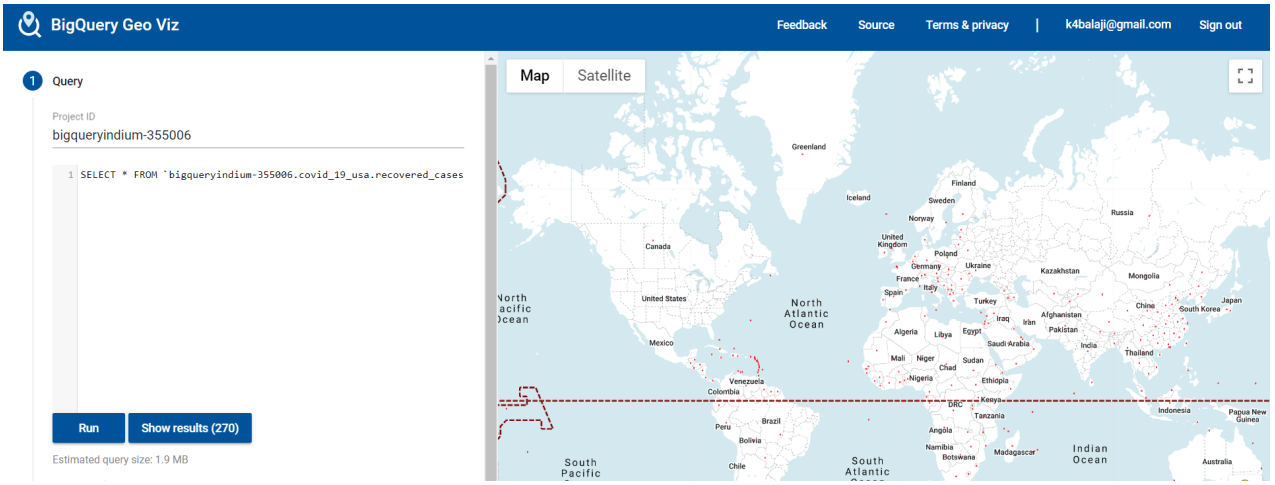
This query shows the confirmed cases, deaths and recovered cases only on **1st of March, 2020** and in the US only.



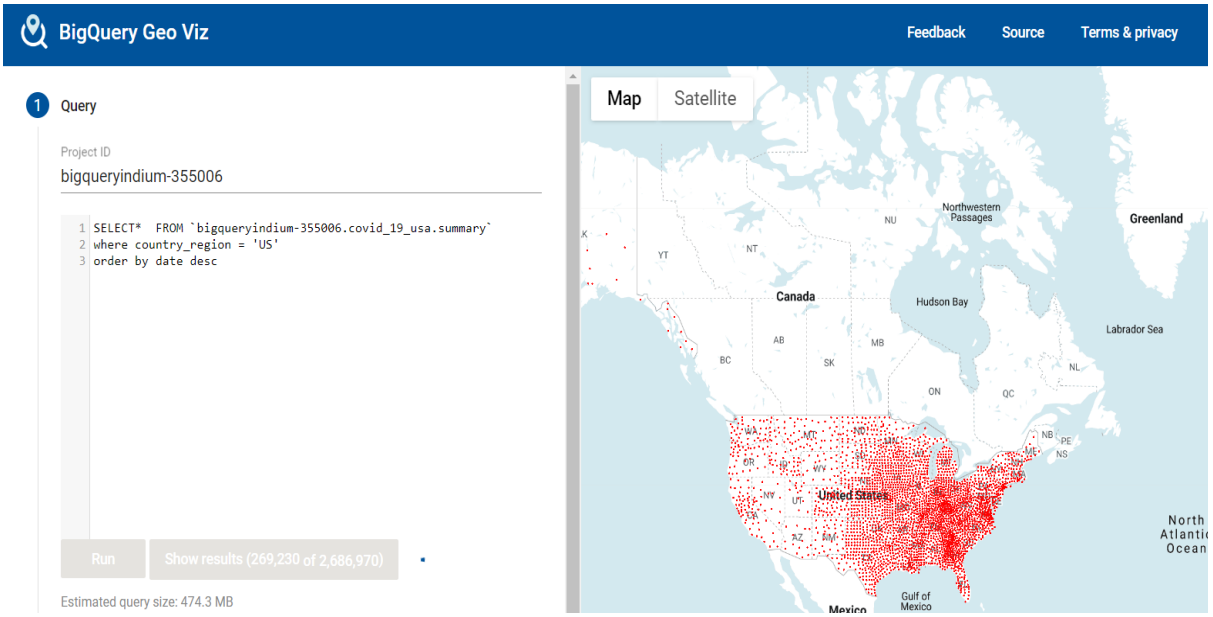
This query shows the country, province_state, confirmed cases, deaths and recovered cases only in the **country US** and in **Vermont** only.



It can be noticed that since there is a data type **GEOGRAPHY**, bigquery allows us to explore this dataset on a map. To do this, we have to click on **EXPORT** and then **Explore with GeoViz**.

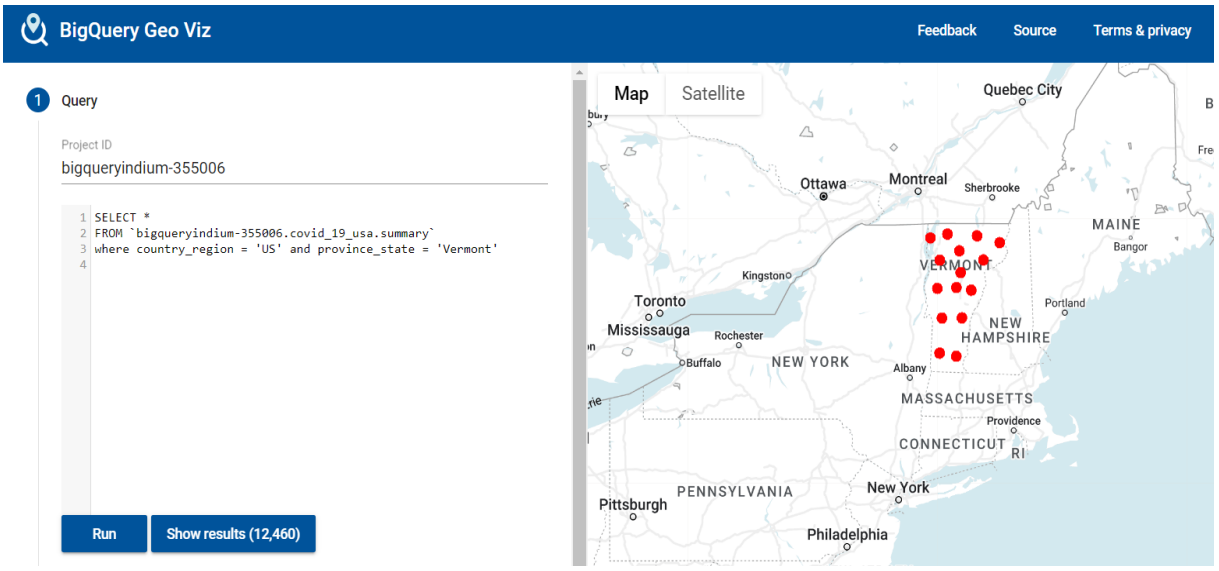


The tiny red dots indicate the query results on the map.



Now let's look at more queries.

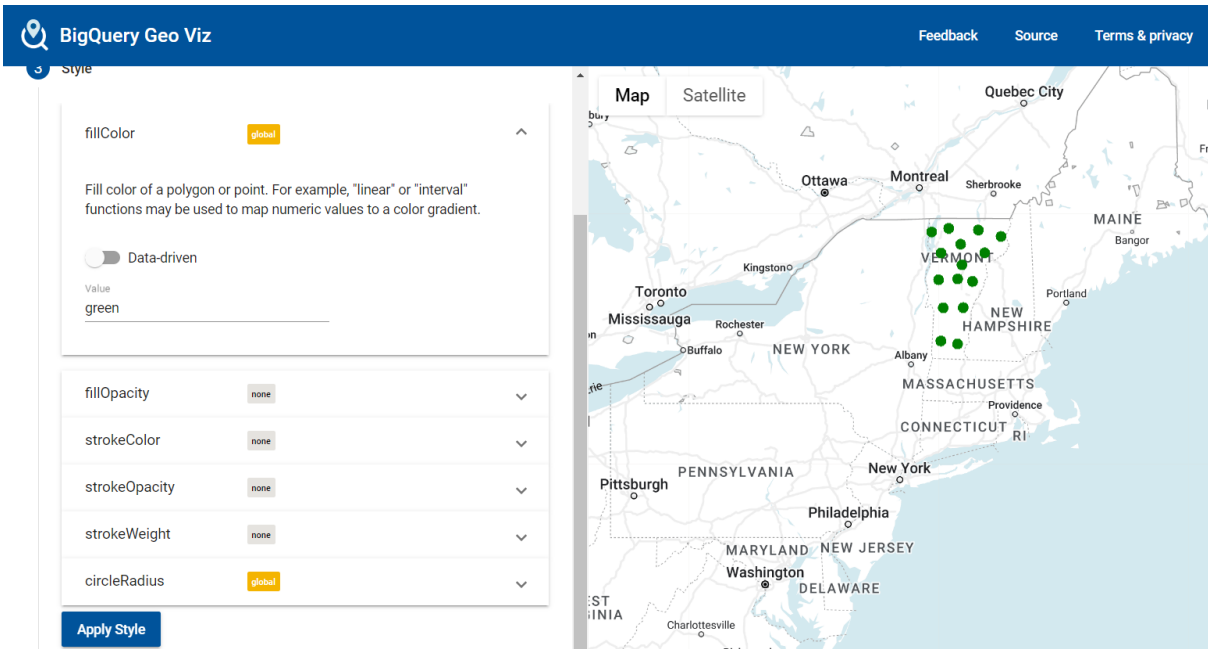
As we can see the red dots are presented only within the US.



The red dots are only within Vermont.

Note that we can also change the colour of the dots. Now, let's change it to green and have a look.

We can also increase or decrease the radius of the points.



GEOSPATIAL ANALYSIS WITH STARBUCKS DATABASE

Let us explore a few problem statements with the starbucks database. I have downloaded the dataset from Kaggle. The said database has the information on the location of its stores worldwide.

For example,

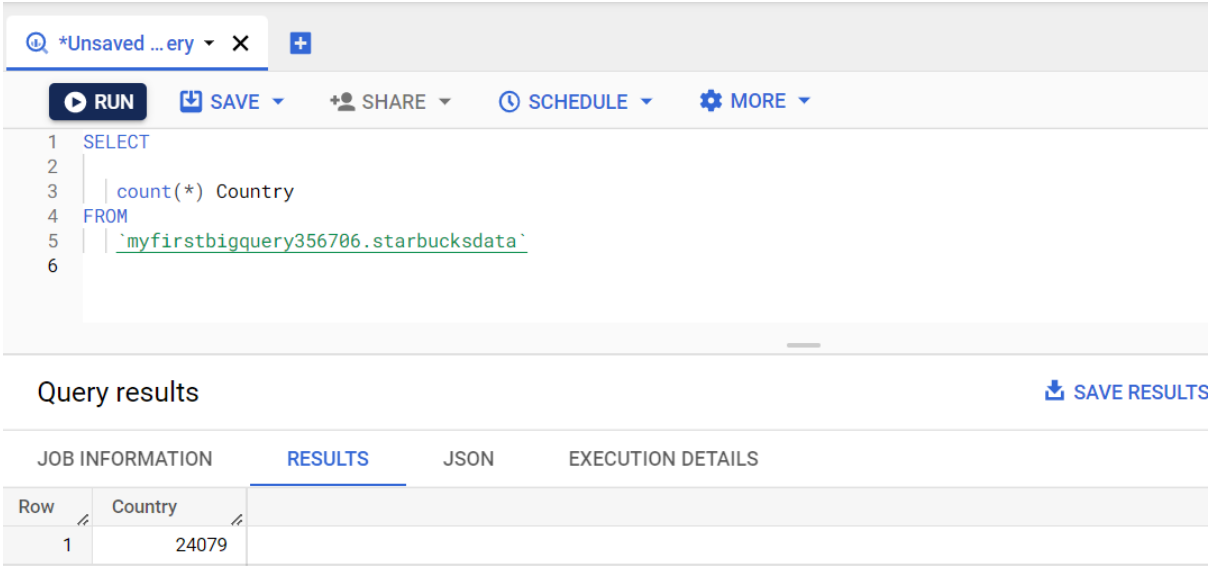
As an induvidual, I am interested in investing in a starbucks store in my hometown of Regina, Canada. Here are a few problem statements that I would need to solve.

1. How many countries in the world have starbucks?
2. How many stores does Canada have?
3. Find the number of distinct cities.
4. Figure out the type of ownership.
5. No. of chains in a city(In this case, City of Regina)
6. What is the distance between each store?

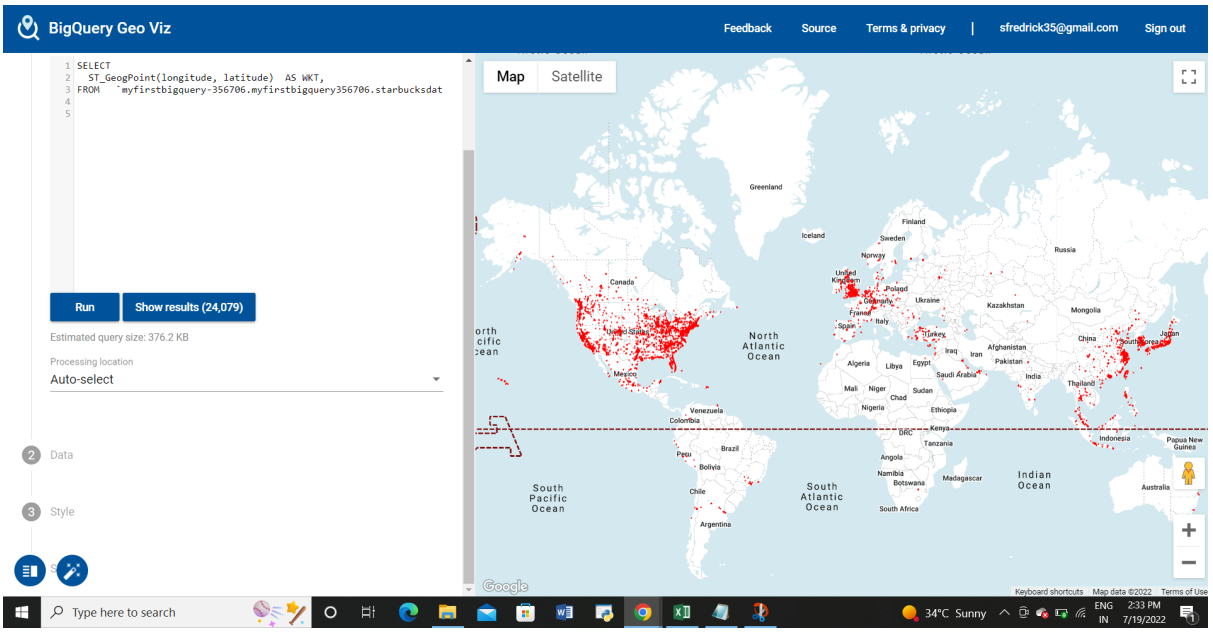
Starbucks started as a roaster and retailer of whole bean and ground coffee, tea and spices with a single store in Seattle's Pike Place Market in 1971. The company now operates more than 24,000 retail stores in 70 countries.

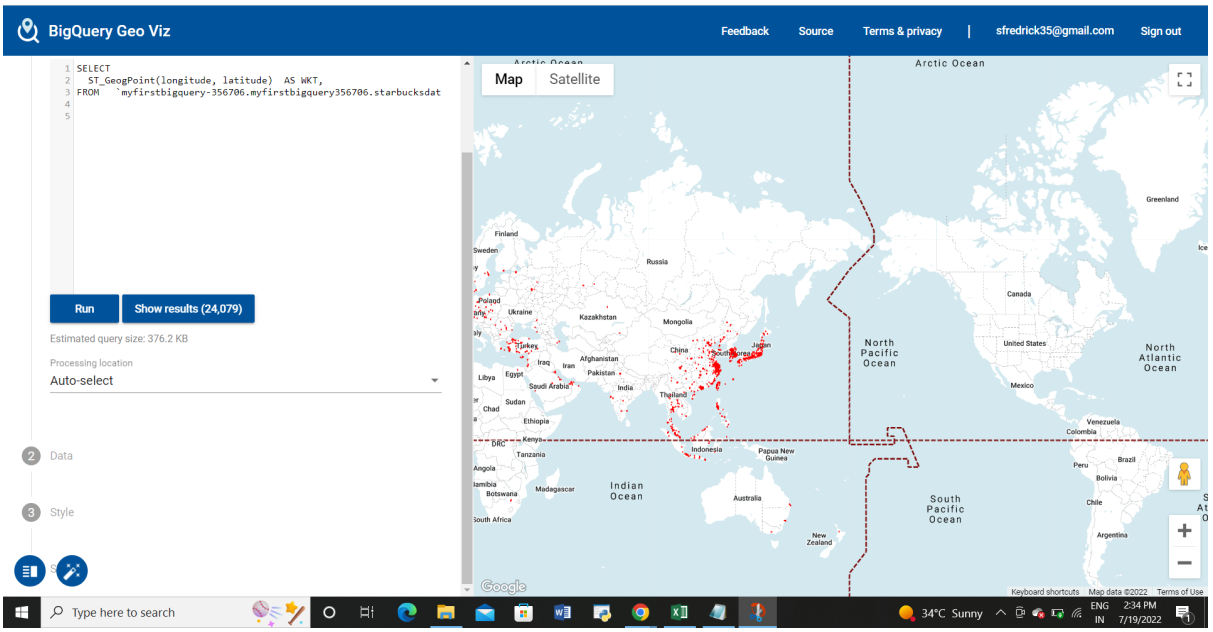
Now, let's find the answers!

Based on our query below, we can see that there are 24,079 stores in the world.



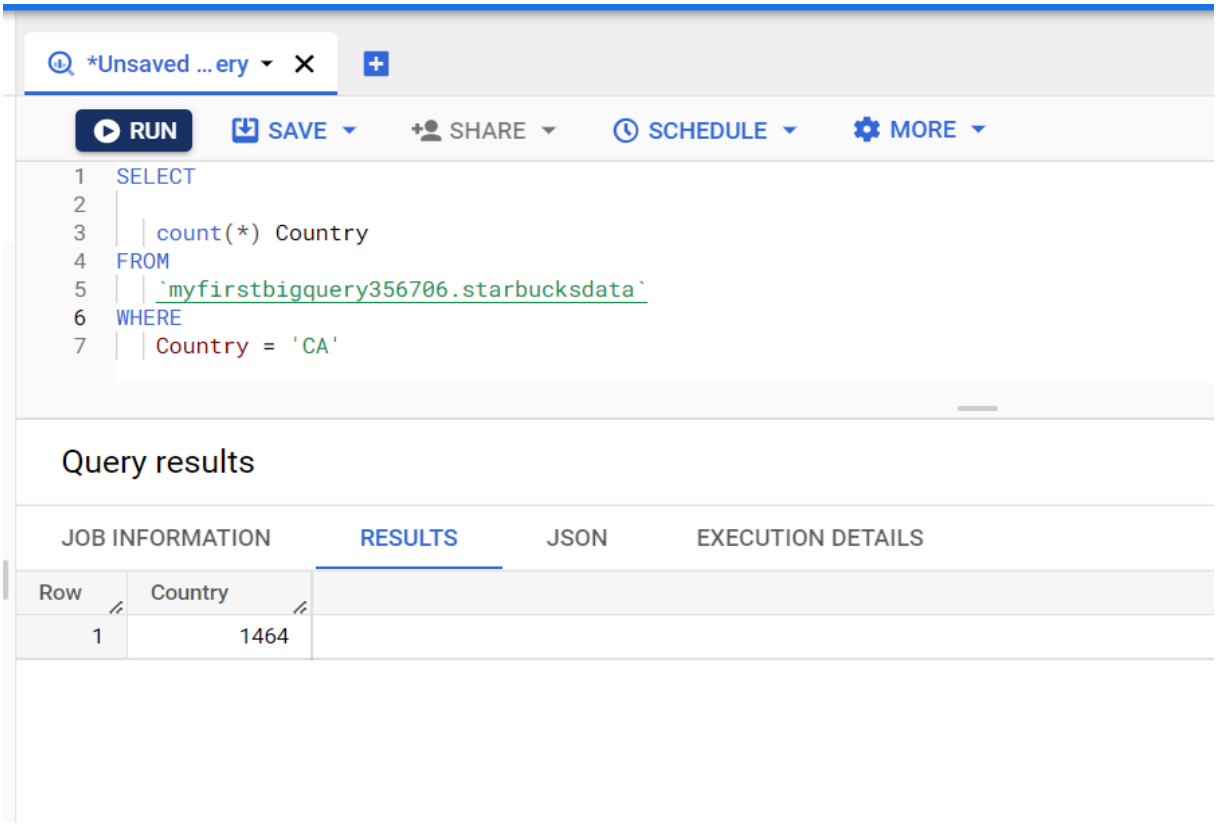
One more interesting thing about bigquery is that we can explore the dataset on maps using GeoViz. Let's take a look below!

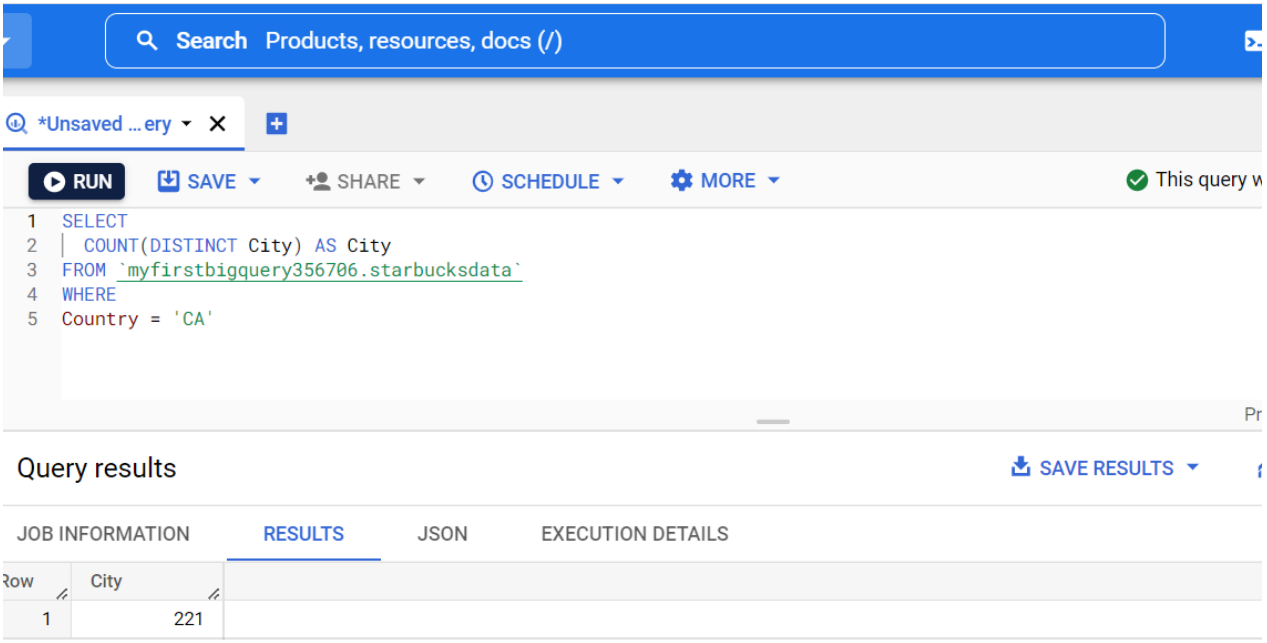




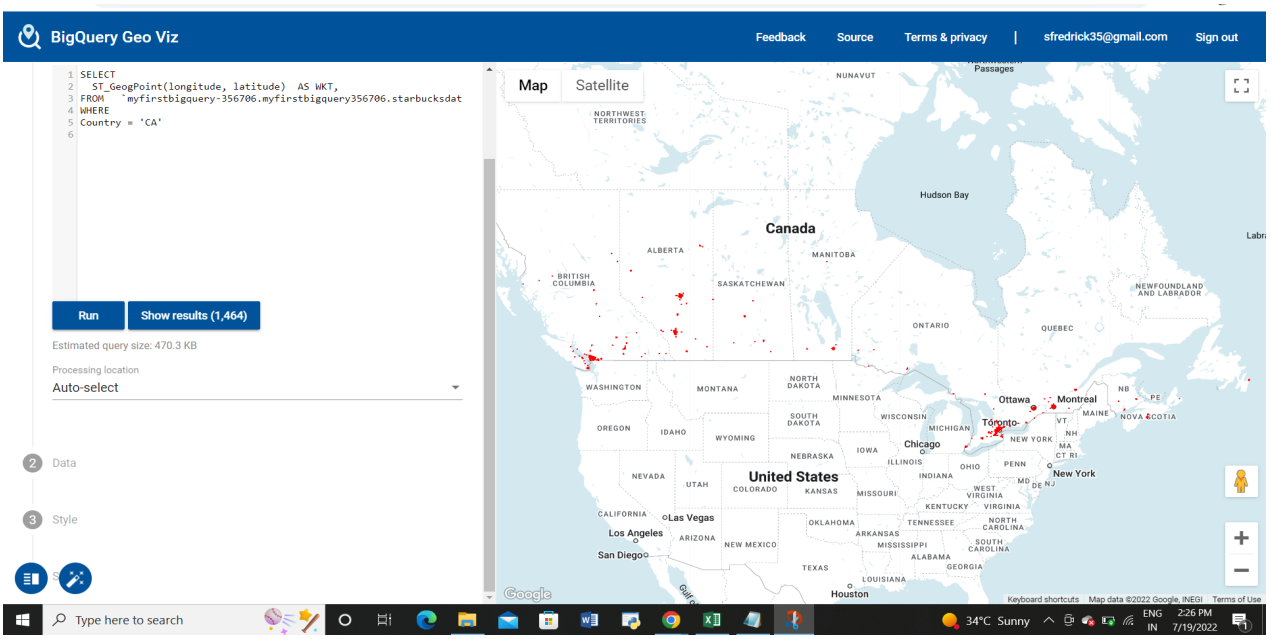
The red colouring shows us the locations of the stores.

I need to know the number of stores in Canada as I am looking to invest in a store here. Here we can see that 1464 stores are operating in Canada.

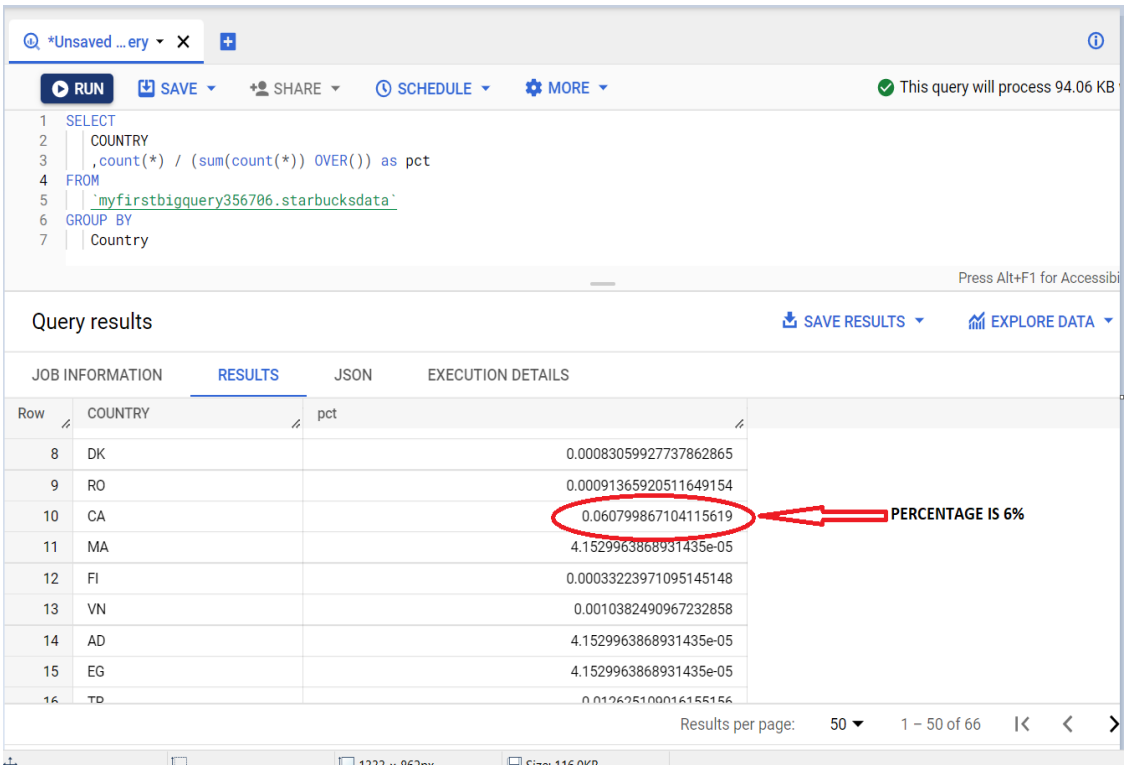




These 1464 stores are present in 221 cities. This can be plotted on a map as well.



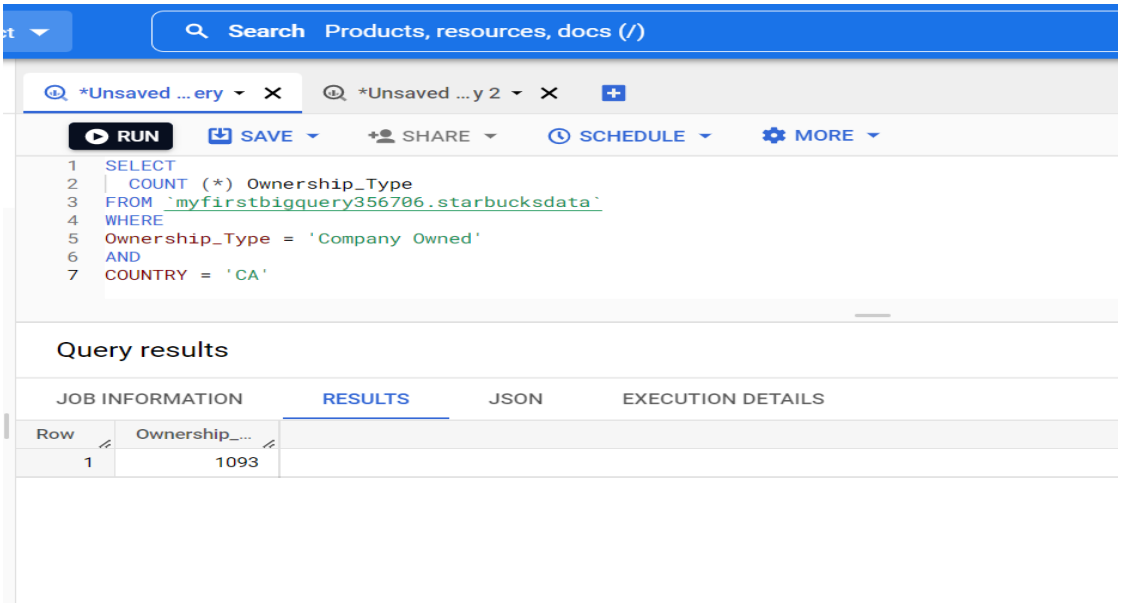
Canada has 6% of the stores in its country. This can be calculated.

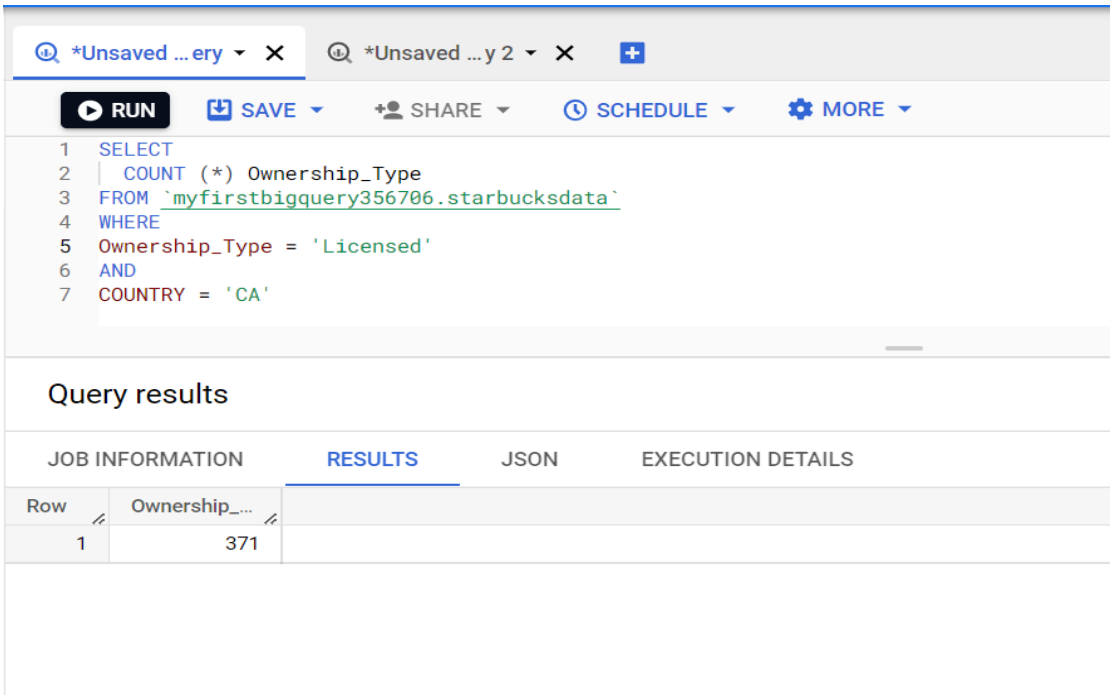


When you explore the dataset, you might find an interesting factor. Not all stores are company owned. Which is not surprising since Starbucks is a multinational chain of coffeehouses.

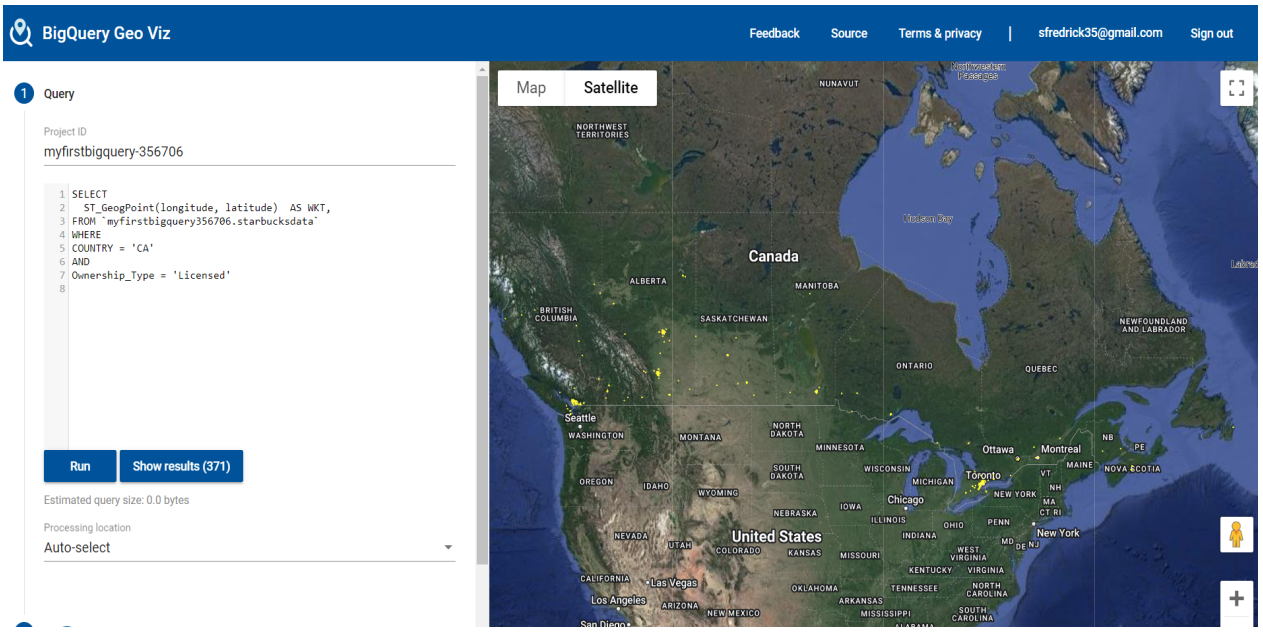
The types of ownership are Licensed, company owned, franchise and joint venture. Let's take a look at the types of ownership in Canada. (Which is Licensed and Company owned. Since it is a small number, we can easily filter it on excel).

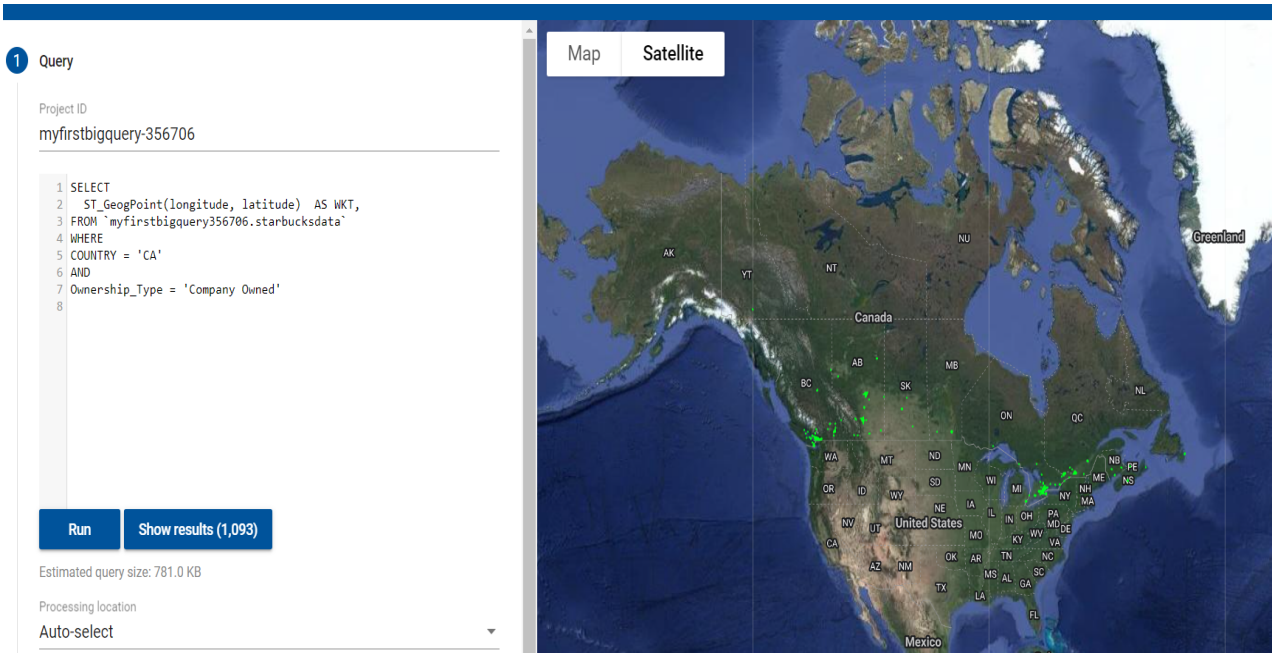
There are about 1093 company owned stores and 371 licensed stores in Canada.





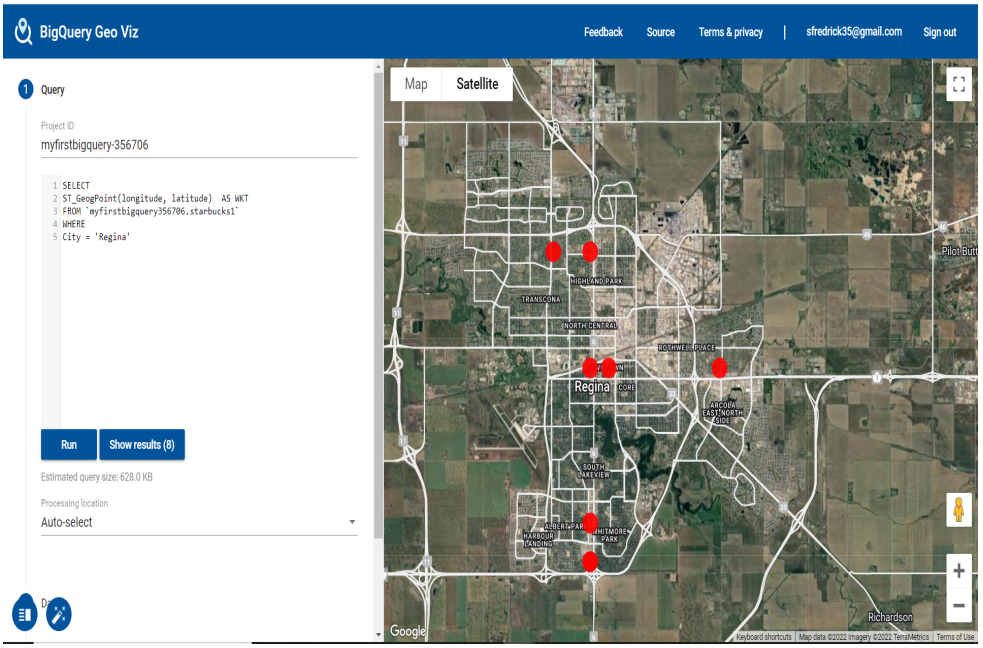
The yellow dots in the map represent the licensed stores and the green dots represent company owned stores.



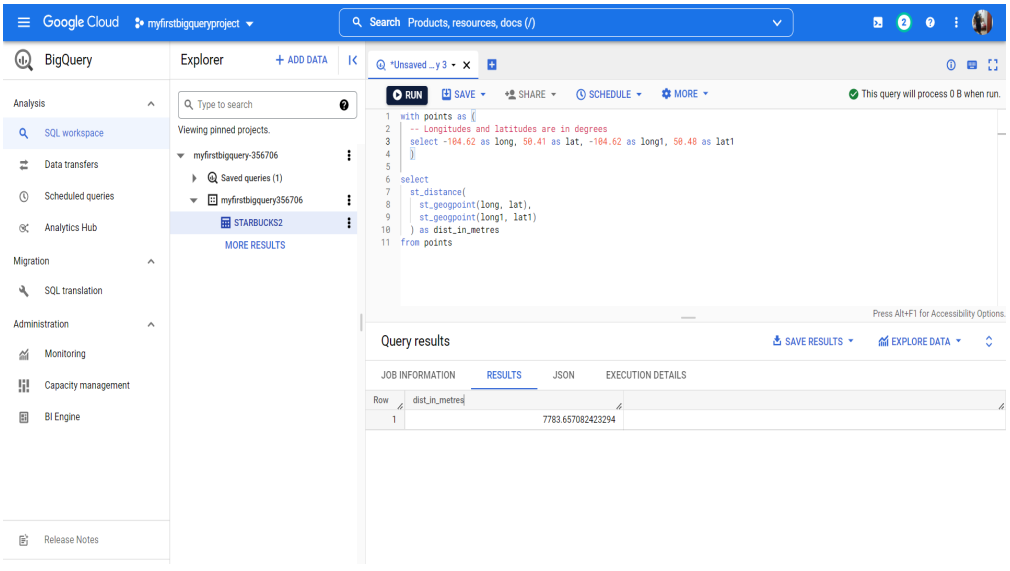


In this example, I want to open a store in the city of Regina.
It is the capital of the province of Saskatchewan. It is the second largest city in the province with a growing population. With an area of 12,931km, it has about 9 stores in the city.

Below is the map view of Regina with the store's location.



Let us find the distance between the stores.



The distance between the stores is approximately 7800 metres.

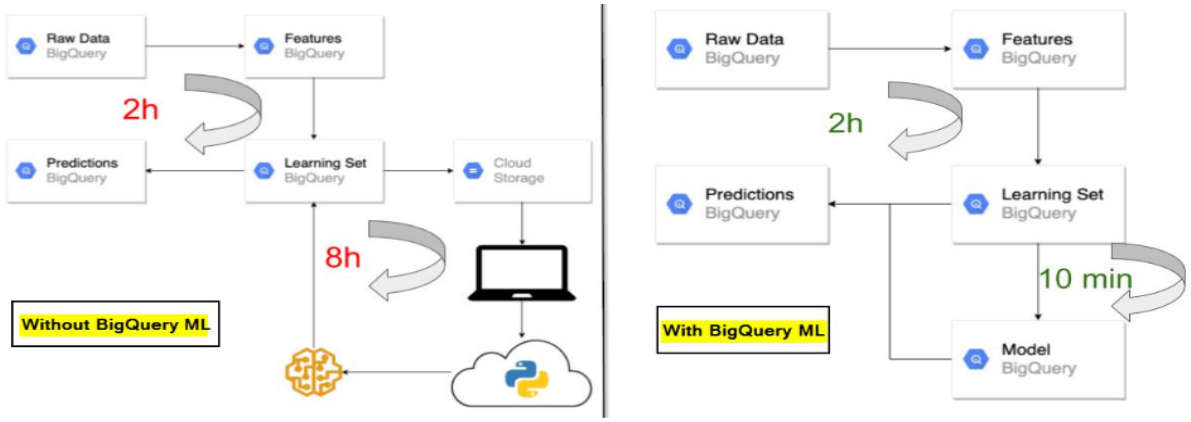
Since the stores come under the same coordinates, the distance between them will be more or less similar.

Et Voila! We have found out the basic answers that we need using geospatial analysis and Bigquery.

BigQuery ML

Google BigQuery ML is a set of tools and extensions that enables users to create, train and execute Machine Learning models in Google BigQuery using standard SQL queries. It eliminates the need for data movement by allowing users to directly create Machine Learning models into the Data Warehouse.

Key Features of Google BigQuery ML - Automatically Generate ML Models, Eliminates Data Transfer and Encrypted Models, supports linear and logistic regression, k-means, matrix factorization, time series, XGBoost, AutoML tables, etc.



Working of Machine learning With and without Bigquery

Different Models Supported in Google BigQuery ML

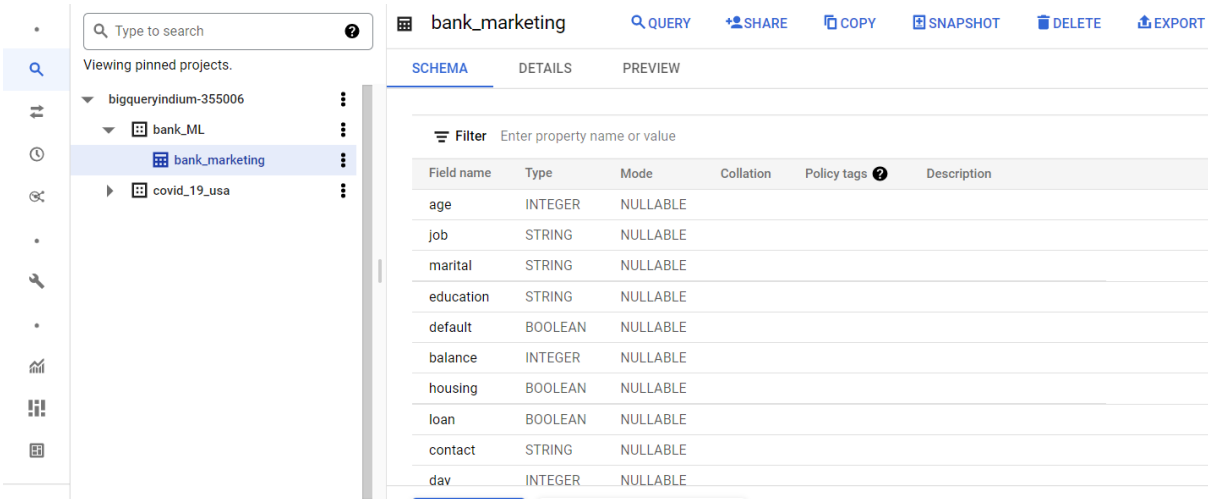
Various Machine Learning models are used to train predict data. All the Machine Learning models supported by Google BigQuery ML are listed below:

- **Linear Regression:** Linear Regression is the most basic model that is used for forecasting. It uses statistical calculations to make predictions. Google BigQuery ML provides this Machine Learning model to forecast trends based on data. Linear Regression is used for real-valued label prediction.
- **Logistic Regression**: Logistic Regression is used for the classification. When the required output is in YES or NO form, Logistic Regression delivers better results than Linear Regression because it converts every result in 1 or 0 i.e., YES or NO.
    - **Binary Logistic Regression:** It is used when the required output only has 2 outcomes meaning YES or NO. Binary-class classification can have only two possible values, 1 and 0.
    - **Multiclass Logistic Regression**: Classification can have multiple outcomes. Google BigQuery ML supports Multiclass Logistic Regression to allow users to solve real-world problems based on classification. It uses a multinomial classifier with a cross-entropy loss function to train Multiclass Logistic Regression models.
- **K-Means Clustering**: K-means clustering model is used for data segmentation. It uses non labeled data and groups similar data points in one group. Users can use the K-means clustering model with the CREATE MODEL statement with the KMEANS model type.

- **Matrix Factorization**: Google BigQuery ML allows users to quickly create and train models. The matrix Factorization model is used to create product recommendation systems. Users can use past customer behavior data with their product purchase data to build a robust recommendation system using SQL.
- **Time Series**: It is used to perform forecasts on time-series data. Google BigQuery ML eliminates the tedious task to handle anomalies in data like holidays, seasonality, etc. With the help of the Time Series model, users can create many time series models, test them and use them for forecasting.

Let us look at a simple Linear Regression model.

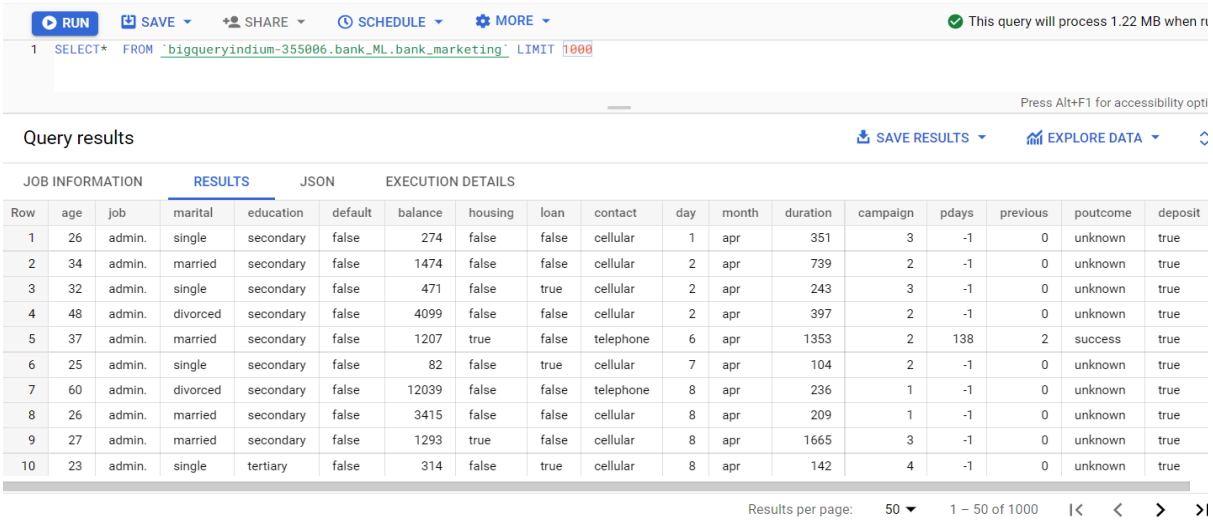
Problem statement 1 - Bank marketing (Dataset from Kaggle)



Field names - age, job, marital, education, default, balance, housing, loan, contact, day, month, duration, campaign, pdays, previous, outcome, deposit

AIM - Predict whether a customer will subscribe to a term deposit or not based on the provided information

Preview of the dataset.



Now, let's check out what's the distribution of customers subscribing and those not to the deposit.

RUN SAVE SHARE SCHEDULE MORE

```
1  select count(*), deposit
2  FROM `bigqueryindium-355006.bank_ML.bank_marketing`
3  group by deposit
4
```

Query results

JOB INFORMATION | RESULTS | JSON | EXECUTION DETAILS

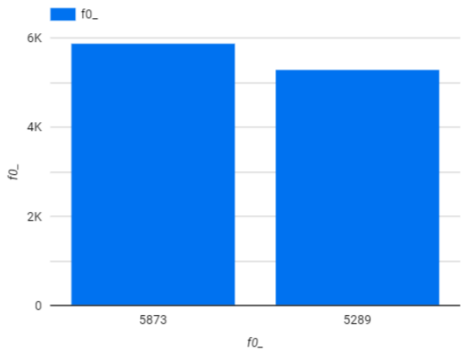
low	f0_	deposit
1	5289	true
2	5873	false

Now, on exploring the obtained result on Data Studio,

bank_marketing

	deposit	f0_
1.	false	5,873
2.	true	5,289

1 - 2 / 2



Now, let's split the data into training, validation and prediction data, in the ratio of 80-10-10.

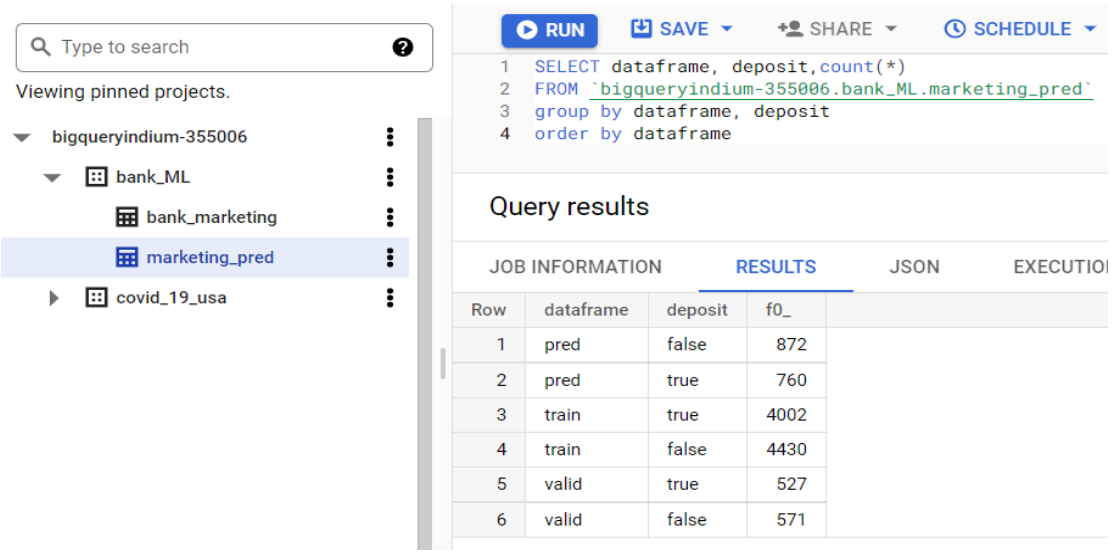
```
1  select age, job,marital, education,'default',balance, housing,
2  loan,contact,day,month,campaign,pdays,previous,poutcome,deposit,
3  CASE
4      WHEN split_field < 0.8 THEN 'train'
5      WHEN split_field = 0.8 THEN 'valid'
6      WHEN split_field > 0.8 THEN 'pred'
7    end as dataframe
8  from (
9  select age, job,marital, education,'default',balance, housing,
10 loan,contact,day,month,campaign,pdays,previous,poutcome,deposit,
11 ROUND(ABS(RAND()),1) as split_field
12 FROM `bigqueryindium-355006.bank_ML.bank_marketing` )
13
```

The inner clause (line 9-12) helps in random sampling by assigning random values within 0 to1 to split_field. CASE helps in splitting the data into 3 groups.

job	marital	education	f0_	balance	housing	loan	contact	day	month	campaign	pdays	previous	poutcome	deposit	dataframe
admin.	single	secondary	default	274	false	false	cellular	1	apr	3	-1	0	unknown	true	train
admin.	married	secondary	default	1474	false	false	cellular	2	apr	2	-1	0	unknown	true	train
admin.	single	secondary	default	471	false	true	cellular	2	apr	3	-1	0	unknown	true	train
admin.	divorced	secondary	default	4099	false	false	cellular	2	apr	2	-1	0	unknown	true	valid
admin.	married	secondary	default	1207	true	false	telephone	6	apr	2	138	2	success	true	train
admin.	single	secondary	default	82	false	true	cellular	7	apr	2	-1	0	unknown	true	pred
admin.	divorced	secondary	default	12039	false	false	telephone	8	apr	1	-1	0	unknown	true	valid

Let's copy this to a new table **marketing_pred** to prevent the change of values in the dataframe column .

Now, let's check the distribution into the 3 groups.

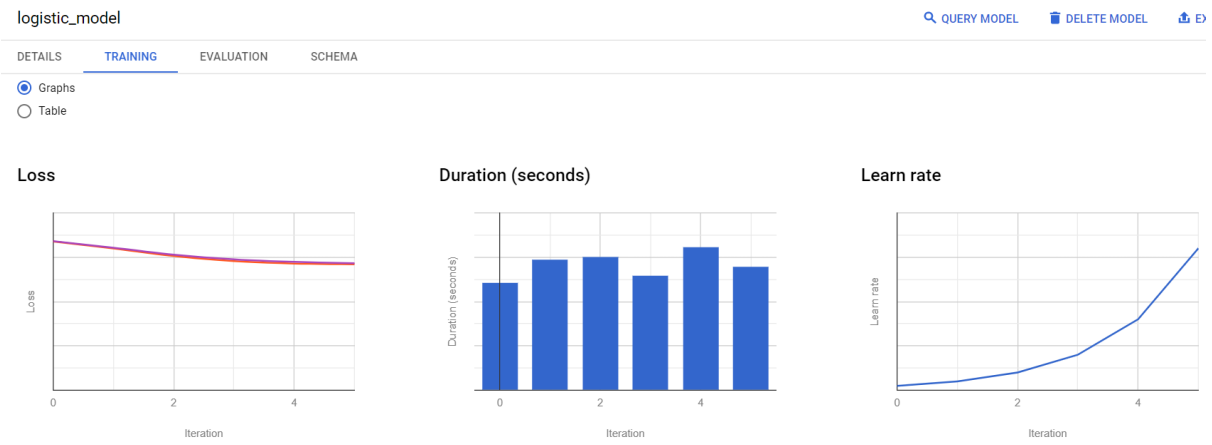


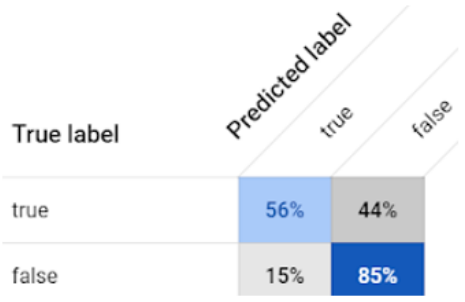
Now, let's train the model using logistic regression.

RUN SAVE SHARE SCHEDULE MORE

```
CREATE or REPLACE MODEL
`bank_ML.logistic_model`
OPTIONS
  ( model_type="logistic_reg",
    input_label_cols=['deposit'] ) AS
SELECT* EXCEPT (dataframe)
FROM `bigqueryindium-355006.bank_ML.marketing_pred`
WHERE dataframe = 'train'
```

Output we get as,





Below query gives the weights/coefficients of each feature in the model.

```
SELECT *
FROM ML.WEIGHTS(MODEL `bigqueryindium-355006.bank_ML.logistic_model` )
```

Query results

JOB INFORMATION | RESULTS | JSON | EXECUTION DETAILS

Row	processed_input	weight	catego....category	catego....weight
1	age	0.00096588255658649016	*null*	*null*
2	job	*null*	blue-collar	-0.11093851040324444
			retired	0.58413191977710488
			management	-0.12791035822067576
			services	-0.067945921343715632
			entrepreneur	-0.18184123491478263
			technician	-0.039505260047204144
			housemaid	-0.56428551280472339
			unknown	-0.014622892775512451
			admin.	0.0027200460901776577

Below query gives the details about each feature.

```
SELECT *
FROM ML.FEATURE_INFO(MODEL `bigqueryindium-355006.bank_ML.logistic_model` )
```

Query results

SAVE R

JOB INFORMATION | RESULTS | JSON | EXECUTION DETAILS

Row	input	min	max	mean	median	stddev	category_count	null_count	dimension
1	age	18.0	95.0	41.31764357608057	38.0	11.941879586814858	*null*	0	*null*
2	job	*null*	*null*	*null*	*null*	*null*	12	0	*null*
3	marital	*null*	*null*	*null*	*null*	*null*	3	0	*null*
4	education	*null*	*null*	*null*	*null*	*null*	4	0	*null*
5	derog	*null*	*null*	*null*	*null*	*null*	1	0	*null*
6	balance	-6847.0	81204.0	1500.466844286557	555.0	3115.4530967678393	*null*	0	*null*
7	housing	*null*	*null*	*null*	*null*	*null*	2	0	*null*
8	loan	*null*	*null*	*null*	*null*	*null*	2	0	*null*
9	contact	*null*	*null*	*null*	*null*	*null*	3	0	*null*
10	day	1.0	31.0	15.587773830669061	15.0	8.45900945005697	*null*	0	*null*

On predicting values for valid dataset,

RUN | SAVE | SHARE | SCHEDULE | MORE

```
SELECT *
FROM ML.EVALUATE(MODEL `bigqueryindium-355006.bank_ML.logistic_model`,
                 (SELECT*
                 FROM `bigqueryindium-355006.bank_ML.marketing_pred`
                 WHERE dataframe='valid'
                 )
)
```

Query results

SAVE RESULT

JOB INFORMATION | RESULTS | JSON | EXECUTION DETAILS

Row	precision	recall	accuracy	f1_score	log_loss	roc_auc
1	0.73366834170854267	0.5540796963946869	0.68943533697632053	0.63135135135135134	0.60041486294969	0.74416383616383619

Note that we want to know customers who will be interested in deposit, so we have to improve our recall.

Now, let's do some feature engineering so as to improve recall using the **TRANSFORM** function.

RUN SAVE SHARE SCHEDULE MORE

```
1  CREATE OR REPLACE MODEL `bigqueryindium-355006.bank_ML.logistic_model_2`
2  TRANSFORM(
3    ML.QUANTILE_BUCKETIZE(age,5) OVER() AS bucketized_age,
4    ML.FEATURE_CROSS(STRUCT(job,education)) job_education,
5    marital,balance,housing,loan,contact,day,month,pdays,previous,poutcome,deposit)
6  OPTIONS(
7    model_type='LOGISTIC_REG',
8    input_label_cols=['deposit']
9  ) AS
10 SELECT * EXCEPT(dataframe,campaign, derog)
11 FROM `bigqueryindium-355006.bank_ML.marketing_pred`
12 WHERE dataframe='train'
```

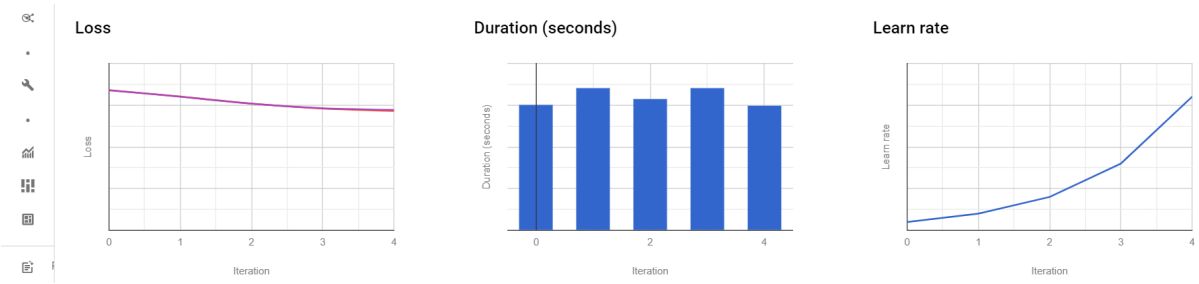
In line 3, 'age', which is a continuous numerical feature, is bucketized into a STRING based on quantiles. Here, we are making it into 5 buckets.

The 'job' and 'education' columns which have categorical values are kind of related, so we can do a feature cross on them.

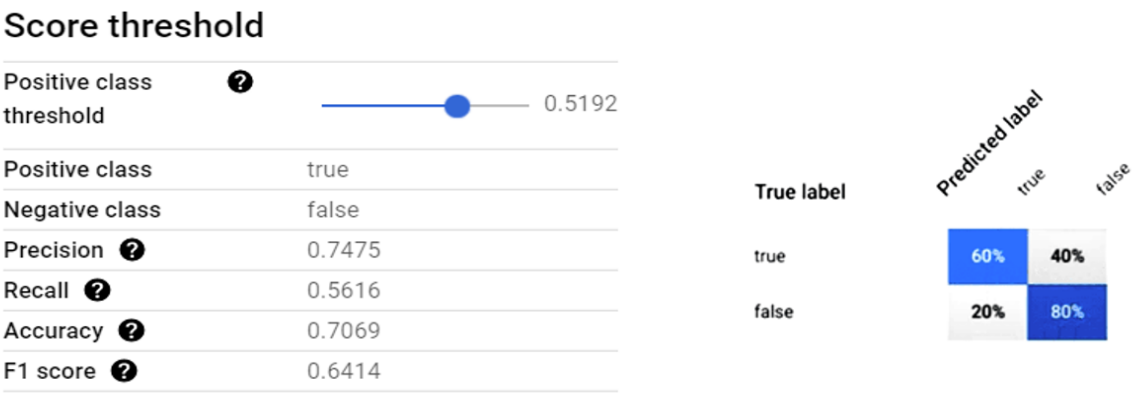
Note that we have also dropped campaign and derog columns.

We'll now apply the logistic regression model on the train part of the dataset.

Now, click on '**go to model**' and let's check out its details.



It can be noticed that the number of iterations have reduced.



We can see that the true positive has increased thus causing a higher recall.

```
1  SELECT *
2  FROM ML.WEIGHTS(MODEL `bigqueryindium-355006.bank_ML.logistic_model_2`)
```

Query results

JOB INFORMATION | RESULTS | JSON | EXECUTION DETAILS

Row	processed_input	weight	catego....category	catego....weight
1	bucketized_age	null	bin_2	-0.079846096895319491
			bin_4	-0.12447591816604348
			bin_5	0.059261741101308038
			bin_1	0.20544639114220695
			bin_3	-0.087251293314640072
2	job_education_job_education	null	services_secondary	-0.11958930699687889
			services_tertiary	0.2178799635747094
			self-employed_tertiary	0.32073672221621924
			retired_unknown	0.34745278405240387

We can now see the changed weights/coefficients of each feature in the new model.

Arrays and Struct in Google BigQuery

ARRAYS

What are arrays?

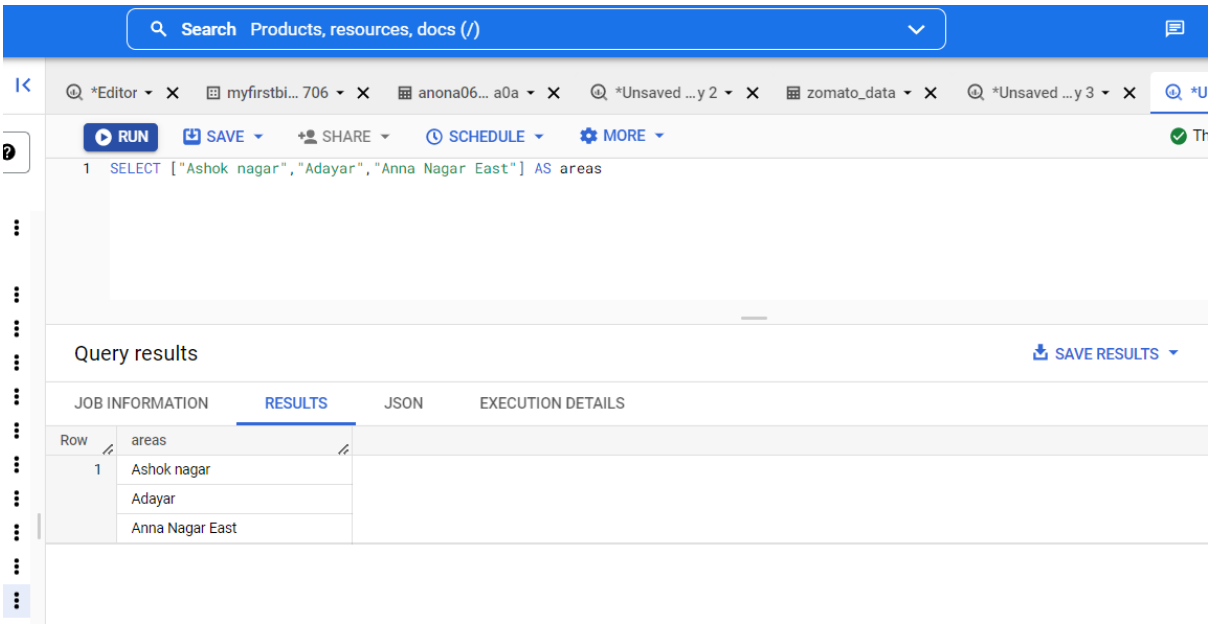
Arrays are basically an ordered list with zero or more values of the same data type. It is used when we want to store repeated values in a single row. They can be very powerful once we learn how to use them. For example, they can be used to:

- Transforming non-normalized data
- Advanced string manipulation
- Optimising storage and performance, etc.,

FUNCTIONS OF ARRAY

There are several features of array:

- Array_Length
- Array_concat
- Array_generate
- Generate_date_array
- Array_reverse

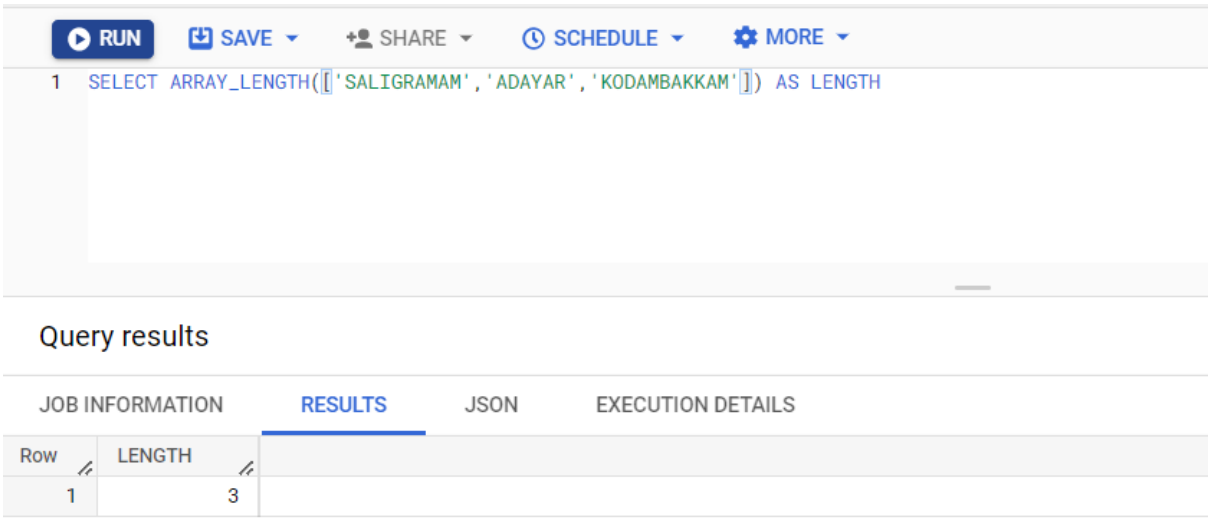


A simple example of an array

Arrays Length in BigQuery:

As the name suggests, an array length tells us the length of an array. This may come in handy when we want to know the length of an array while indexing.

We can use the array_length to do this.

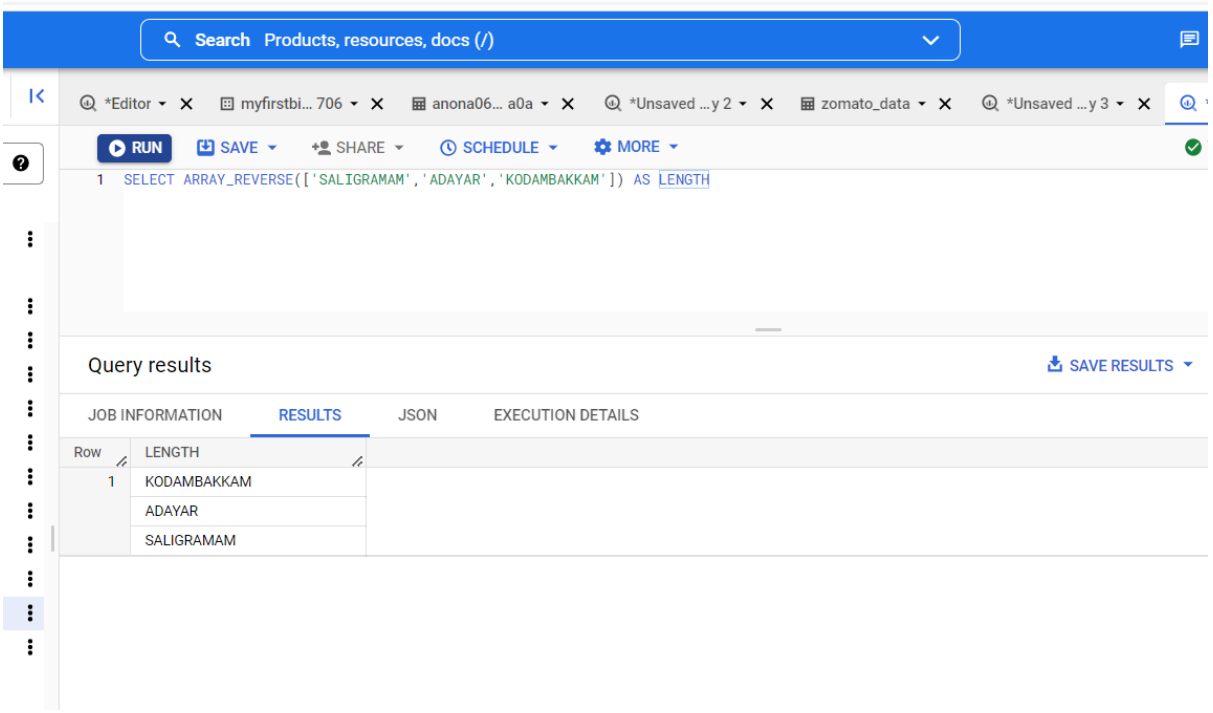


A simple example of an array length

Arrays Reverse in BigQuery:

The order of array elements can be reversed. If we want to access the elements at the bottom of an array, we can use this function.

We can use the array_reverse to do this.

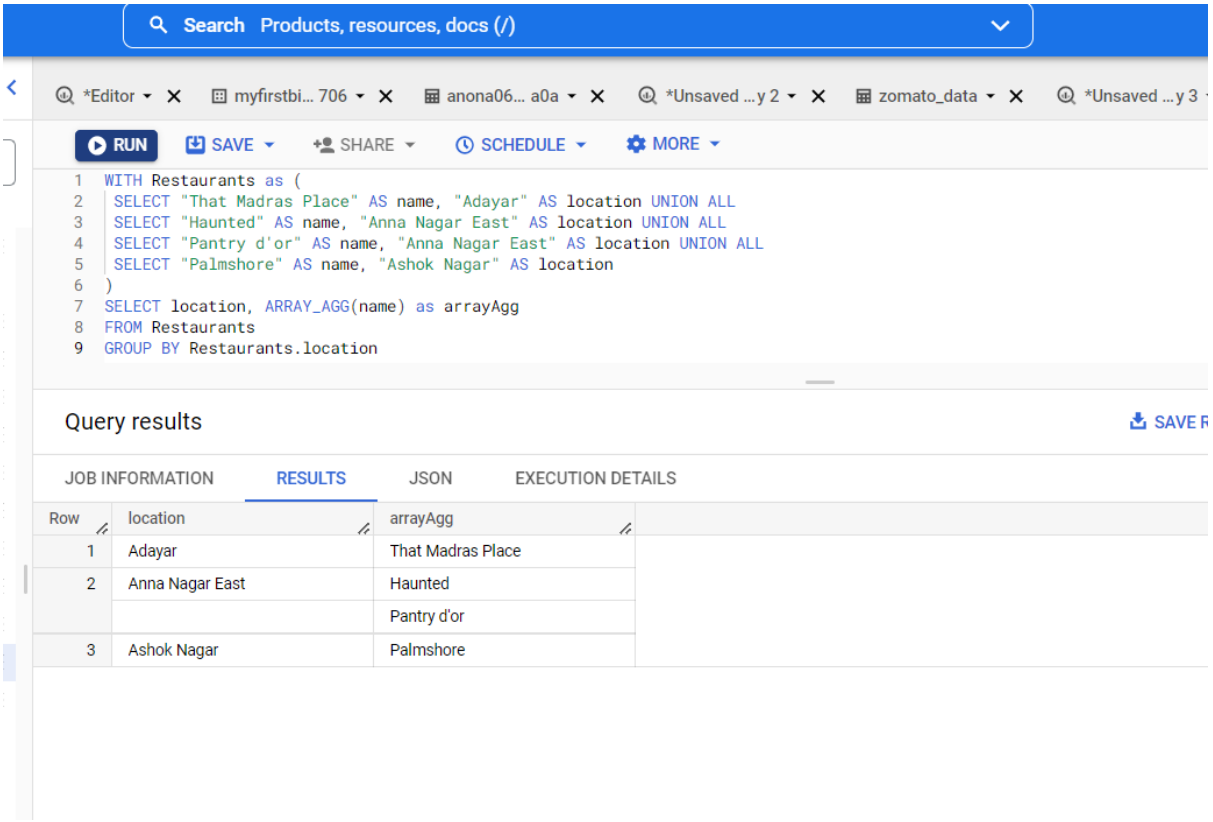


A simple example of an array reverse

Array Aggregate in BigQuery:

This functions **returns an ARRAY of expression values**. It is basically the opposite of UNNEST (used to flatten an array into its components to make it usable for analysis and database operations).

We use the array_agg to do this.

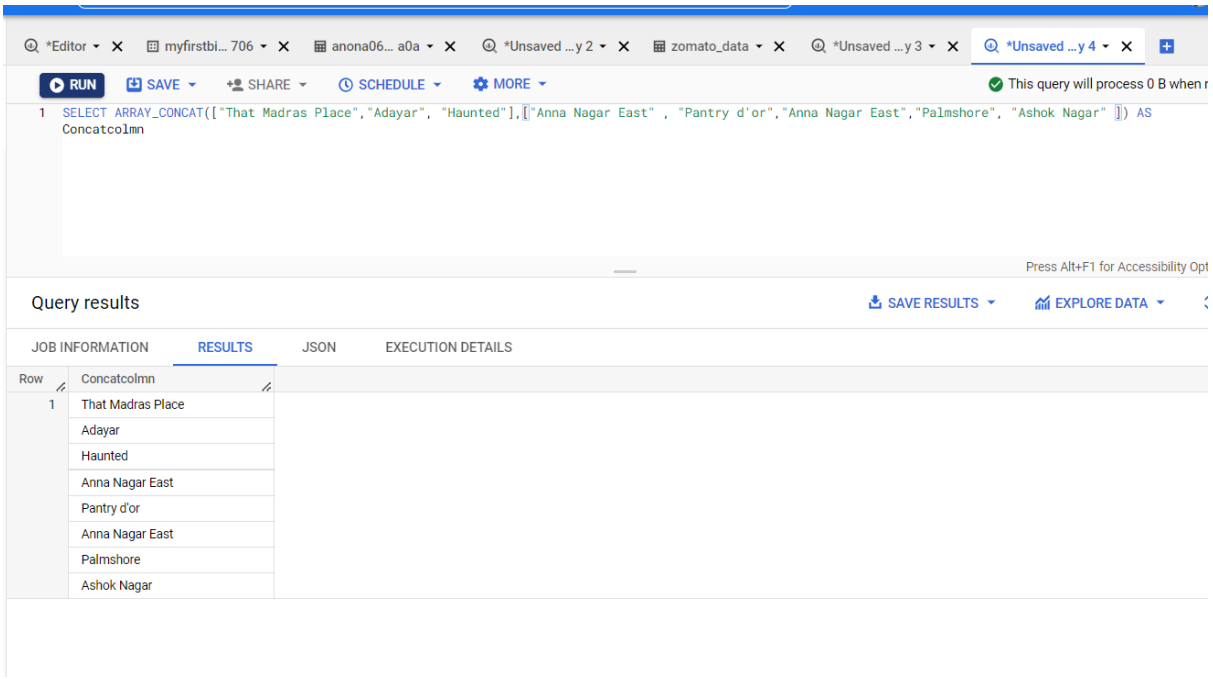


A simple example of an array_agg

Array Concat in BigQuery:

This function concatenates one or more arrays with the same element type into a single array.

We use the array_concat to do this.



A simple example of an array_concat

Array Generate in BigQuery:

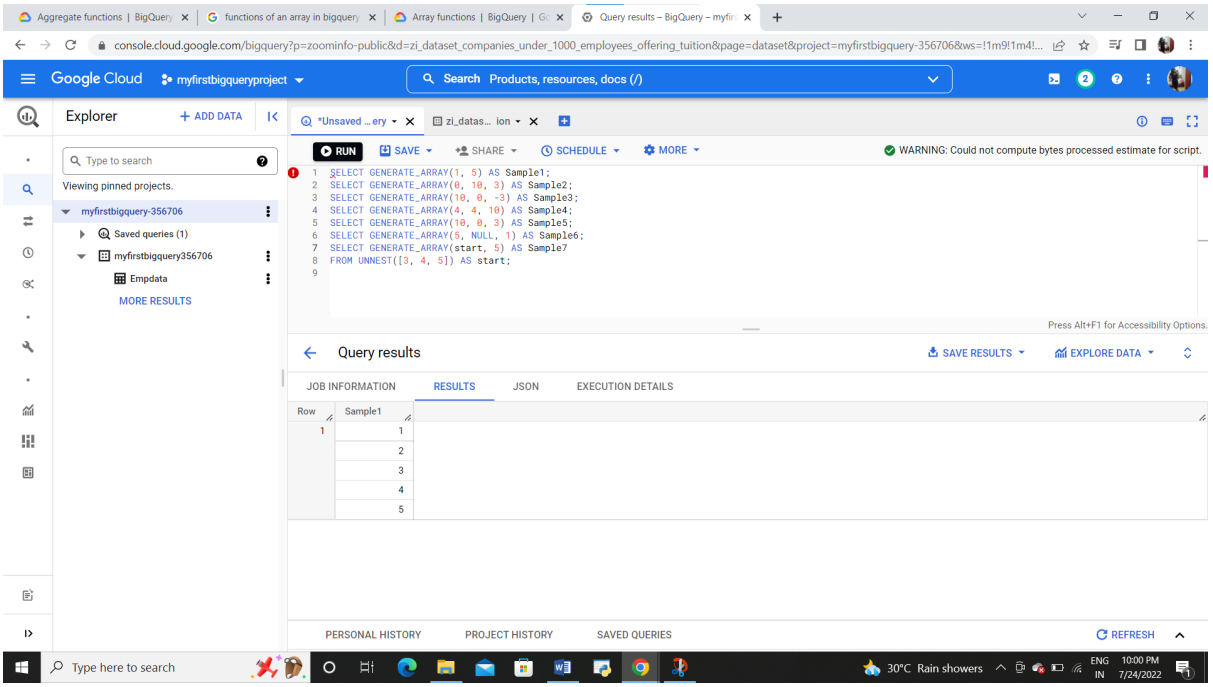
This returns an array of values. We need to define the start and end expressions in order to implement this function. We also need to define a step expression in order to determine the increment value.

We use the array_generate to do this.

Now let us take a look at a few examples:

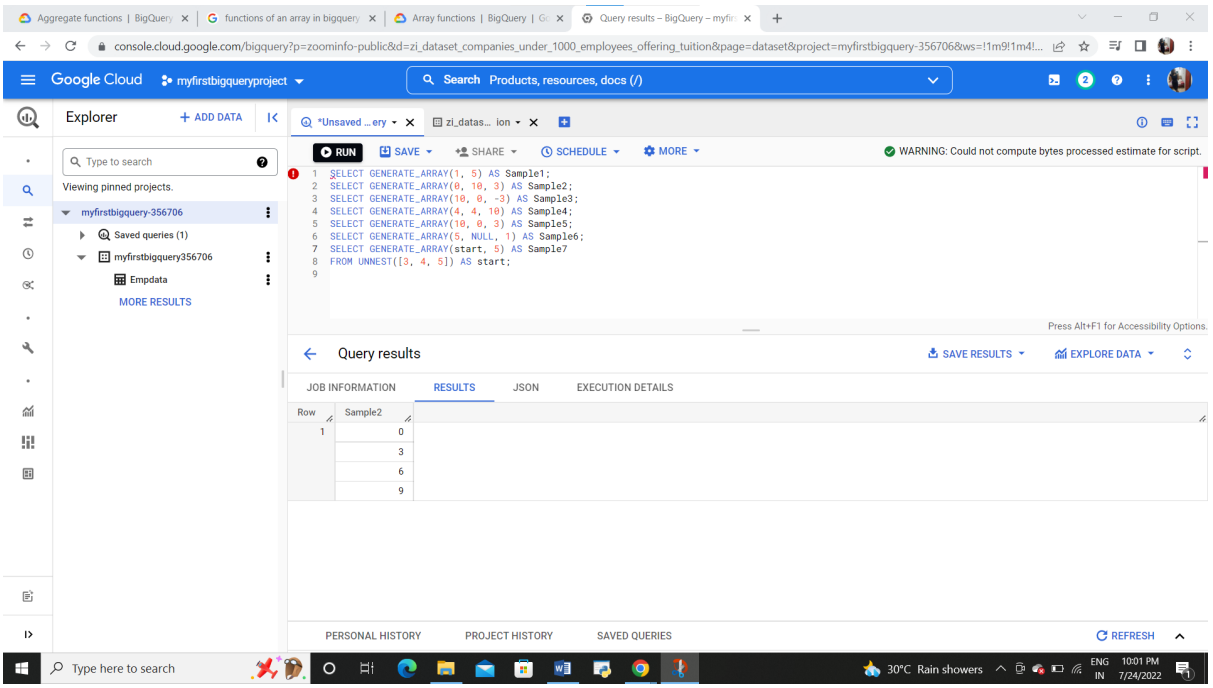
1.The defined start and end expression for sample 1 is 1 and 5 respectively.

Hence, it will return the value up to 5 with a default step of 1.



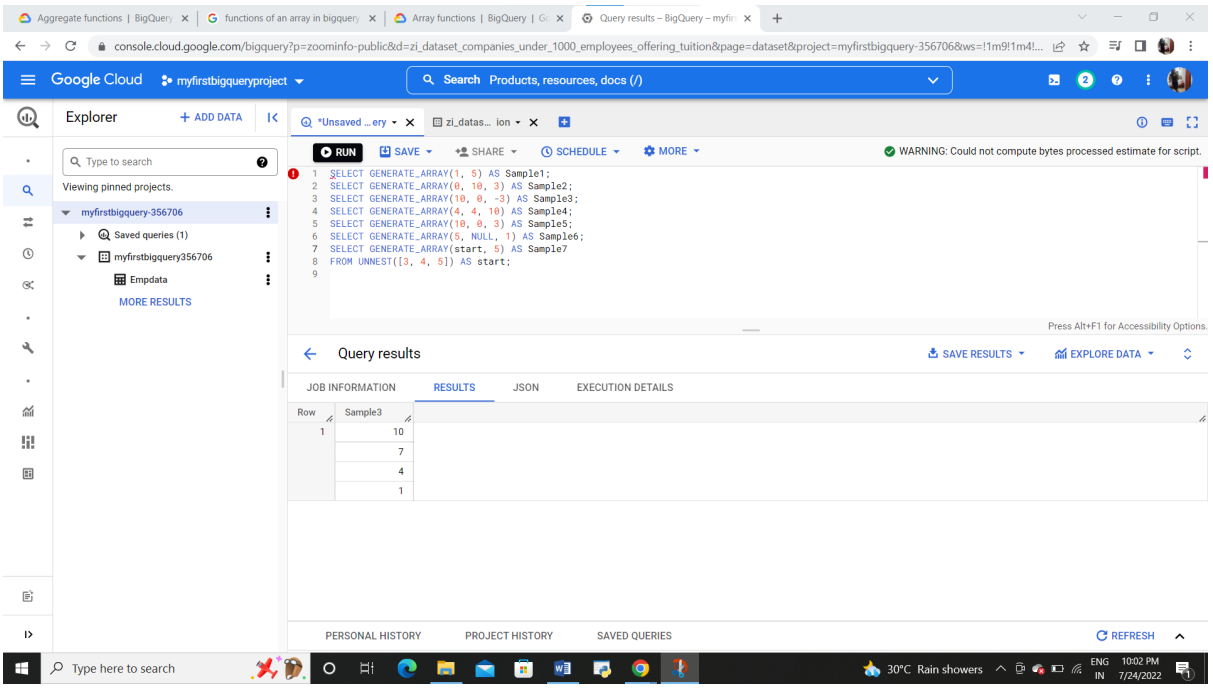
A simple example of an array_Generate

2.In this sample, I have mentioned the start expression as 0 and end expression as 10 with a step expression as 3. Hence the values will increase by 3 until it reaches the end expression value.



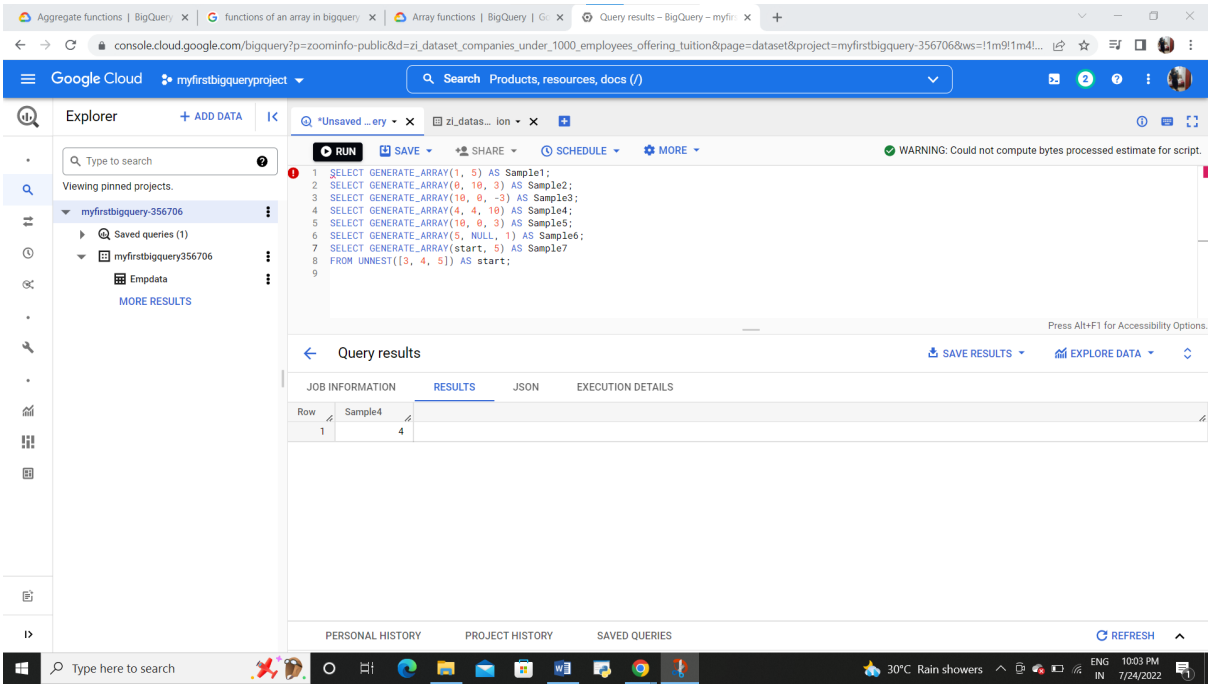
A simple example of an array_Generate

3.In this sample, I have mentioned the step value as -3, which would mean the start expression would decrease in value until it reaches the end expression.



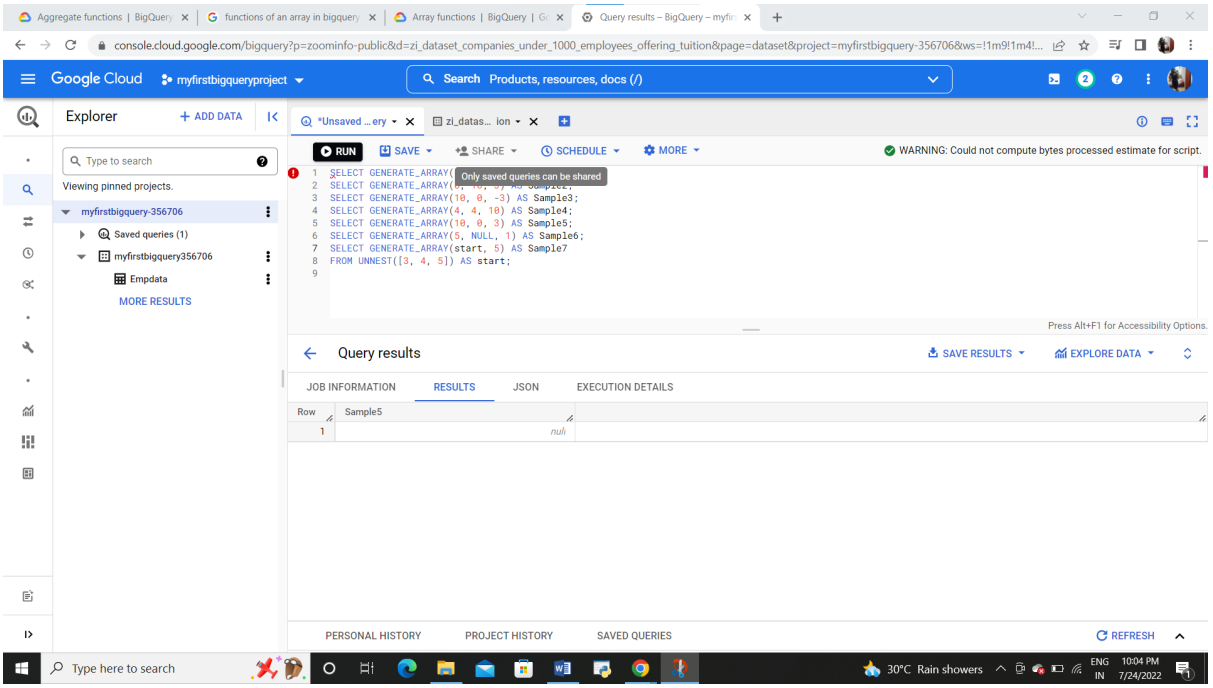
A simple example of an array_Generate

4. The following returns an array using the same value for the start and end expression.



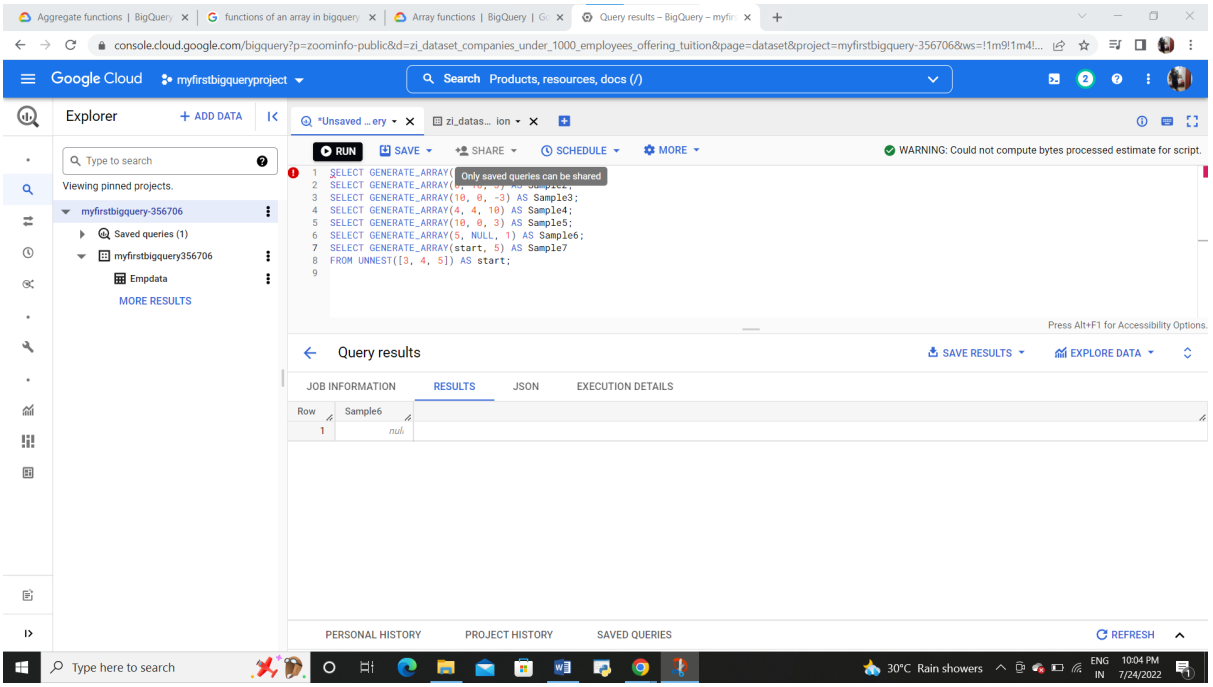
A simple example of an array_Generate

5.The following returns an empty array as the start expression is higher than the end expression. And we cannot decrease the value either because the step expression is a positive Integer.



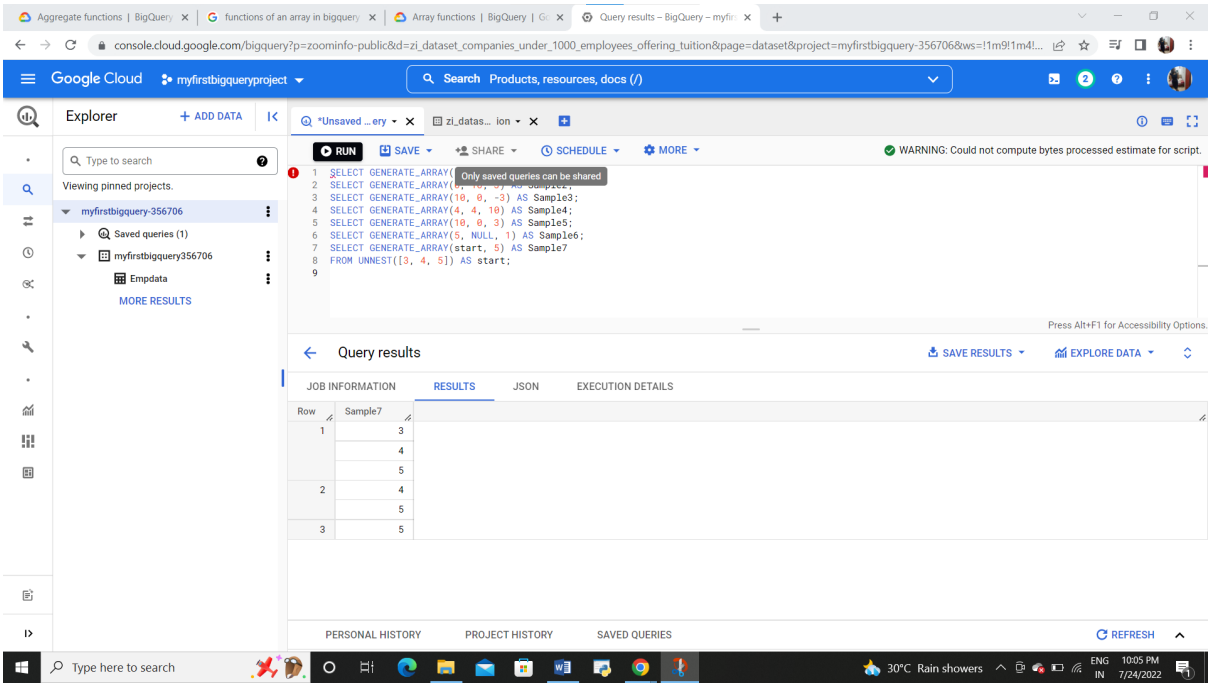
A simple example of an array_Generate

6. The following sample returns a null value because the end expression is a null value.



A simple example of an array_Generate

7. The following sample returns multiple.

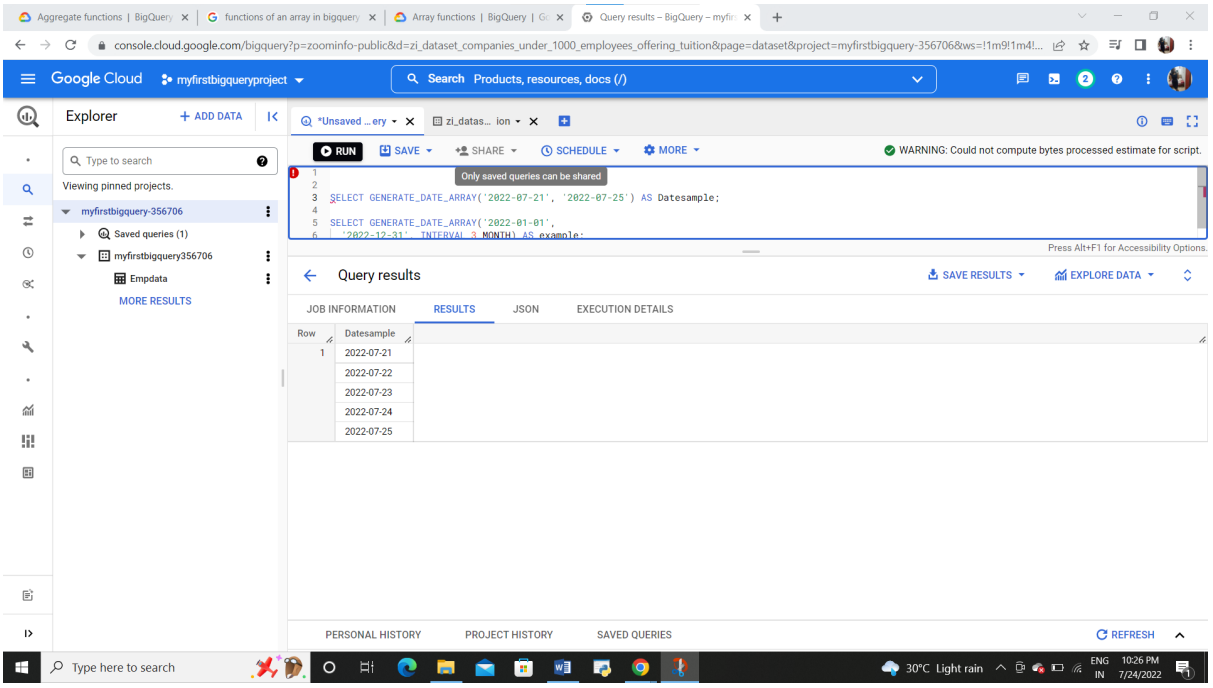


A simple example of an array_Generate

Generate_Date_Array in BigQuery:

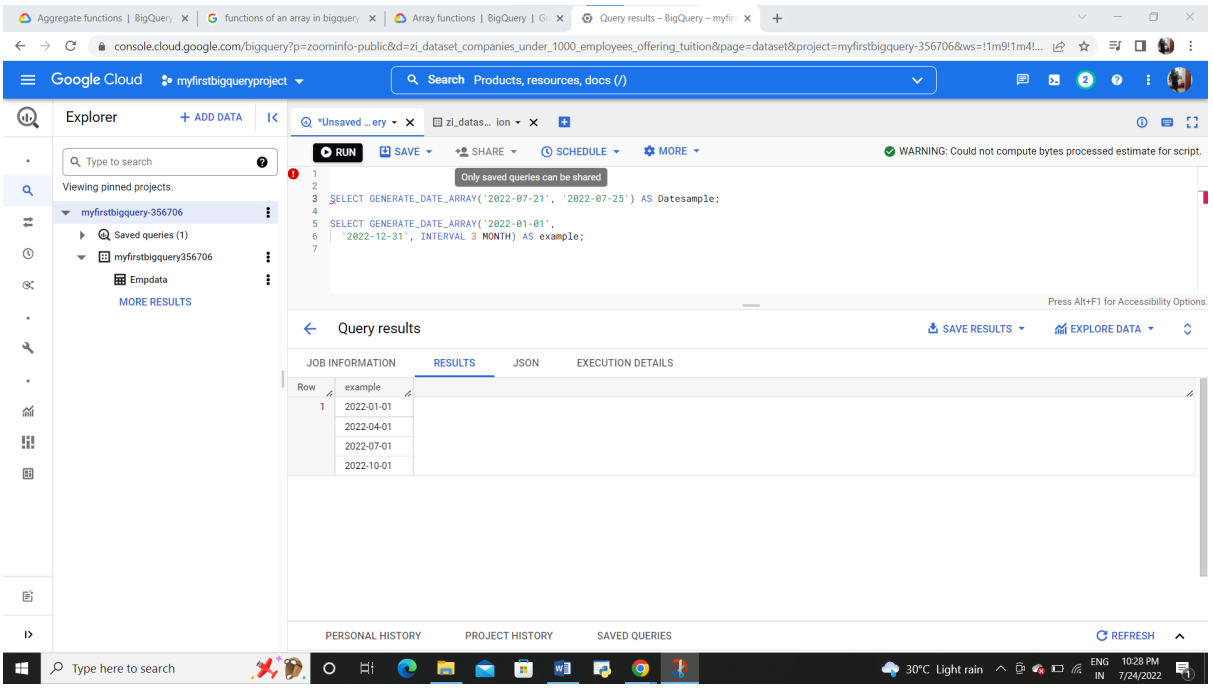
This function returns an array of dates. Just like the generate function, we need a start and end expression for generating dates.

1. The following example returns the dates with an interval of 1.



A simple example of an Generate_date_array

2. Let us find out the dates for a year with 3 month intervals.



A simple example of an Generate_date_array

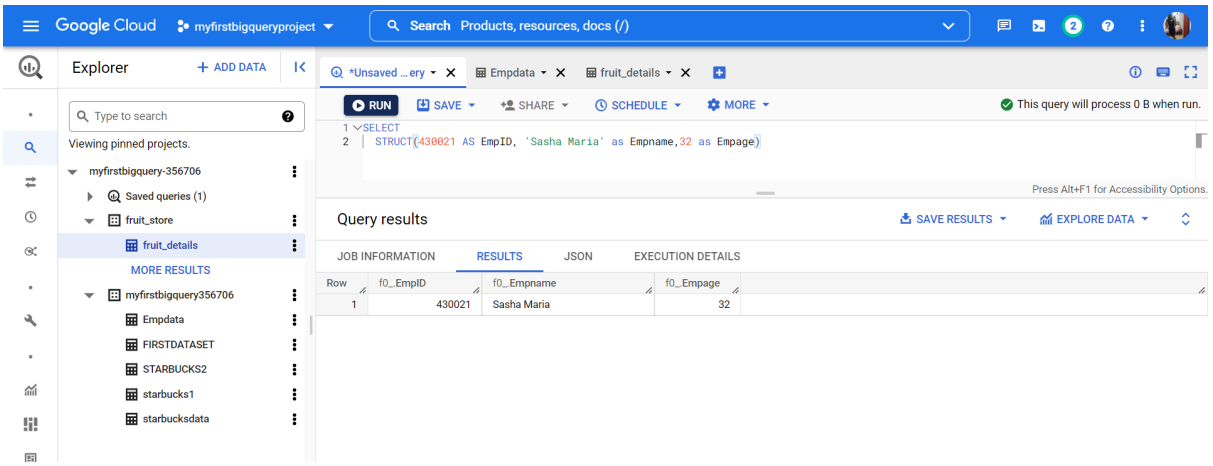
STRUCTS

Structs are flexible containers of ordered fields each with a type (required) and a name (optional) – From Google cloud.

Unlike arrays, you can store multiple data types in structs and by that even arrays can be stored in structs. To put it simply, the struct is the root or the parent column with multiple child columns. For example, an employee has a qualification associated with different fields such as university, degree, start date and end date.

We use the keyword STRUCT to create the table.

Let's take a look at a simple example below:

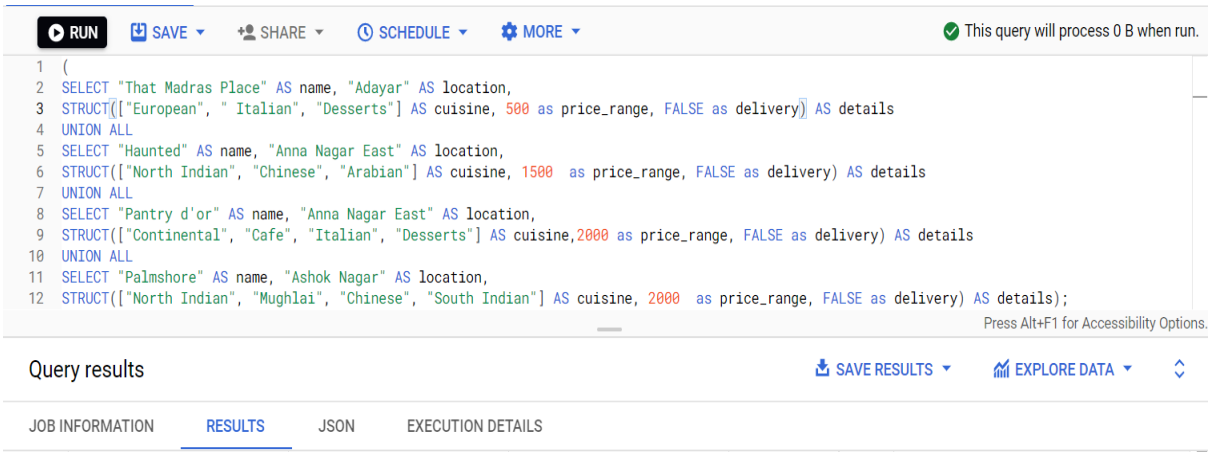


A simple example of struct

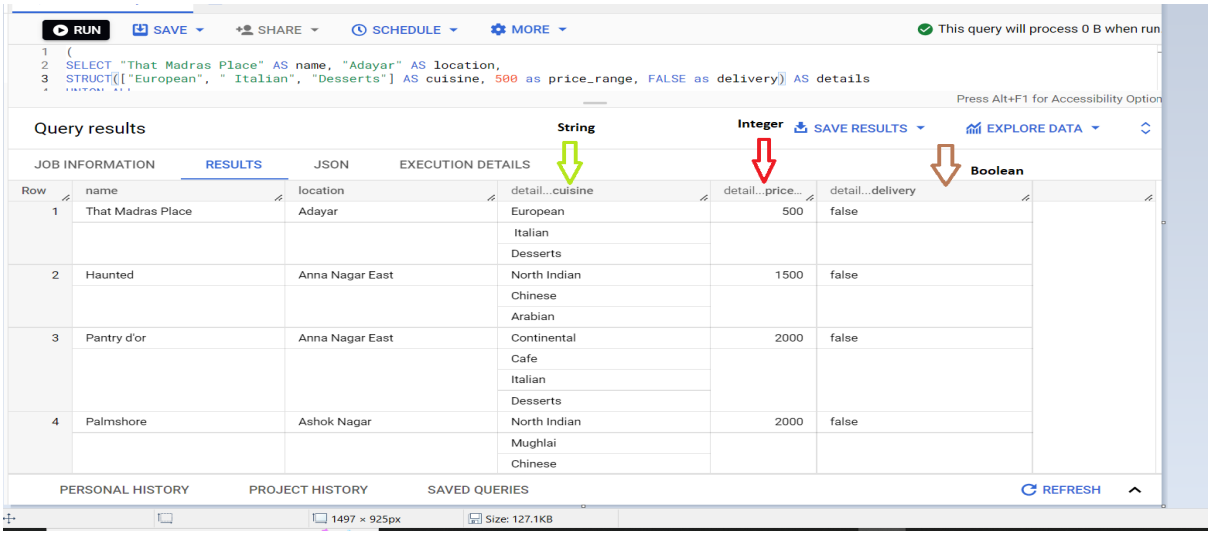
The best way to understand structs is to create one. Let us look at the restaurant database from Zomato. (Source: Kaggle)

The database has details about the restaurants location, address, rating, cuisines available, table booking, online delivery, etc., etc.,

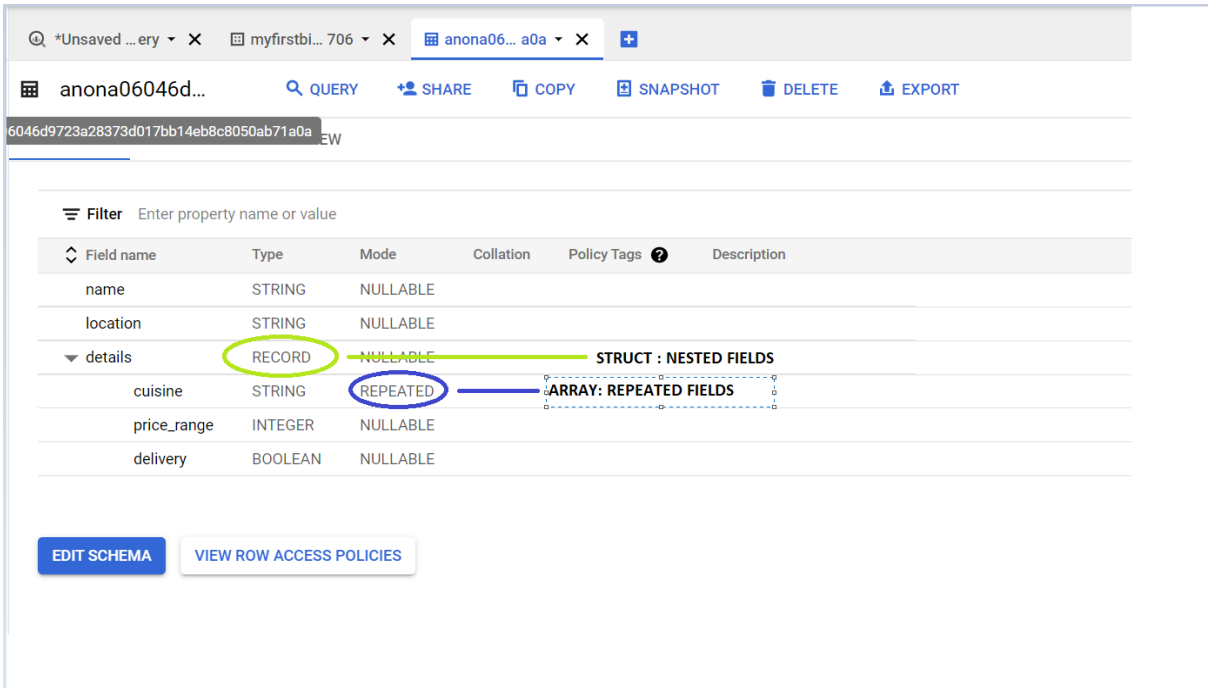
I have filtered out the restaurants in Chennai and selected a few.



Sample query for Struct creation



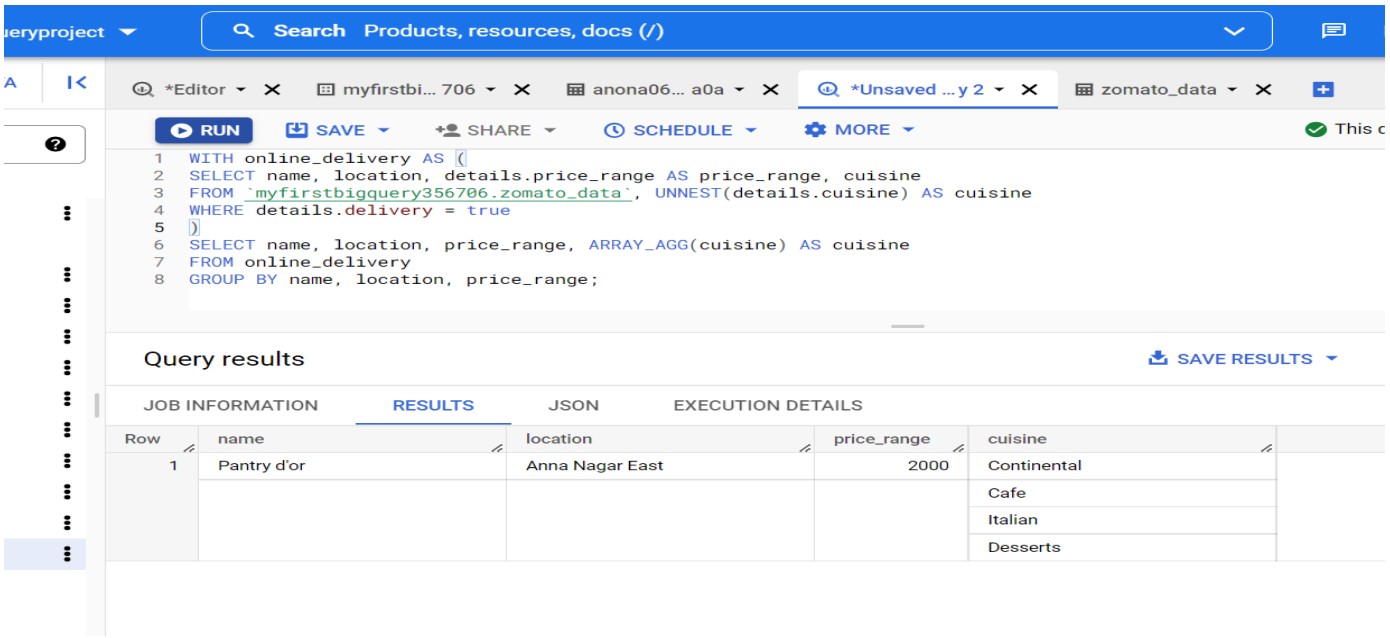
Output showing struct containing different data types.



Let us check the schema of the table we just created. If the mode is record (which is nested fields) it means it is a Struct. But if the mode is Record/repeated fields, then it is an array.

Schema of the zomato_data table

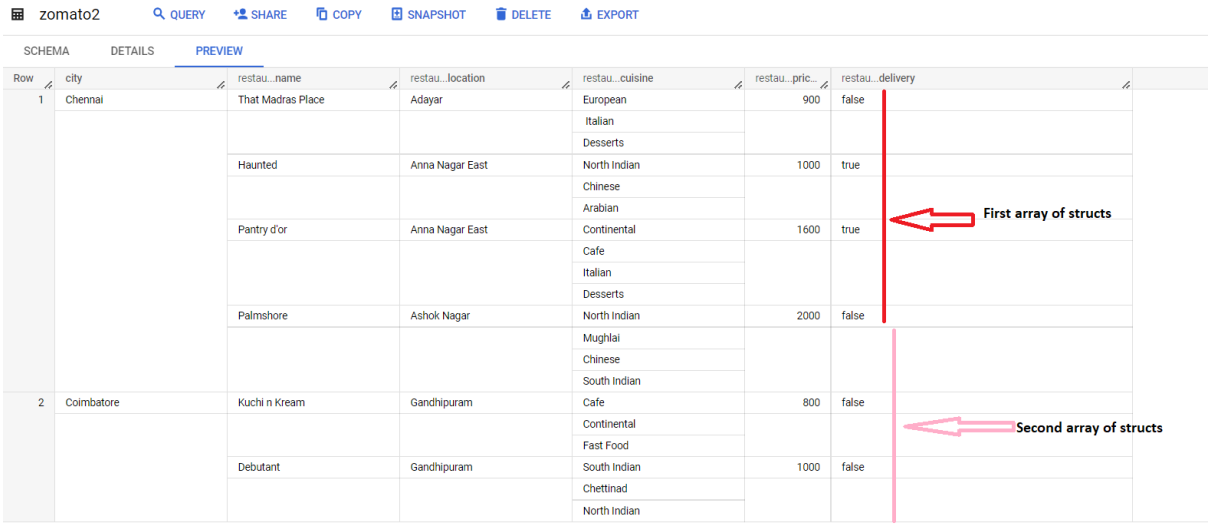
Now, in the given database, I want to see which restaurant accepts online delivery. This is where the dot notation comes in handy. (WHERE = details.delivery = true). This will return all the restaurants that have online delivery.Below is the code for the dot notation.



Struct with Dot notation.

NESTED RECORDS:

We have created nested records that denormalized multiple levels without joins. As we can see in the picture below, all records exist in one table. Each record is nested to save storage but we can use the UNNEST function for expansion if needed. Maintaining nested records will eliminate the need for repeating data, using join or creating subtables.



Row	city	restau...name	restau...location	restau...cuisine	restau...pric...	restau...delivery
1	Chennai	That Madras Place	Adayar	European	900	false
				Italian		
				Desserts		
		Haunted	Anna Nagar East	North Indian	1000	true
				Chinese		
				Arabian		
		Pantry d'or	Anna Nagar East	Continental	1600	true
				Cafe		
				Italian		
				Desserts		
		Palmshore	Ashok Nagar	North Indian	2000	false
				Mughlai		
				Chinese		
				South Indian		
2	Coimbatore	Kuchi n Kream	Gandhipuram	Cafe	800	false
				Continental		
				Fast Food		
		Debutant	Gandhipuram	South Indian	1000	false
				Chettinad		
				North Indian		

Nested records.

Query for nested records.

We have looked at arrays and structs in BigQuery. We've explored the definitions and examples. We can sum up these use cases under two import points.

- Arrays support elements of the same data types.
- Structs support elements of different data types which also include arrays.Nested records in Bigquery are structs that help us save storage and eliminates the risk of repeated data. This will enhance the performance in return and also reduce the storage cost.
- Nested records in Bigquery are structs that helps us save storage and eliminates the risk of repeated data. This will enhance the performance in return and also reduces the storage cost

BigQuery Partitioning and Clustering

Partitioning

A partitioned table is a special table that is divided into segments, called partitions, that make it easier to manage and query your data. Dividing a large table into smaller partitions, improves query performance and control costs by reducing the number of bytes read by a query.

Partitioning is used under the following circumstances:

You want to know query costs before a query runs. Partition pruning is done before the query runs, so you can get the query cost after partitioning pruning through a dry run. Cluster pruning is done when the query runs, so the cost is known only after the query finishes.

You need partition-level management. For example, you want to set a partition expiration time, load data to a specific partition, or delete partitions.

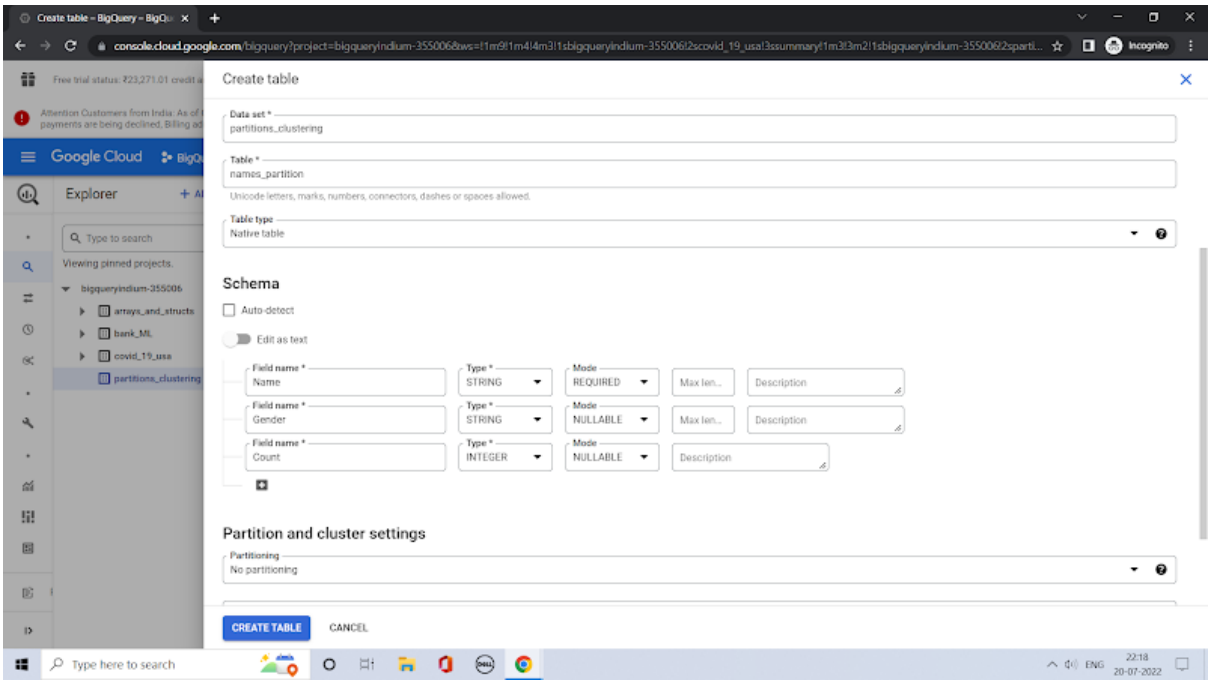
You want to specify how the data is partitioned and what data is in each partition. For example, you want to define time granularity or define the ranges used to partition the table for integer range partitioning.

We can partition BigQuery tables by 3 ways -:

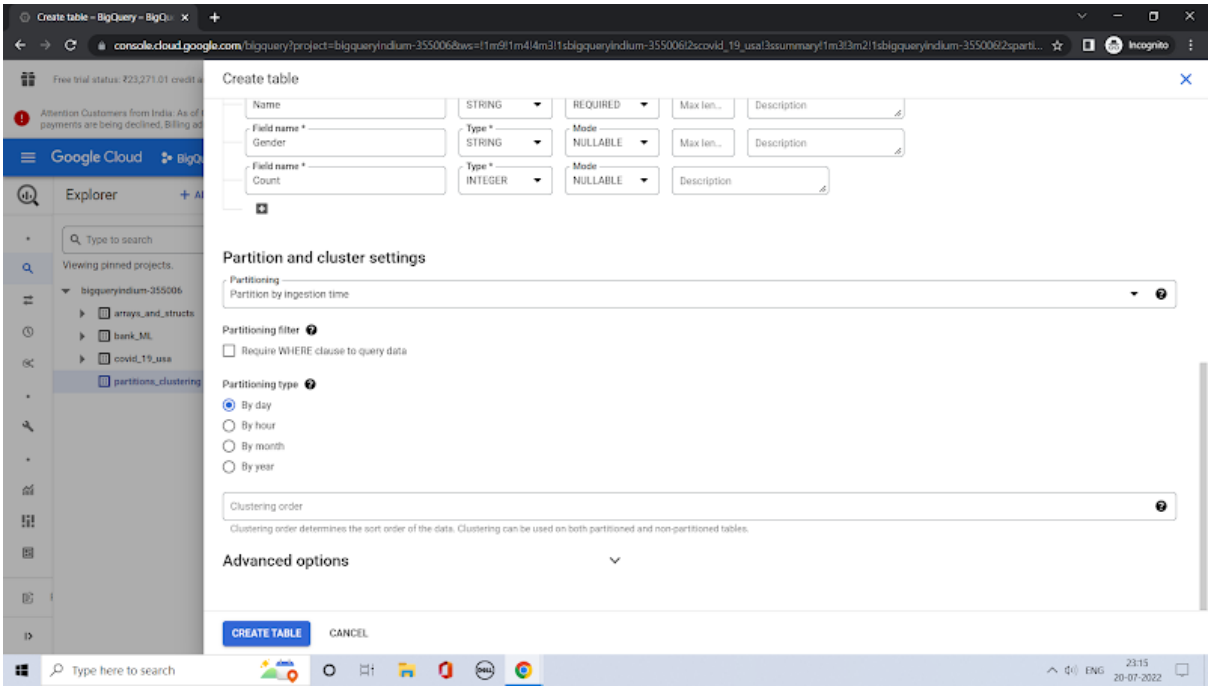
Time-unit column - Tables are partitioned based on a TIMESTAMP, DATE, or DATETIME column in the table.
Ingestion time - Tables are partitioned based on the timestamp when BigQuery ingests the data.
Integer range - Tables are partitioned based on an integer column.

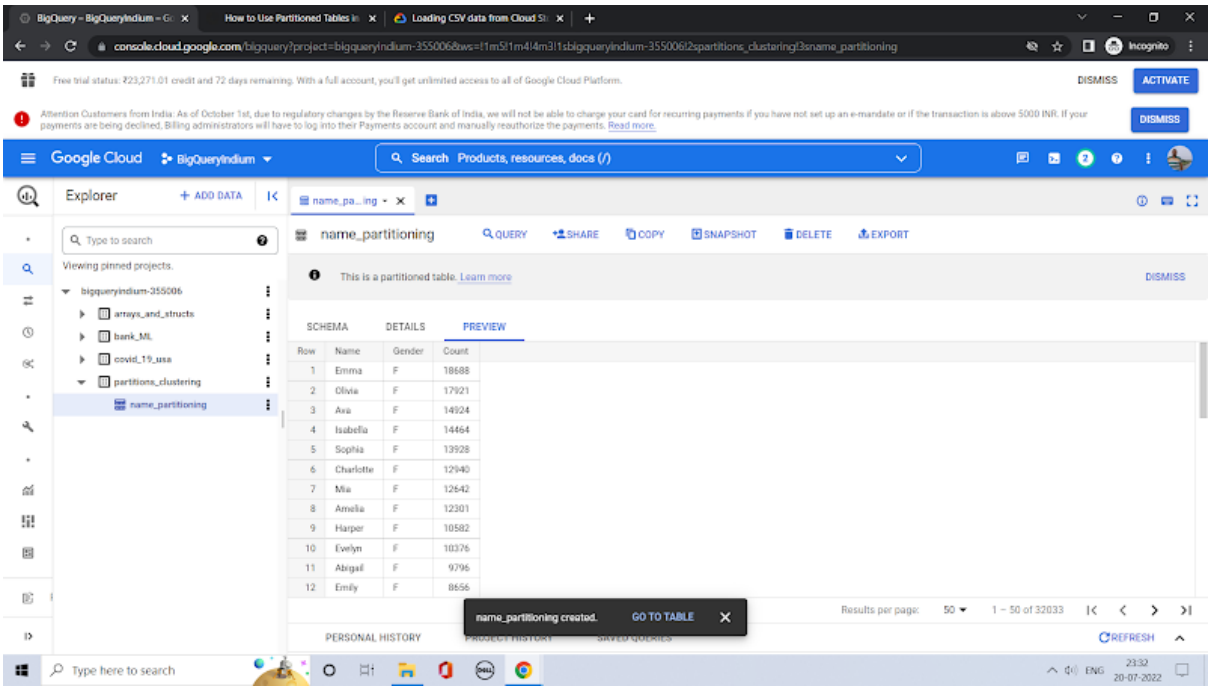
Now, let's understand this with an example.

Take the dataset called names. It has 3 fields - **Name, Gender and Count** (number of people with that particular name).



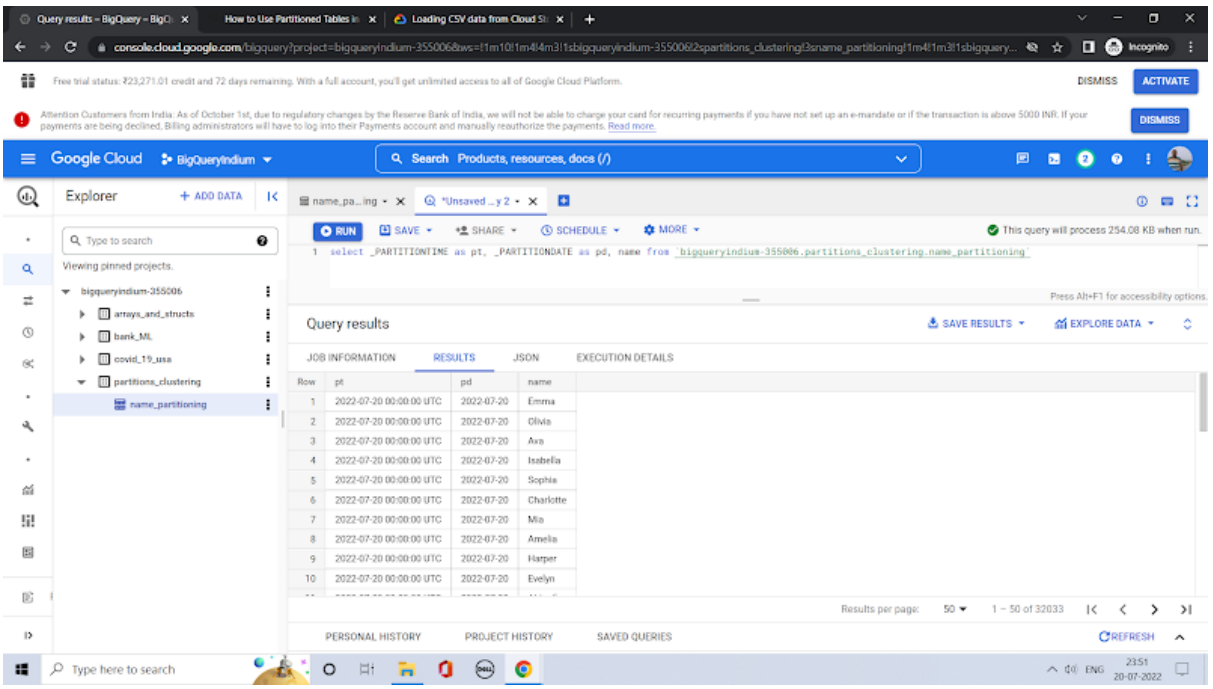
Now let's partition it based on ingestion time (here let's take partitioning type as day).





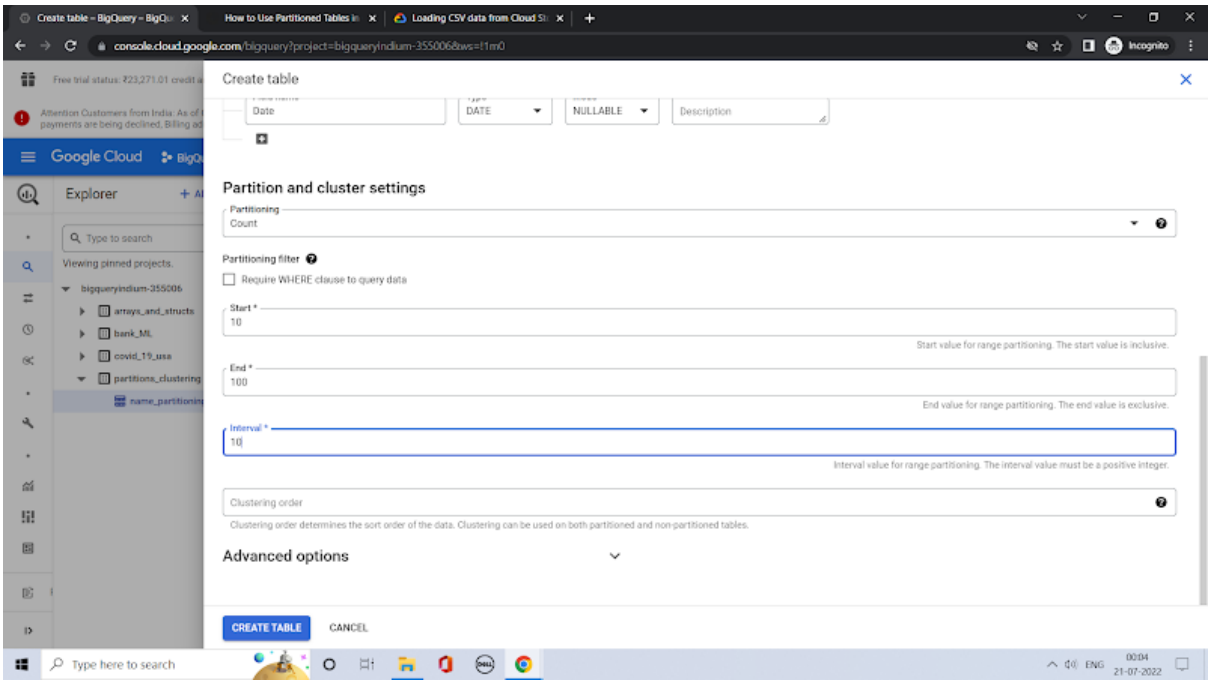
NOTE -: The tables can be differentiated as partitioned or not by looking at their symbols in front of their names.

Now, let's perform a query on the partitioned table **name_partitioning.**

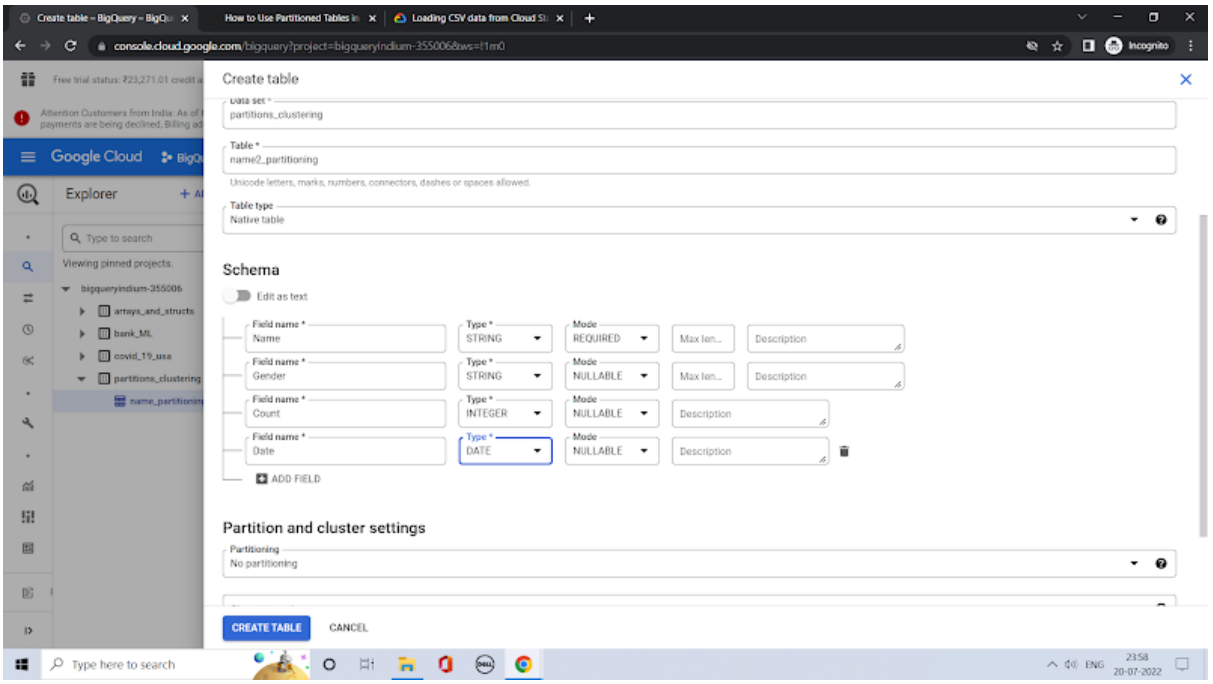


NOTE -: Whenever we query partition time and partition date we have to use aliases (in the above case, pt, pd).

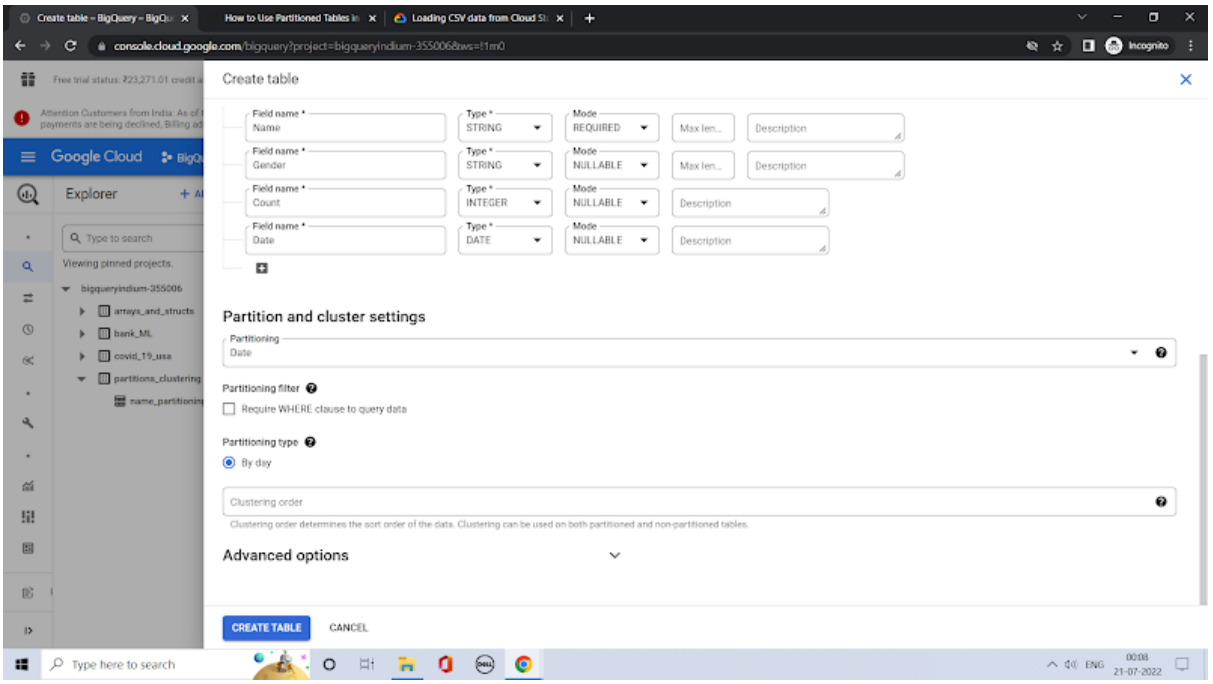
The above dataset can also be partitioned by its count.



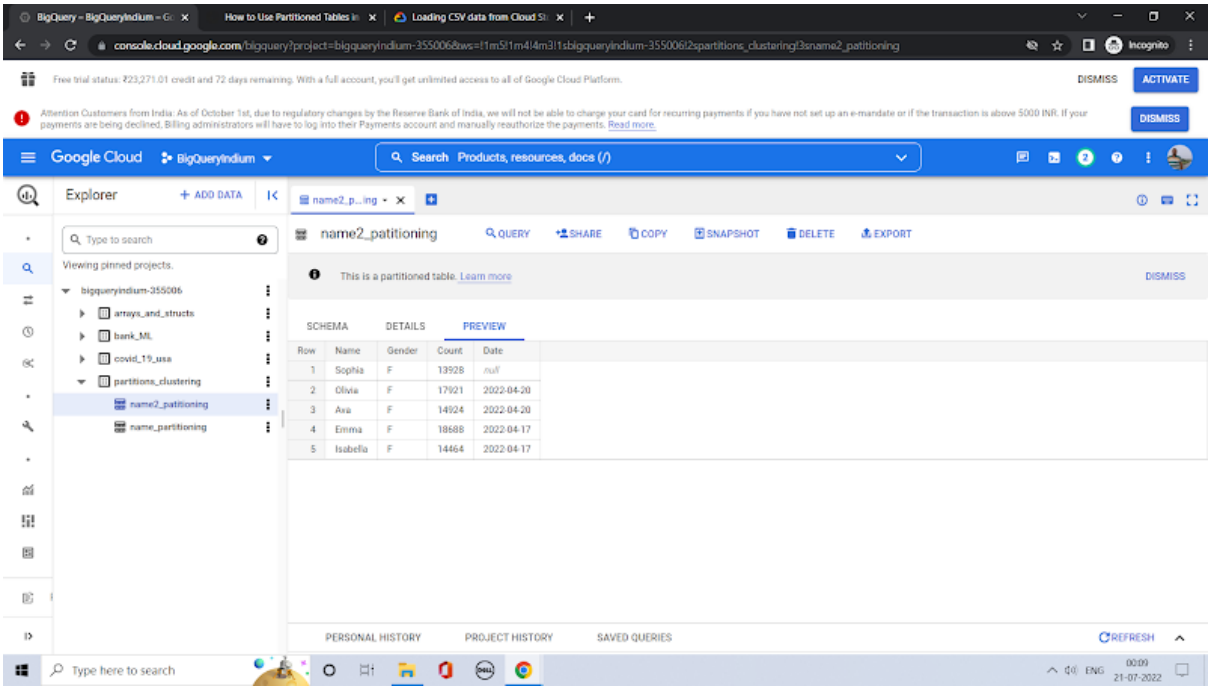
Now, let's take another dataset to understand partitioning wrt date. It has the following schema.



Now, we also have the option of partition by date.



Its preview.



NOTE -: In the above preview row 1 does not throw an error even though it does not have a Date value as it considers it as part of NULL partition.

CLUSTERING

Clustering is another way of organising data which stores one next to the other all those rows that share similar values in the chosen clustering columns. This process increases the query efficiency and performance. Note that BigQuery supports this feature only on partitioned tables.

Clustering should be used under the following circumstances -:

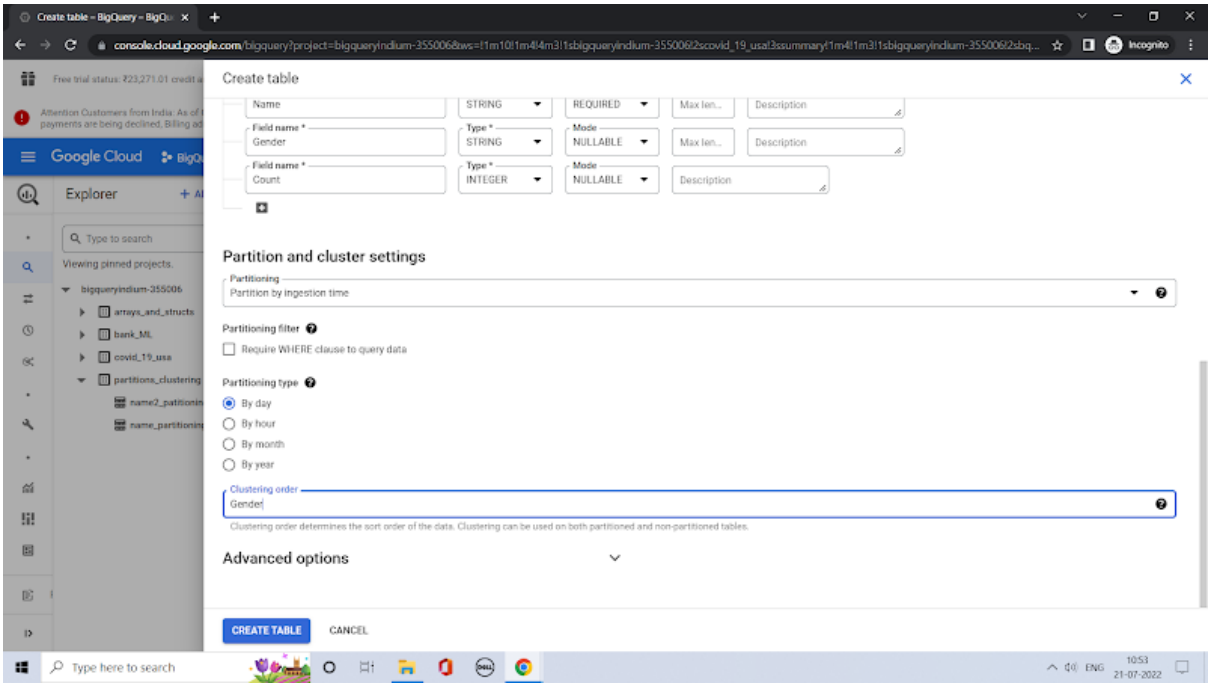
We don't need strict cost guarantees before running the query.

We need more granularity than partitioning alone allows. To get clustering benefits in addition to partitioning benefits, we can use the same column for both partitioning and clustering.

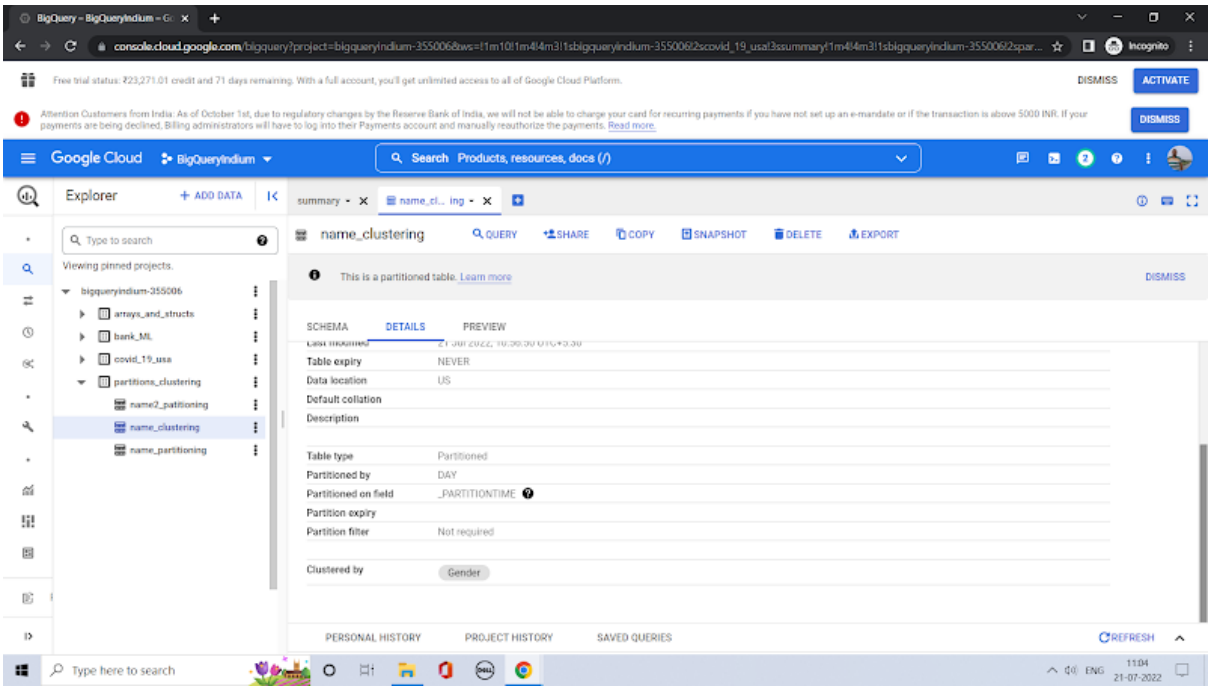
Queries commonly use filters or aggregation against multiple particular columns.

The cardinality of the number of values in a column or group of columns is large.

Now, let's take up the same names dataset for understanding clustering.



NOTE -: We can give upto 4 fields to define clustering order.



Now, any query that is searched wrt **gender** will be very fast compared to a regular query.

Its preview.