EX.NO 8: Store and Query massive Datasets using Google Cloud BigQuery

Date:

READING MATERIALS:

Google BigQuery is a serverless, highly scalable data warehouse that comes with a built-in query engine. The query engine is capable of running SQL queries on terabytes of data in a matter of seconds, and petabytes in only minutes. You get this performance without having to manage any infrastructure and without having to create or rebuild indexes.

Working with BigQuery

BigQuery is a data warehouse, implying a degree of centralization and ubiquity. The query we demonstrated in the previous section was applied to a single dataset. However, the benefits of BigQuery become even more apparent when we do joins of datasets from completely different sources or when we query against data that is stored outside BigQuery.

From an architectural perspective, BigQuery is fundamentally different from on-premises data warehouses like Teradata or Vertica as well as from cloud data warehouses like Redshift and Microsoft Azure Data Warehouse. BigQuery is the first data warehouse to be a scale-out solution, so the only limit on speed and scale is the amount of hardware in the datacenter.

Components that make BigQuery successful and unique are.

Separation of Compute and Storage

In many data warehouses, compute and storage reside together on the same physical hardware. This colocation means that in order to add more storage, you might need to add more compute power as well. Or to add more compute power, you'd also need to get additional storage capacity.

If everyone's data needs were similar, this wouldn't be a problem; there would be a consistent golden ratio of compute to storage that everyone would live by. But in practice, one or the other of the factors tends to be a limitation. Some data warehouses are limited by compute capacity, so they slow down at peak times. Other data warehouses are limited by storage capacity, so maintainers need to figure out what data to throw out.

When you separate compute from storage as BigQuery does, it means that you never need to throw out data, unless you no longer want it. This might not sound like a big deal, but having access

to full-fidelity data is immensely powerful. You might decide you want to calculate something in a different way, so you can go back to the raw data to requery it. You would not be able to do this if you had discarded the source data due to space constraints. You might decide that you want to dig into why some aggregate value exhibits strange behavior. You couldn't do this if you had deleted the data that contributed to the aggregation.

Scaling compute is equally powerful. BigQuery resources are denominated in terms of "slots," which are, roughly speaking, about half of a CPU core. BigQuery uses slots as an abstraction to indicate how many physical compute resources are available. Queries running too slow? Just add more slots. More people want to create reports? Add more slots. Want to cut back on your expenses? Decrease your slots.

Because BigQuery is a multitenant system that manages large pools of hardware resources, it is able to dole out the slots on a per-query or per-user basis. It is possible to reserve hardware for your project or organization, or you can run your queries in the shared on-demand pool. By sharing resources in this way, BigQuery can devote very large amounts of computing power to your queries. If you need more computing power than is available in the on-demand pool, you can purchase more via the BigQuery Reservation API.

Several BigQuery customers have reservations in the tens of thousands of slots, which means that if they run only one query at a time, those queries can consume tens of thousands of CPU cores at once. With some reasonable assumptions about numbers of CPU cycles per processed row, it is pretty easy to see that these instances can process billions or even trillions of rows per second.

In BigQuery, there are some customers that have petabytes of data but use a relatively small amount of it on a daily basis. Other customers store only a few gigabytes of data but perform complex queries using thousands of CPUs. There isn't a one-size-fits-all approach that works for all use cases. Fortunately, the separation of compute and storage allows BigQuery to accommodate a wide range of customer needs.

Storage and Networking Infrastructure

BigQuery differs from other cloud data warehouses in that queries are served primarily from spinning disks in a distributed filesystem. Most competitor systems need to cache data within compute nodes to get good performance. BigQuery, on the other hand, relies on two systems unique

to Google, the Colossus File System and Jupiter networking, to ensure that data can be queried quickly no matter where it physically resides in the compute cluster.

Managed Storage

BigQuery's storage system is built on the idea that when you're dealing with structured storage, the appropriate abstraction is the table, not the file. Some other cloud-based and open source data processing systems expose the concept of the file to users, which puts users on the hook for managing file sizes and ensuring that the schema remains consistent. Even though creating files of an appropriate size for a static data store is possible, it is notoriously difficult to maintain optimal file sizes for data that is changing over time. Similarly, it is difficult to maintain a consistent schema when you have a large number of files with self-describing schemas (e.g., Avro or Parquet)—typically, every software update to systems producing those files results in changes to the schema. BigQuery ensures that all the data held within a table has a consistent schema and enforces a proper migration path for historical data. By abstracting the underlying data formats and file sizes from the user, BigQuery can provide a seamless experience so that queries are always fast.

Managed storage is strongly typed, which means that data is validated at entry to the system. Because BigQuery manages the storage and allows users to interact with this storage only via its APIs, it can count on the underlying data not being modified outside of BigQuery. Thus, BigQuery can guarantee to not throw a validation error at read time about any of the data present in its managed storage. This guarantee also implies an authoritative schema, which is useful when figuring out how to query your tables. Besides improving query performance, the presence of an authoritative schema helps when trying to make sense of what data you have because a BigQuery schema contains not just type information but also annotations and table descriptions about how the fields can be used.

One downside of managed storage is that it is more difficult to directly access and process the data using other frameworks. For example, had the data been available at the abstraction level of files, you might have been able to directly run a Hadoop job over a BigQuery dataset. BigQuery addresses this issue by providing a structured parallel API to read the data. This API lets you read at

full speed from Spark or Hadoop jobs, but it also provides extra features, like projection, filtering, and dynamic rebalancing.

Integration with Google Cloud Platform

Google Cloud follows the design principle called "separation of responsibility," wherein a small number of high-quality, highly focused products integrate tightly with each other. It is, therefore, important to consider the entire Google Cloud Platform (GCP) when comparing BigQuery with other database products.A number of different GCP products extend the usefulness of BigQuery or make it easier to understand how BigQuery is being used.

EX.NO 8: **Store and Query massive Datasets using Google Cloud BigQuery**

Date:

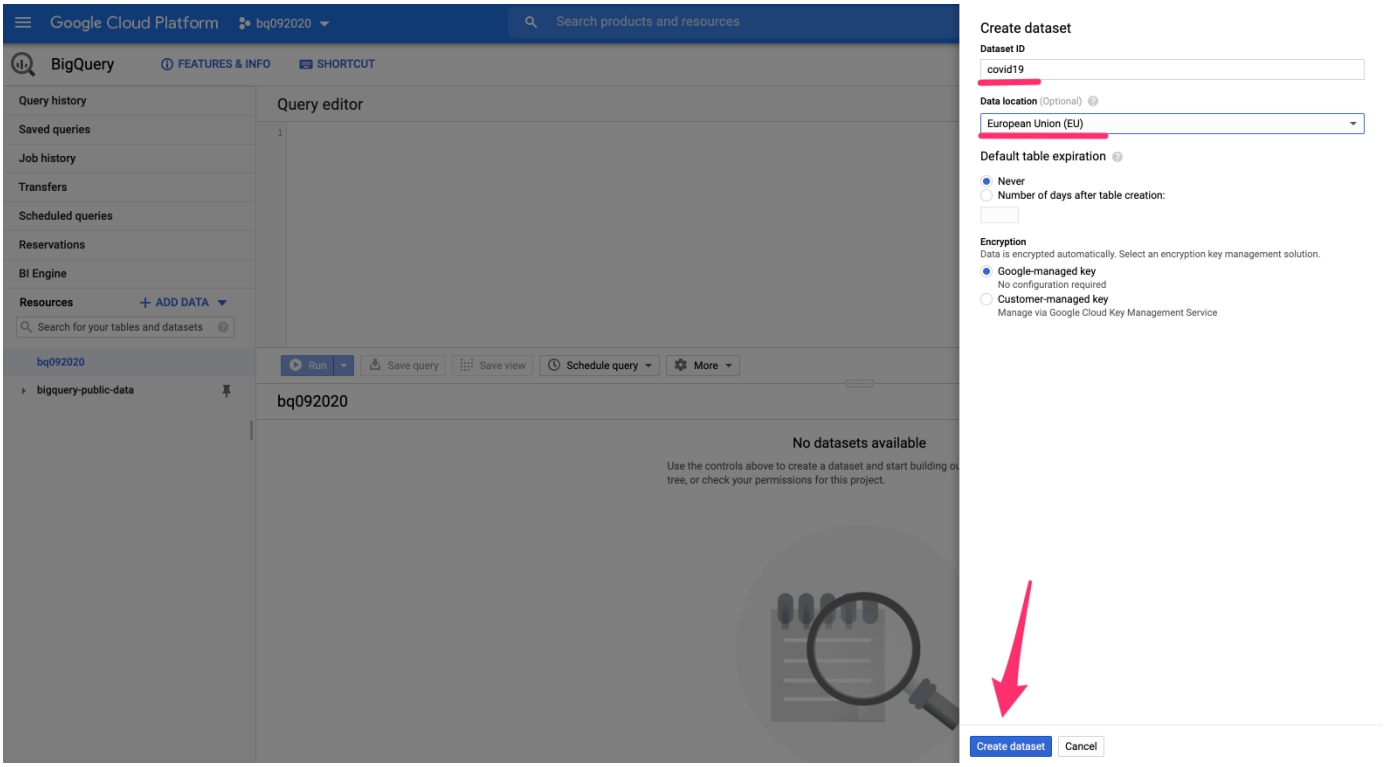
AIM:

PROCEDURE with SCREENSHOTS:

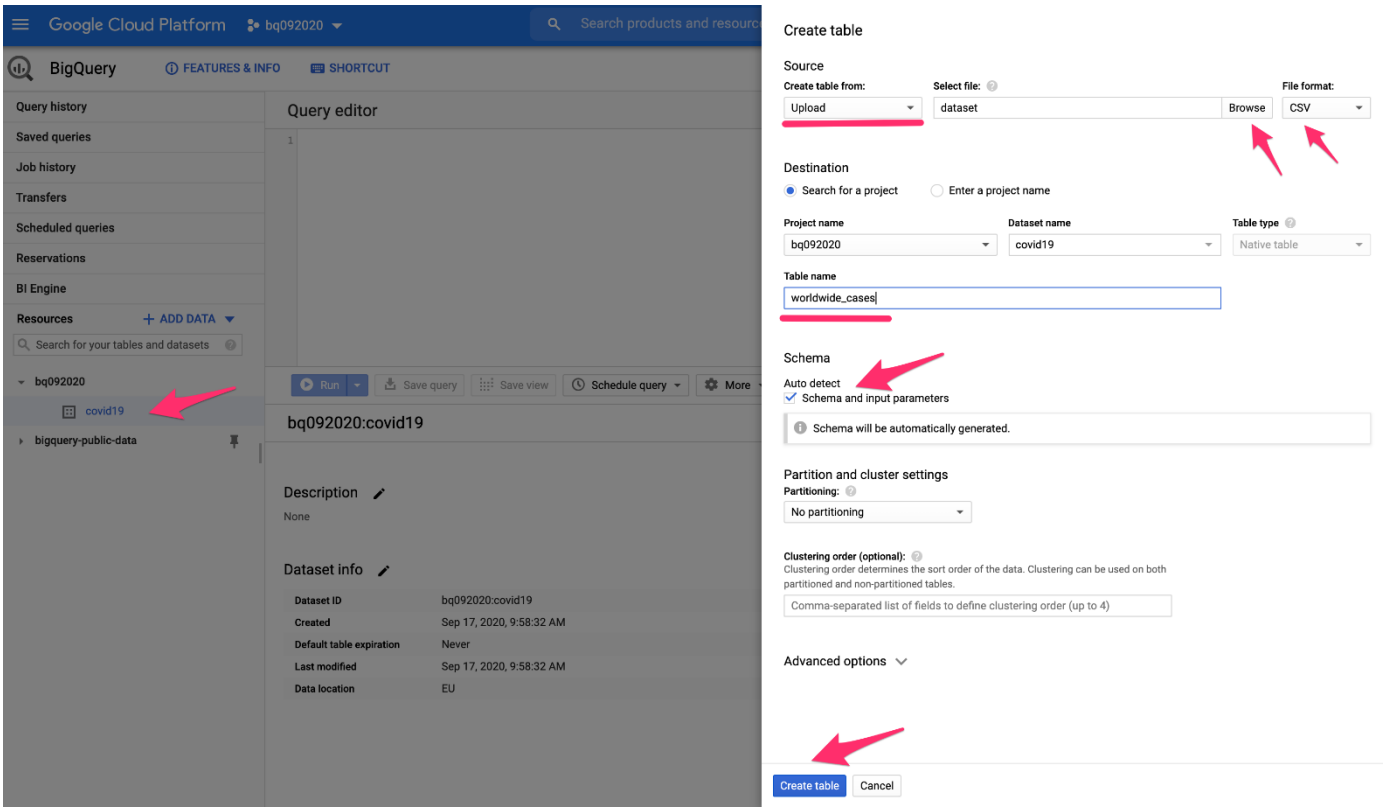
To Store Datasets using Google Cloud BigQuery.:

Step 1: In the Google Cloud Platform, navigate to the Google BigQuery Console under the Big Data section.

Step 2: Find the “CREATE DATASET” button on the right-side panel and initiate the creation process. Give the dataset a unique identifier and select the geographical location to store and process the data. Save it using the button at the bottom of the panel.



Step 3 : Select the newly created dataset and press the CREATE TABLE button. Use Upload as the source method, CSV as the file format, and select the local dataset file from your machine. Give it a table name and select the Auto Detect option for the schema. Save it using the button at the bottom of the panel.



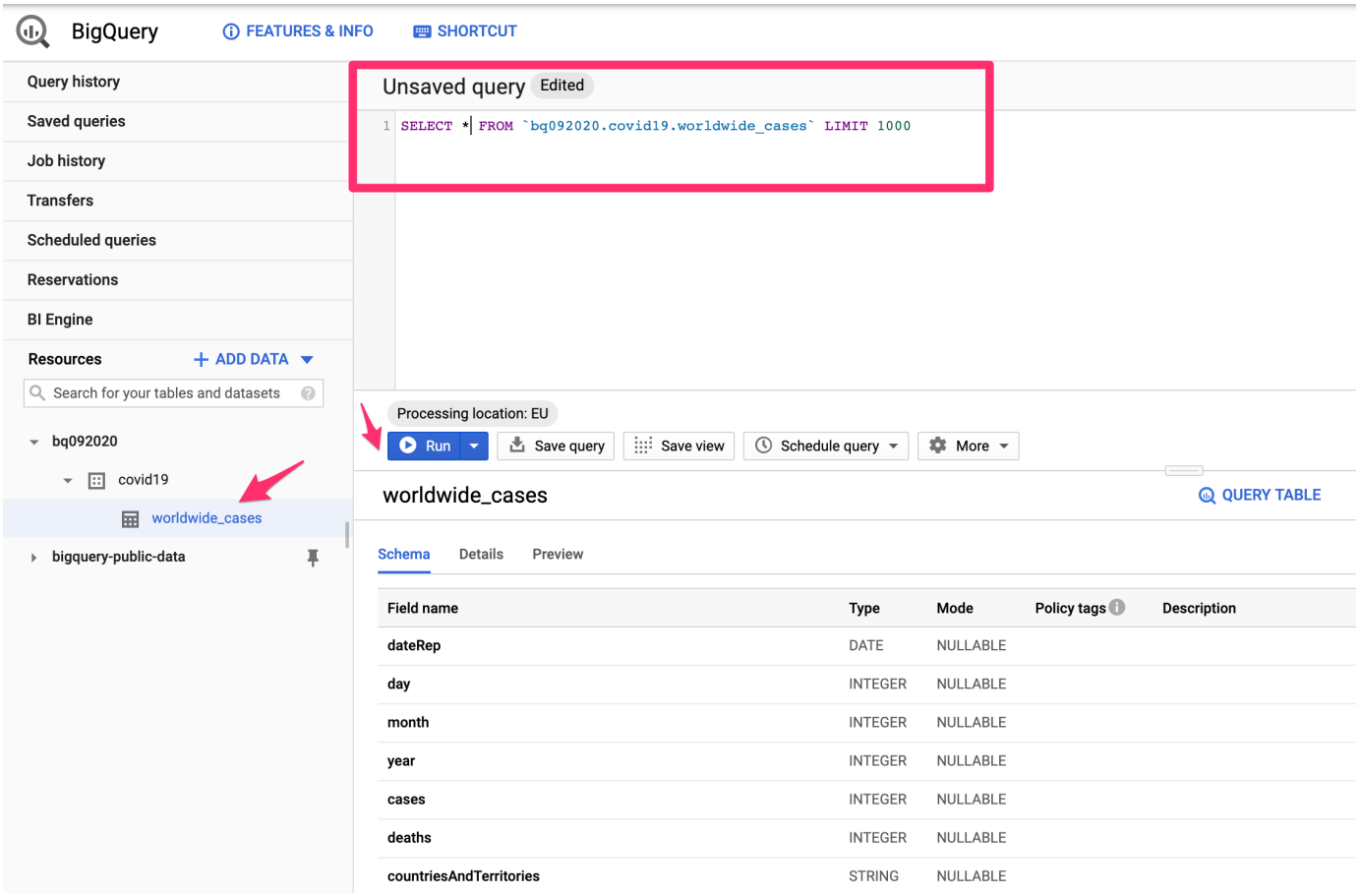
To Query Datasets using Google Cloud BigQuery:

With the dataset uploaded and stored in BigQuery, you will be able to start immediately querying the data using standard SQL.

Step 1 : In the panel, try a simple query such as

SELECT * FROM `bq092020.covid19.worldwide_cases` LIMIT 1000

to retrieve up to a thousand rows.



Step 2 : BigQuery Analytics are quite powerful. The interface gives access to full-fledged SQL capabilities, therefore you can use a more advanced query such as.

SELECT countriesAndTerritories, sum(cases) AS N_Cases, sum(deaths) AS N_Deaths, count(*) AS

N_Rows FROM `bq092020.covid19.worldwide_cases` GROUP BY countriesAndTerritories

LIMIT 1000

Running the query above will provide aggregated results the Number of Cases and Deaths per Country/Territory.

Query editor

```
SELECT countriesAndTerritories, sum(cases) AS N_Cases, sum(deaths) AS N_Deaths, count(*) AS N_Rows
FROM `bq092020.covid19.worldwide_cases`
GROUP BY countriesAndTerritories
LIMIT 1000
```

Processing location: EU

Run | Save query | Save view | Schedule query | More

Query results

SAVE RESULTS | EXPLORE DATA

Query complete (0.3 sec elapsed, 1.1 MB processed)

Job information | Results | JSON | Execution details

Row	countriesAndTerritories	N_Cases	N_Deaths	N_Rows
1	Andorra	1438	53	187
2	United_Arab_Emirates	80940	401	255
3	Afghanistan	38855	1436	251
4	Antigua_and_Barbuda	95	3	181
5	Anguilla	3	0	174
6	Albania	11672	340	192
7	Armenia	46119	920	252
8	Angola	3439	134	179
9	Argentina	565432	11710	194
10	Austria	34744	757	261
11	Australia	26738	816	261

Evaluation by faculty	
Criteria	Marks
	/20
	/25
	/20
	/10
Total	/75
Faculty Signature with Date	

RESULT: