Creating Date-Partitioned Tables in BigQuery

Overview

This project explores **BigQuery**, Google's fully managed, serverless data warehouse, focusing on **date-partitioned tables** to optimize query performance and reduce costs. The lab uses an **ecommerce dataset** from Google Analytics to demonstrate how partitioning improves efficiency when querying large datasets.

Key Objectives

- Query partitioned tables.
- Create custom partitioned tables.
- Implement **auto-expiring partitions** for cost and compliance management.

Setup & Environment

Tools & Services Used

- **BigQuery** (Google Cloud's analytics database)
- **Public Datasets** (NOAA weather data, Google Analytics ecommerce data)
- **SQL** (Standard SQL dialect)

Dataset Sources

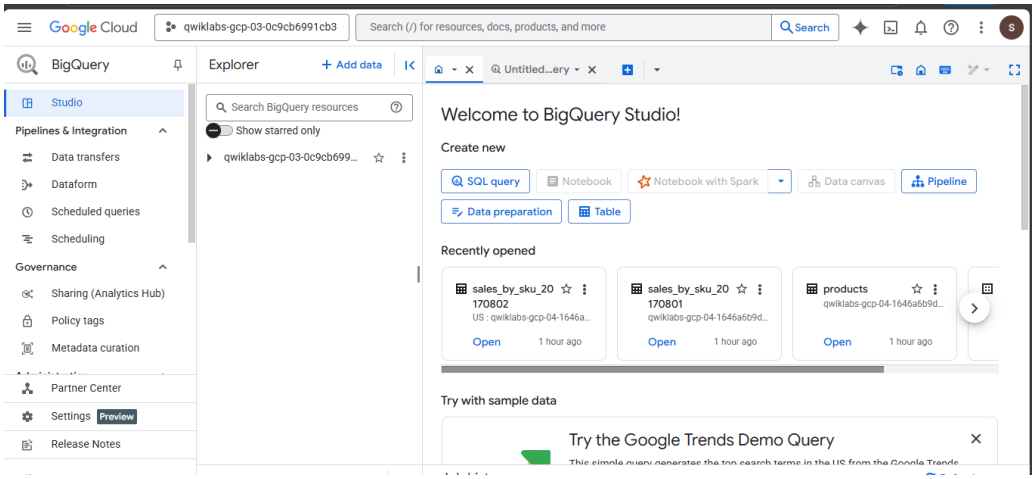
1. **Google Analytics Ecommerce Dataset** (data-to-insights.ecommerce.all_sessions_raw)
2. **NOAA GSOD Weather Dataset** (bigquery-public-data.noaa_gsod)

Open the BigQuery console

1. In the Google Cloud Console, select **Navigation menu** > **BigQuery**.

   The **Welcome to BigQuery in the Cloud Console** message box opens. This message box provides a link to the quickstart guide and the release notes.

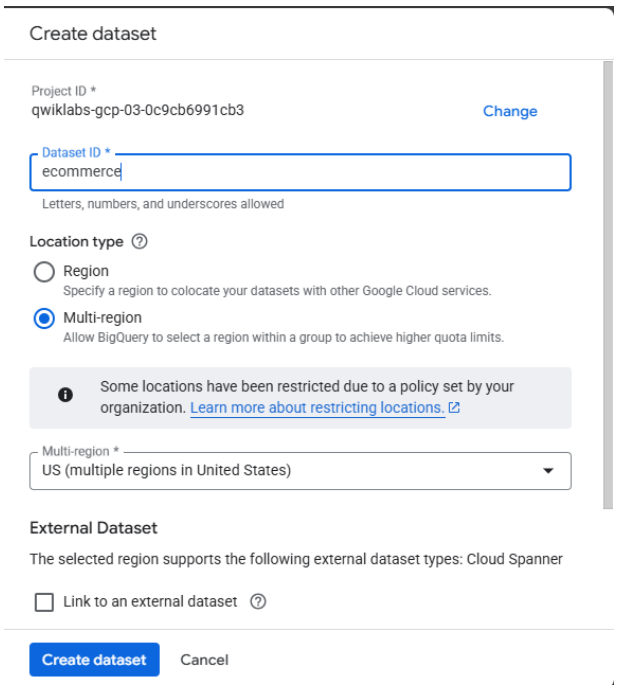
2. Click **Done**.

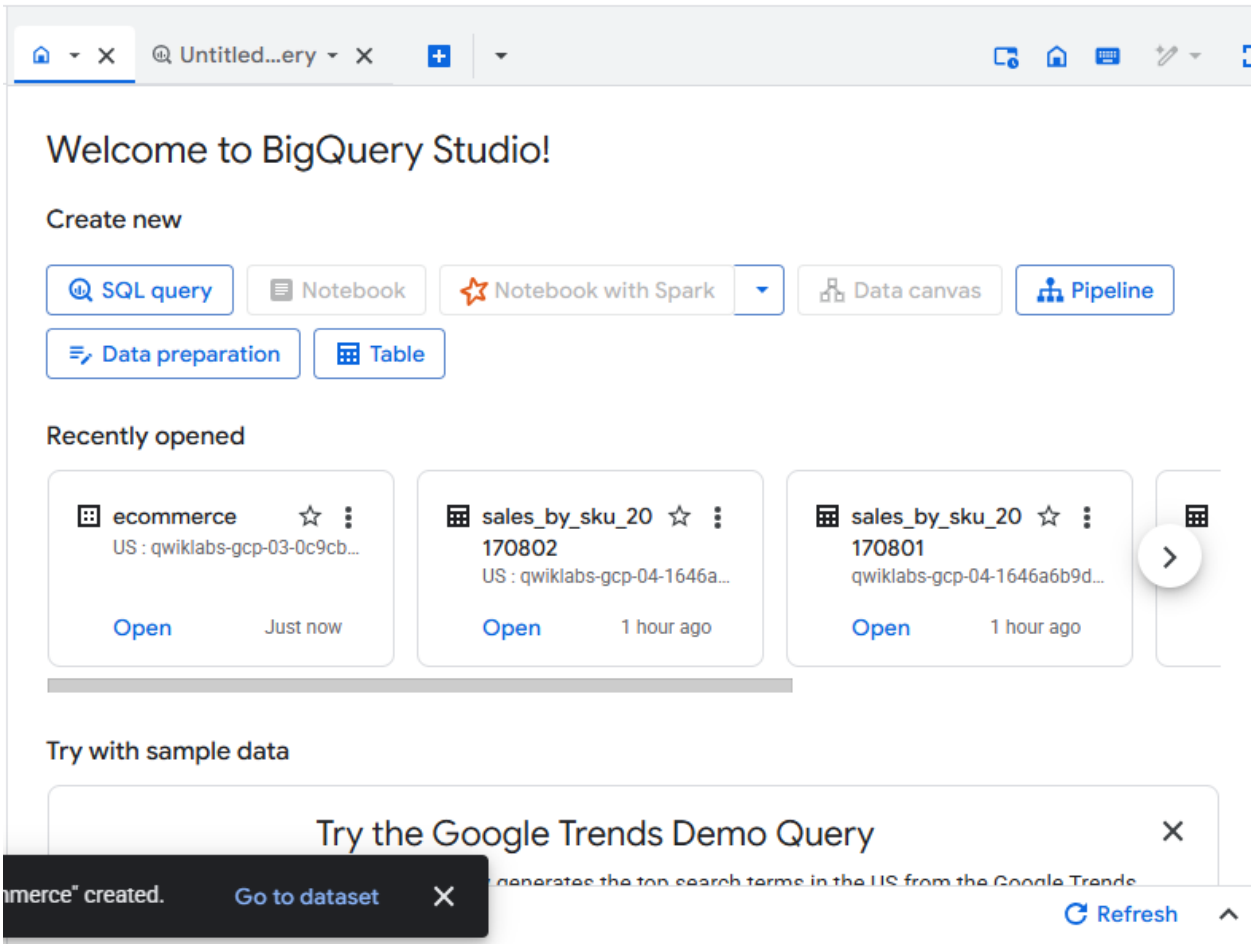


Task 1. Create a new dataset

1. First, you will create a dataset to store your tables.
2. In the **Explorer pane**, near your project id, click on **View actions** then click **Create dataset**.
3. Set **Dataset ID** to **ecommerce**.

   Leave the other options at their default values (Data Location, Default table Expiration).
4. Click **Create dataset**.





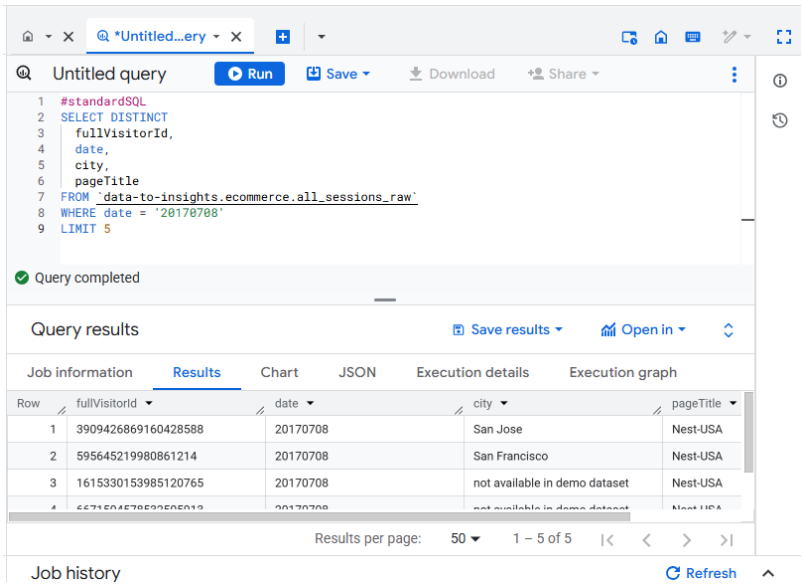
Task 2. Create tables with date partitions

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

Now create a new table and bind a date or timestamp column as a partition. Before we do that, let's explore the data in the non-partitioned table first.

Query web page analytics for a sample of visitors in 2017

#standardSQL SELECT DISTINCT fullVisitorId, date, city, pageTitle FROM `data-to-insights.ecommerce.all_sessions_raw` WHERE date = '20170708' LIMIT 5



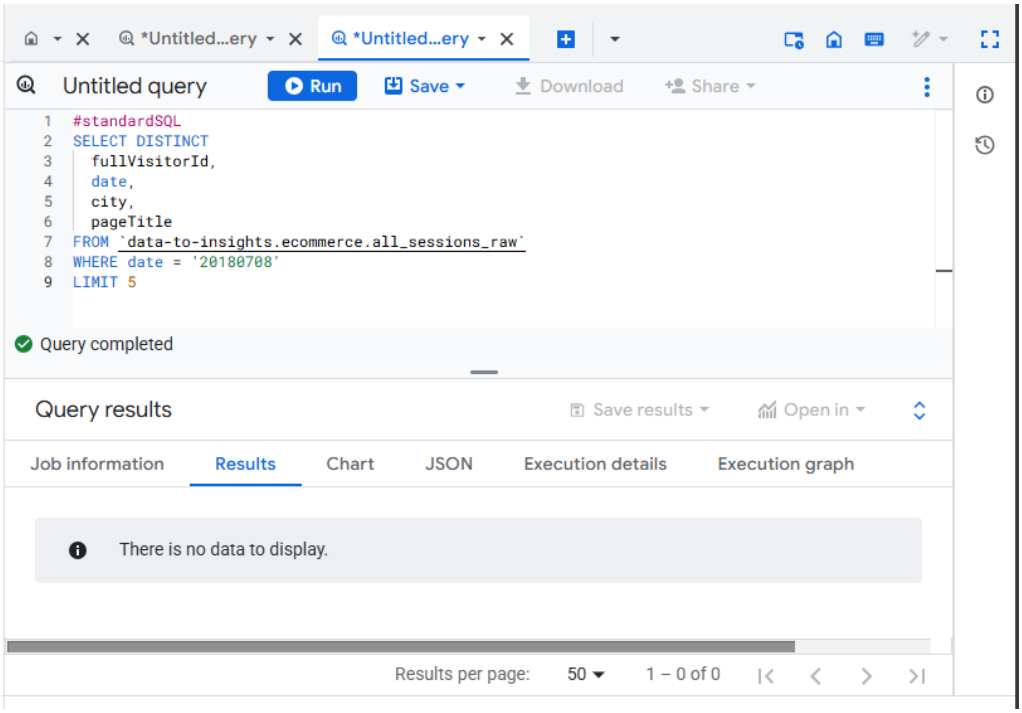
2. Click **Run**.

Query web page analytics for a sample of visitors in 2018

Let's modify the query to look at visitors for 2018 now.

1. Click **+ SQL query** to clear the **Query Editor**, then add this new query. Note the WHERE date parameter is changed to 20180708:

#standardSQL SELECT DISTINCT fullVisitorId, date, city, pageTitle FROM `data-to-insights.ecommerce.all_sessions_raw` WHERE date = '20180708' LIMIT 5



Click **Run**

Common use-cases for date-partitioned tables

Scanning through the entire dataset everytime to compare rows against a WHERE condition is wasteful. This is especially true if you only really care about records for a specific period of time like:

- All transactions for the last year
- All visitor interactions within the last 7 days
- All products sold in the last month

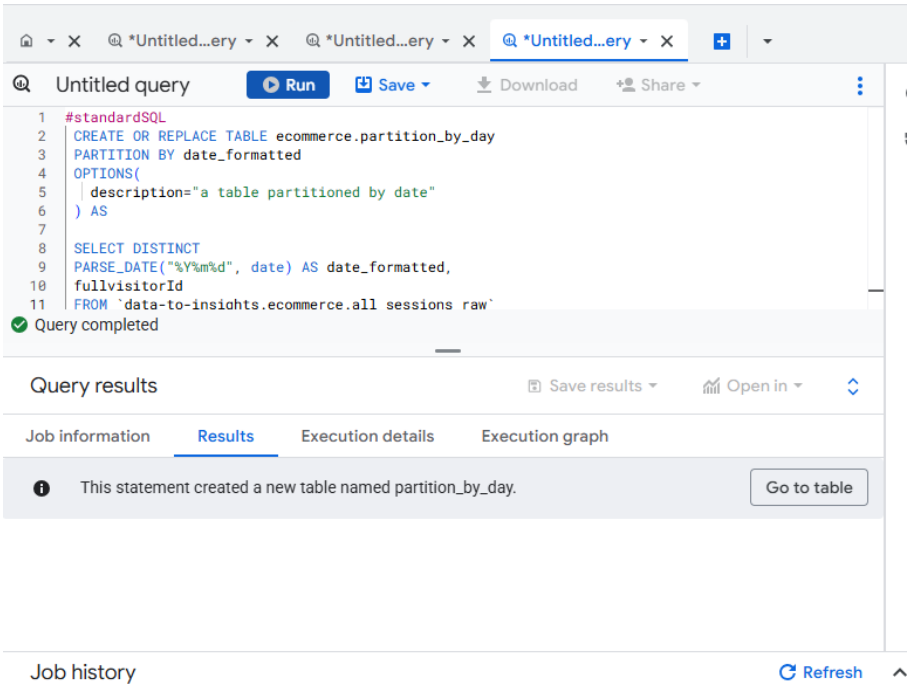
Instead of scanning the entire dataset and filtering on a date field like we did in the earlier queries, Now set up a date-partitioned table. This allows you to completely ignore scanning records in certain partitions if they are irrelevant to our query.

Create a new partitioned table based on date

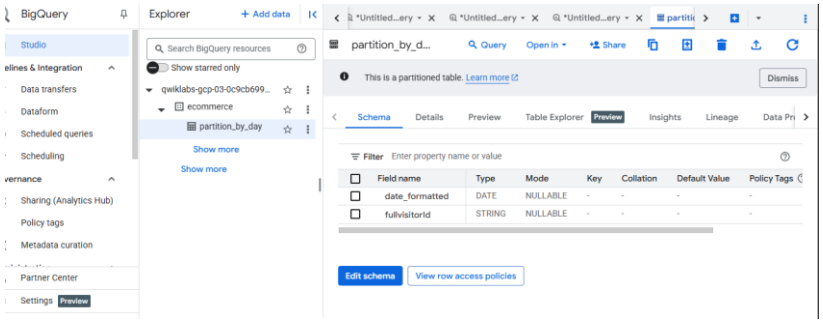
1. **Click + SQL query , add the below query, then click Run:**

#standardSQL CREATE OR REPLACE TABLE ecommerce.partition_by_day PARTITION BY date_formatted OPTIONS(description="a table partitioned by date") AS

SELECT DISTINCT PARSE_DATE("%Y%m%d", date) AS date_formatted, fullvisitorId FROM `data-to-insights.ecommerce.all_sessions_raw`



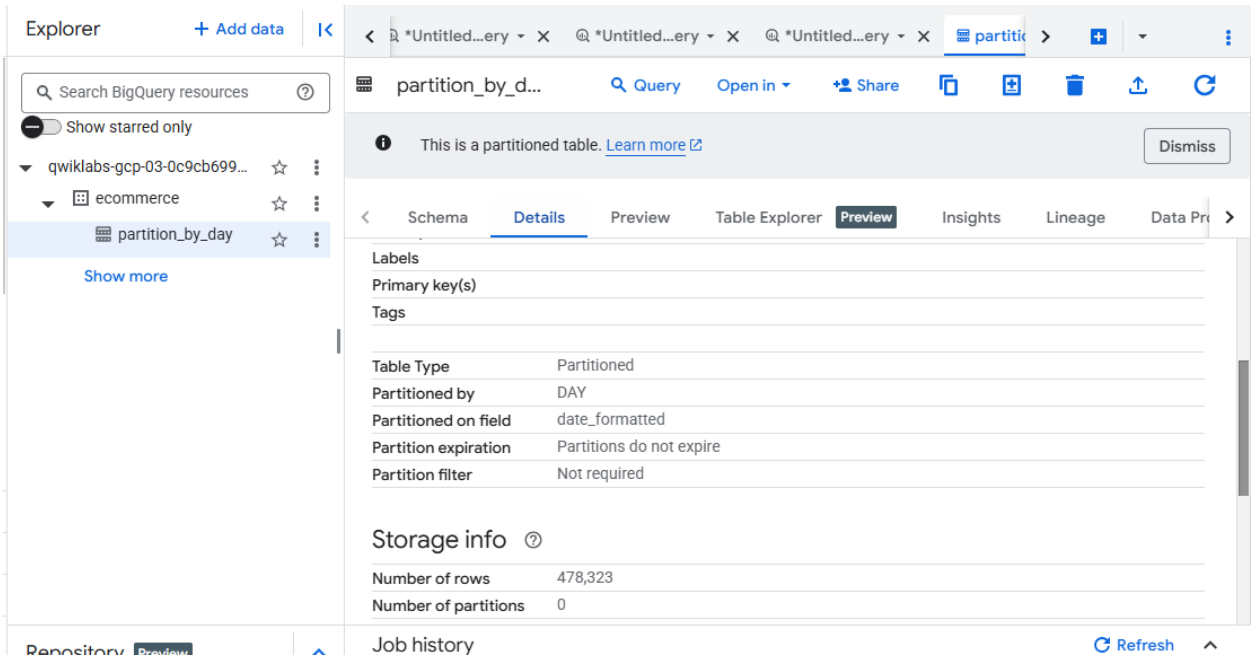
2. Click on the **ecommerce** dataset, then select the new **partiton_by_day** table:



3. Click on the **Details** tab.

   Confirm that you see:

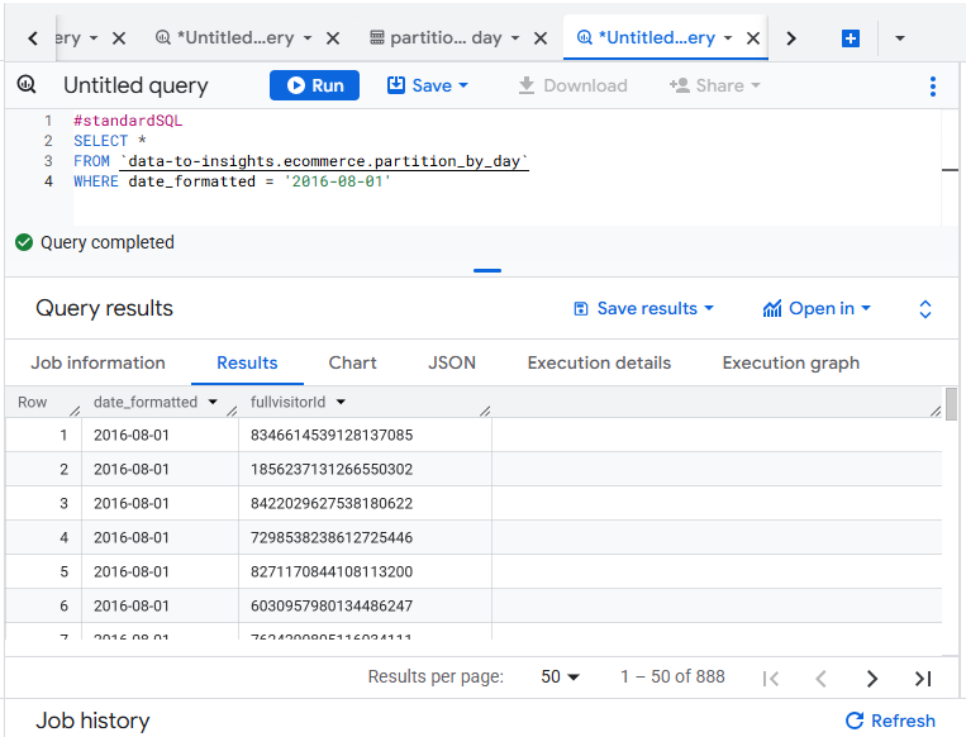
- Partitioned by: Day
- Partitioning on: date_formatted



Task 3. Review results from queries on a partitioned table

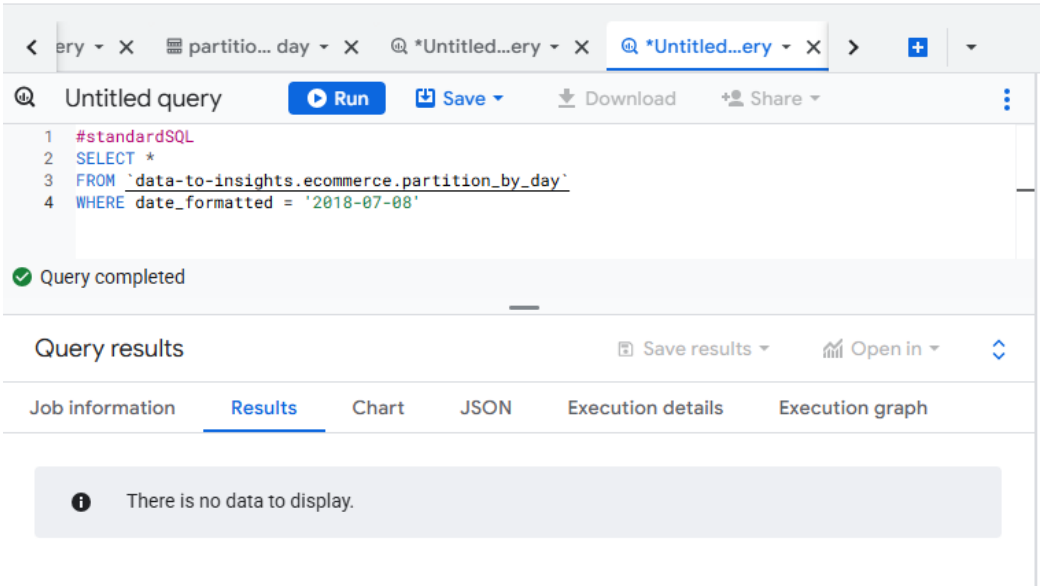
1. Run the below query, and note the total bytes to be processed:

#standardSQL SELECT * FROM `data-to-insights.ecommerce.partition_by_day` WHERE date_formatted = '2016-08-01'



2. Now run the below query, and note the total bytes to be processed:

#standardSQL SELECT * FROM `data-to-insights.ecommerce.partition_by_day` WHERE date_formatted = '2018-07-08'

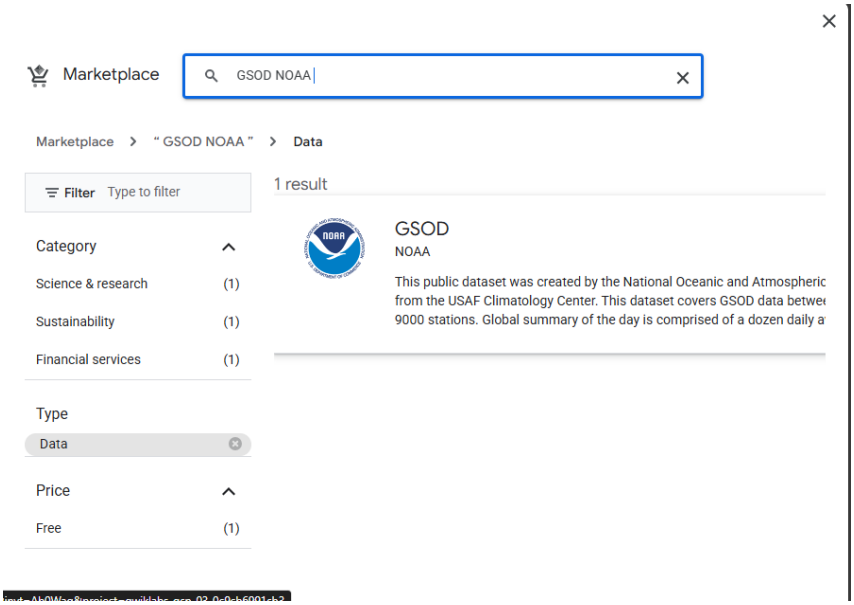


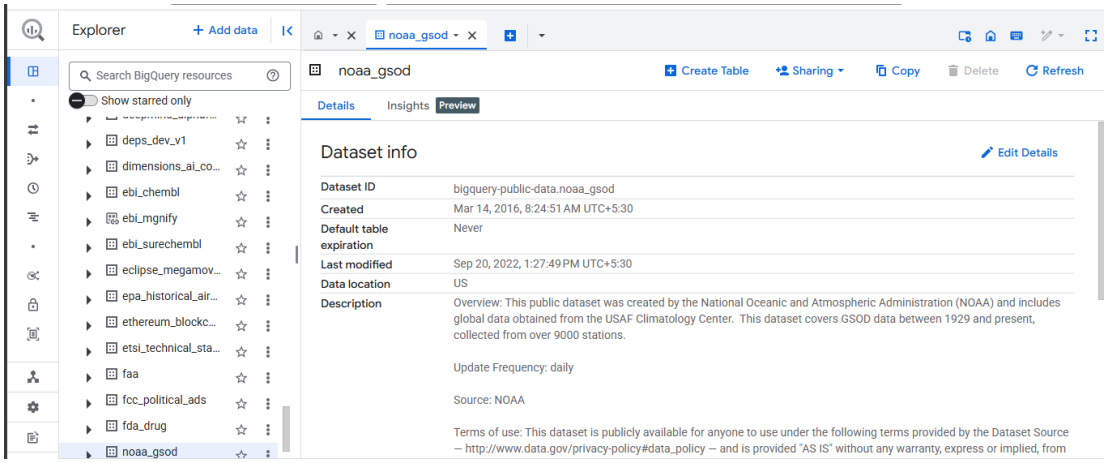
Task 4. Create an auto-expiring partitioned table

Auto-expiring partitioned tables are used to comply with data privacy statutes, and can be used to avoid unnecessary storage (which you'll be charged for in a production environment). If you want to create a rolling window of data, add an expiration date so the partition disappears after you're finished using it.

Explore the available NOAA weather data tables

1. In the left menu, in Explorer, click on **+ Add data** and select **Public datasets**.
2. Search for **GSOD NOAA** then select the dataset.
3. Click on **View Dataset**.
4. **Scroll through** the tables in the **noaa_gsod** dataset (which are manually sharded and not partitioned):





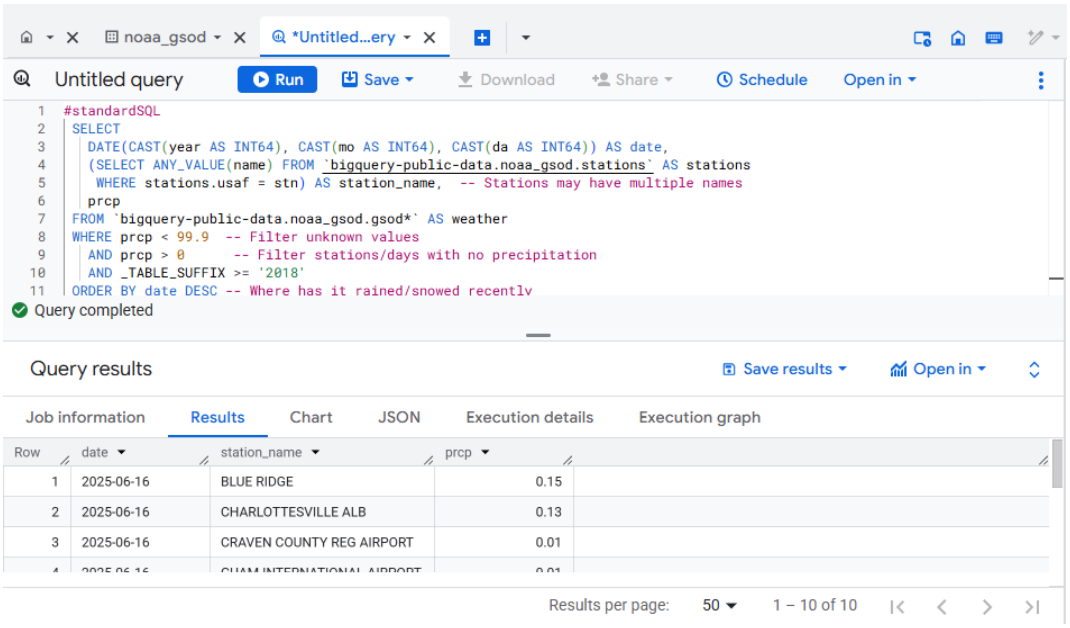
Your goal is to create a table that:

- Queries on weather data from 2018 onward
- Filters to only include days that have had some precipitation (rain, snow, etc.)
- Only stores each partition of data for 90 days from that partition's date (rolling window)

1. First, **copy and paste** this below query:

#standardSQL SELECT DATE(CAST(year AS INT64), CAST(mo AS INT64), CAST(da AS INT64)) AS date, (SELECT ANY_VALUE(name) FROM `bigquery-public-data.noaa_gsod.stations` AS stations WHERE stations.usaf = stn) AS station_name, --

Stations may have multiple names prcp FROM `bigquery-public-data.noaa_gsod.gsod*` AS weather WHERE prcp < 99.9 -- Filter unknown values AND prcp > 0 -- Filter stations/days with no precipitation AND _TABLE_SUFFIX >= '2018' ORDER BY date DESC -- Where has it rained/snowed recently LIMIT 10



2. Click **Run**.
3. Confirm the date is properly formatted and the precipitation field is showing non-zero values.

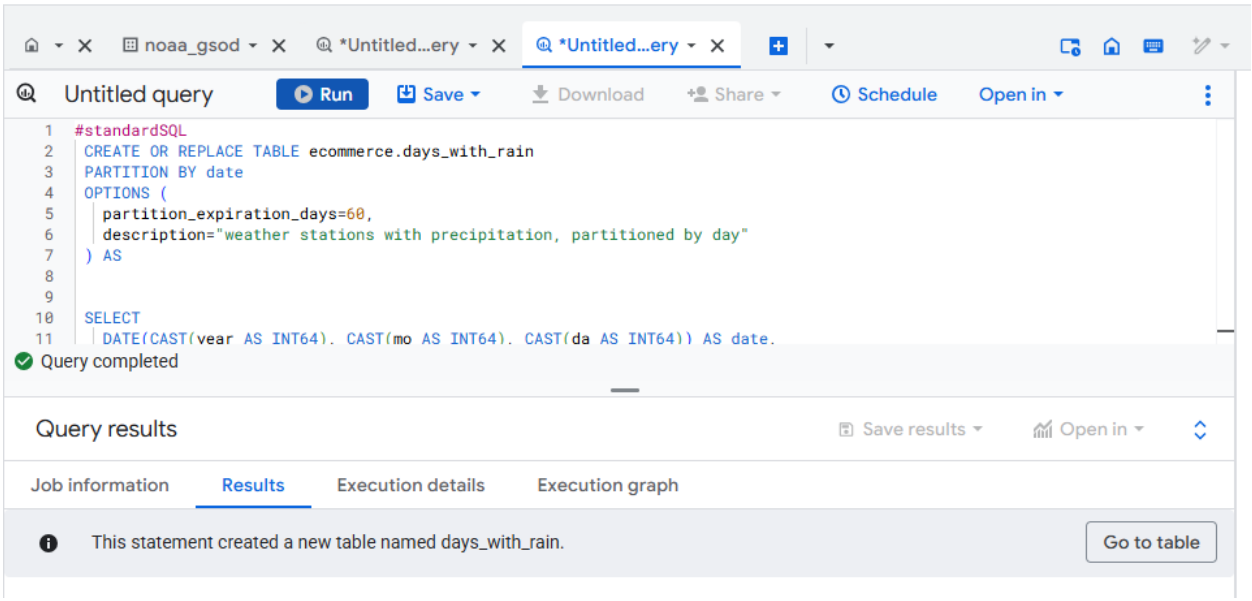
Task 5. Your turn: create a partitioned table

- Modify the previous query to create a table with the below specifications:
  - Table name: ecommerce.days_with_rain
  - Use the date field as your PARTITION BY
  - For OPTIONS, specify partition_expiration_days = 60
  - Add the table description = "weather stations with precipitation, partitioned by day"

Your query should look like this:

#standardSQL CREATE OR REPLACE TABLE ecommerce.days_with_rain PARTITION BY date OPTIONS (partition_expiration_days=60, description="weather stations with precipitation, partitioned by day") AS

SELECT DATE(CAST(year AS INT64), CAST(mo AS INT64), CAST(da AS INT64)) AS date, (SELECT ANY_VALUE(name) FROM `bigquery-public-data.noaa_gsod.stations` AS stations WHERE stations.usaf = stn) AS station_name, -- Stations may have multiple names prcp FROM `bigquery-public-data.noaa_gsod.gsod*` AS weather WHERE prcp < 99.9 -- Filter unknown values AND prcp > 0 -- Filter AND _TABLE_SUFFIX >= '2018'

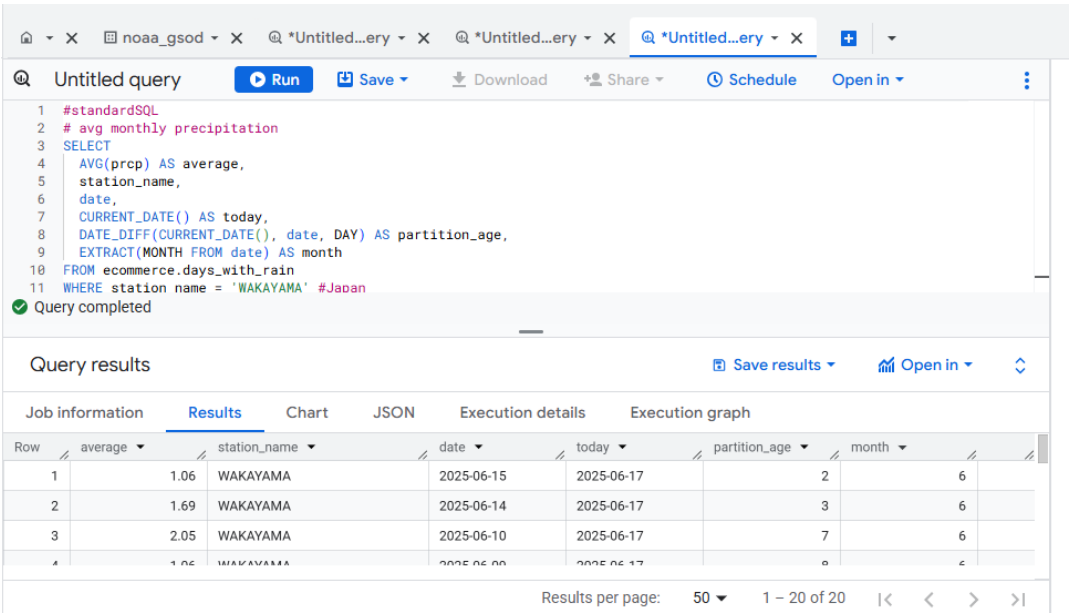


Confirm data partition expiration is working

To confirm you are only storing data from 60 days in the past up until today, run the DATE_DIFF query to get the age of your partitions, which are set to expire after 60 days.

Below is a query which tracks the average rainfall for the NOAA weather station in Wakayama, Japan which has significant precipitation.

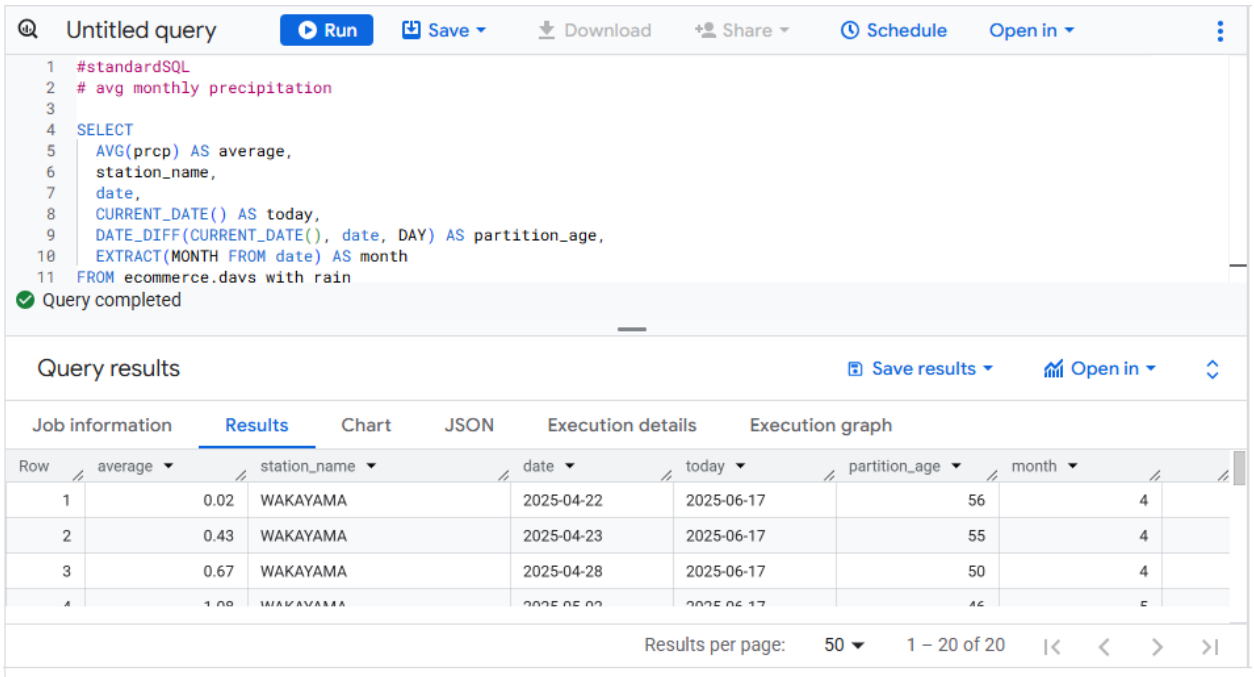
- Add this query and run it:



Task 6. Confirm the oldest partition_age is at or below 60 days

Update the ORDER BY clause to show the oldest partitions first.

- Add this query and run it:



Key Learnings

1. **Partitioning Improves Efficiency**

a. Reduces **query costs** by scanning only relevant partitions.

b. Avoids full table scans (e.g., 1.74 GB → 25 KB).

2. **Auto-Expiring Partitions for Cost Control**

a. Ensures **old data is automatically purged**.

b. Useful for **rolling time windows** (e.g., last 60 days).

3. **BigQuery Optimization Best Practices**

a. Use PARTITION BY for large tables.

b. Apply partition_expiration_days for transient data.

Conclusion

This project successfully demonstrated:

- ✅ **Creating and querying partitioned tables** in BigQuery.
- ✅ **Reducing query costs** by avoiding full scans.
- ✅ **Auto-expiring partitions** for efficient data lifecycle management.