Working with JSON, Arrays, and Structs in BigQuery

Task 1. Create a new dataset to store the tables

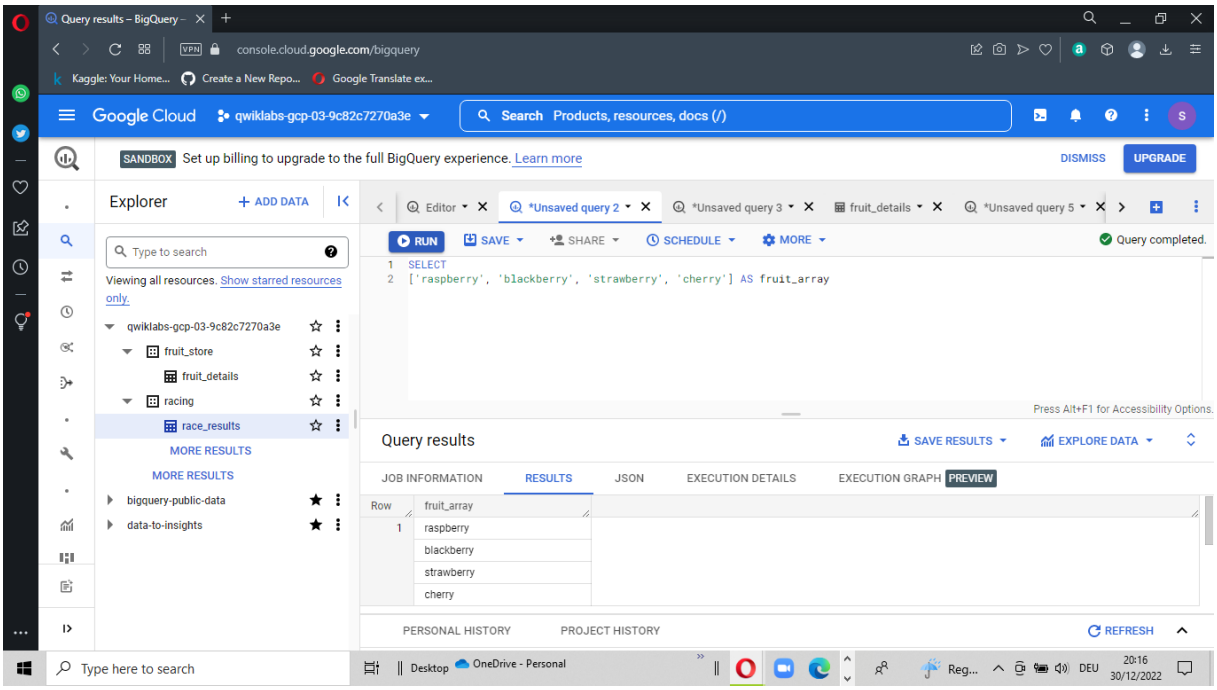
1. In your BigQuery, click the three dots next to your Project ID and select **Create dataset**:
2. Name the new dataset fruit_store. Leave the other options at their default values (Data Location, Default Expiration).

3. Click **Create dataset**.

Task 2. Practice working with arrays in SQL

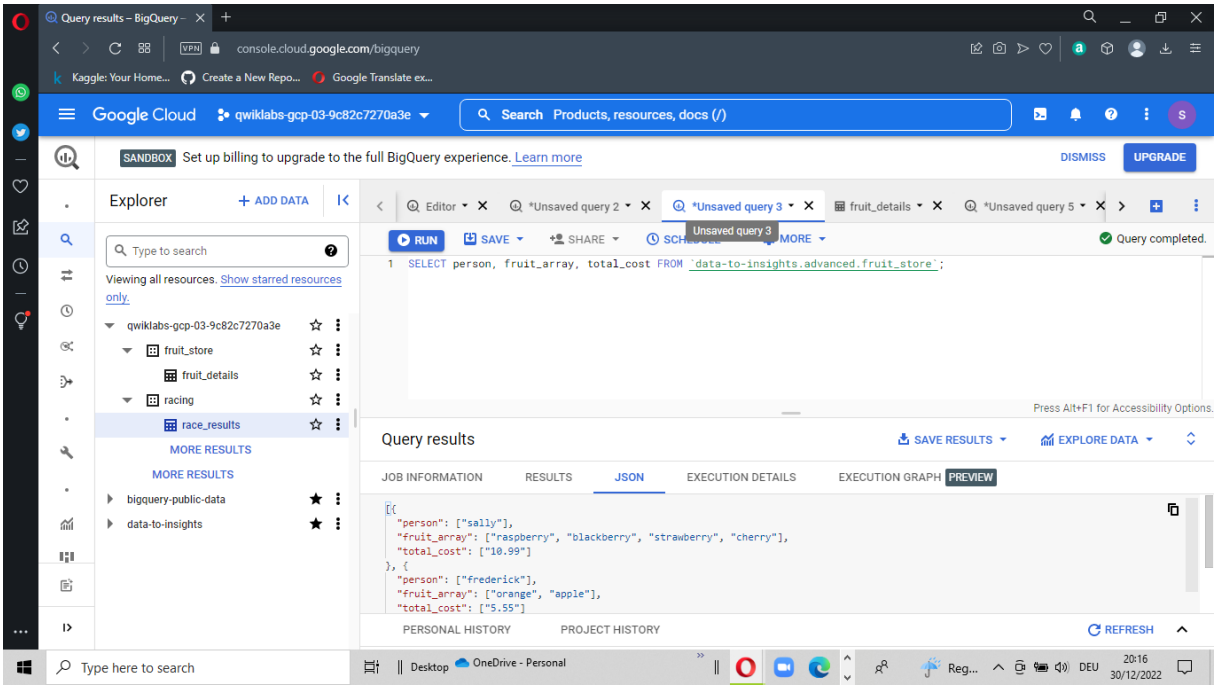
An array is simply a list of items in brackets [ ]
BigQuery visually displays arrays as *flattened*. It simply lists the value in the array vertically (note that all of those values still belong to a single row).

Solution:



Arrays can only share one data type (all strings, all numbers).
Here's the final table to query against:
After viewing the results, click the **JSON** tab to view the nested structure of the results.

Solution:



Loading semi-structured JSON into BigQuery

What if you had a JSON file that you needed to ingest into BigQuery?

Create a new table fruit_details in the dataset.

1. Click on fruit_store dataset, then click on the vertical 3-dots, and select **Open**.

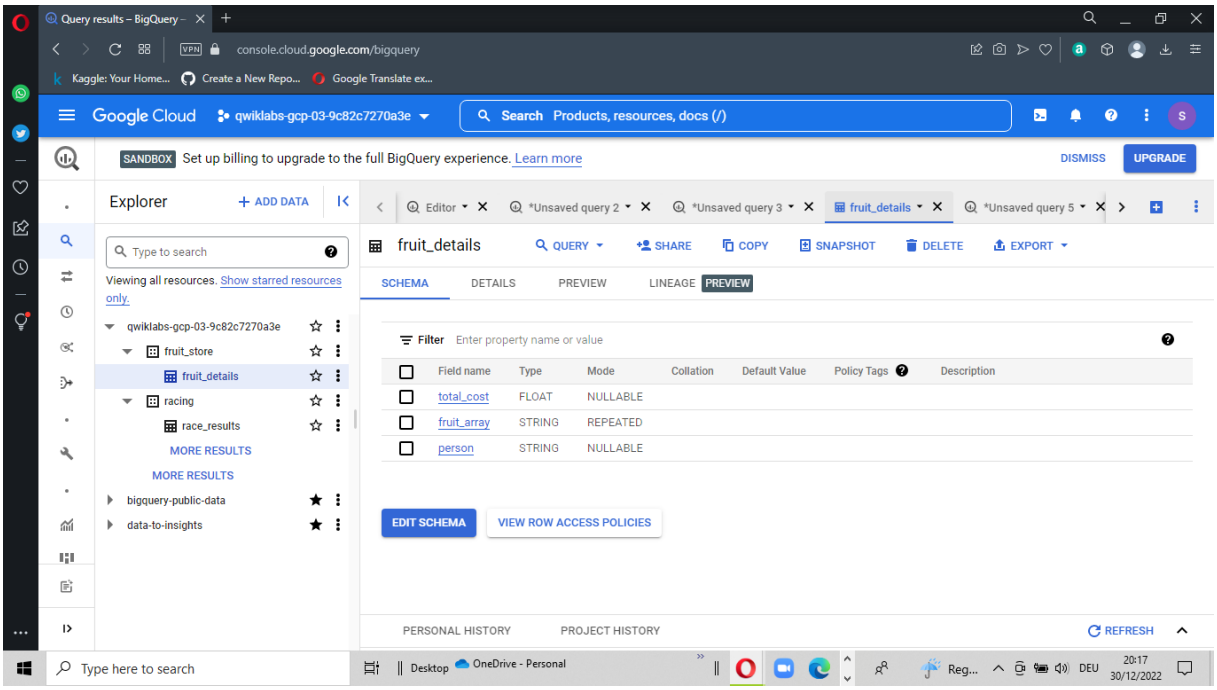
Now you will see the **Create Table** option.

2. Name the table fruit_details.
3. Add the following details for the table:

- **Source**: Choose **Google Cloud Storage** in the **Create table from** dropdown.
- **Select file from Cloud Storage bucket**: data-insights-course/labs/optimizing-for-performance/shopping_cart.json
- **File format**: JSONL (Newline delimited JSON)

4. Call the new table fruit_details.
5. Check the checkbox of **Schema (Auto detect)**.
6. Click **Create table**.

Solution:

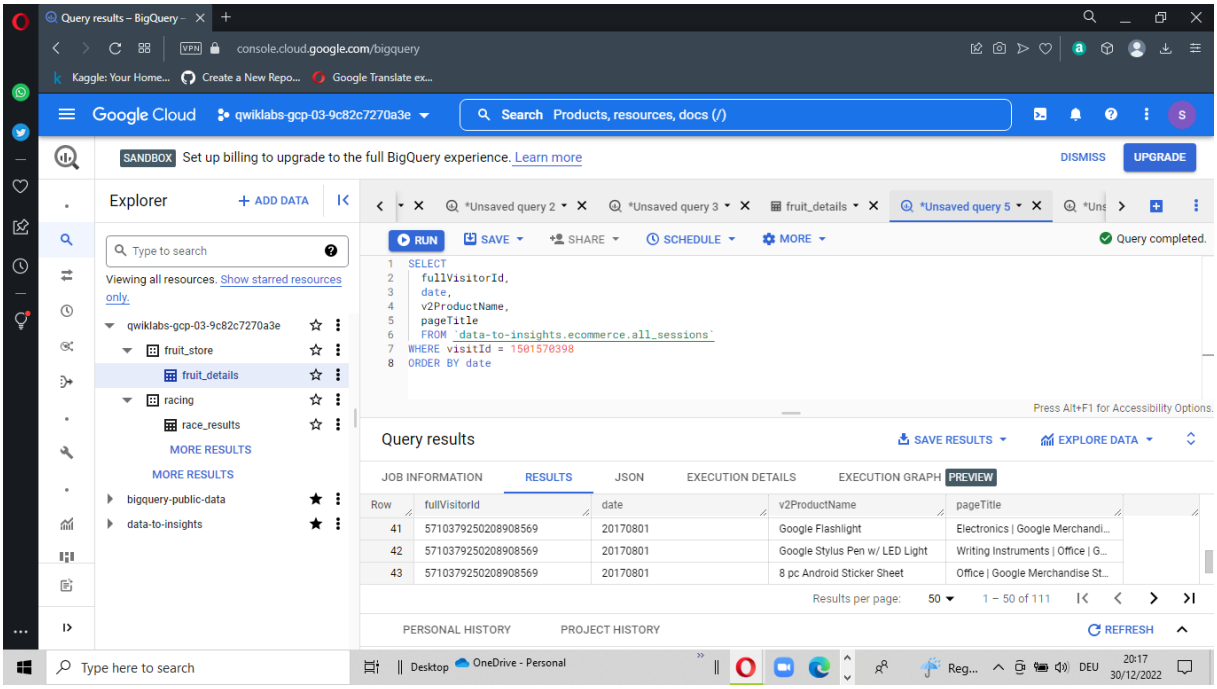


Task 3. Creating your own arrays with ARRAY_AGG()

Don't have arrays in your tables already? You can create them!

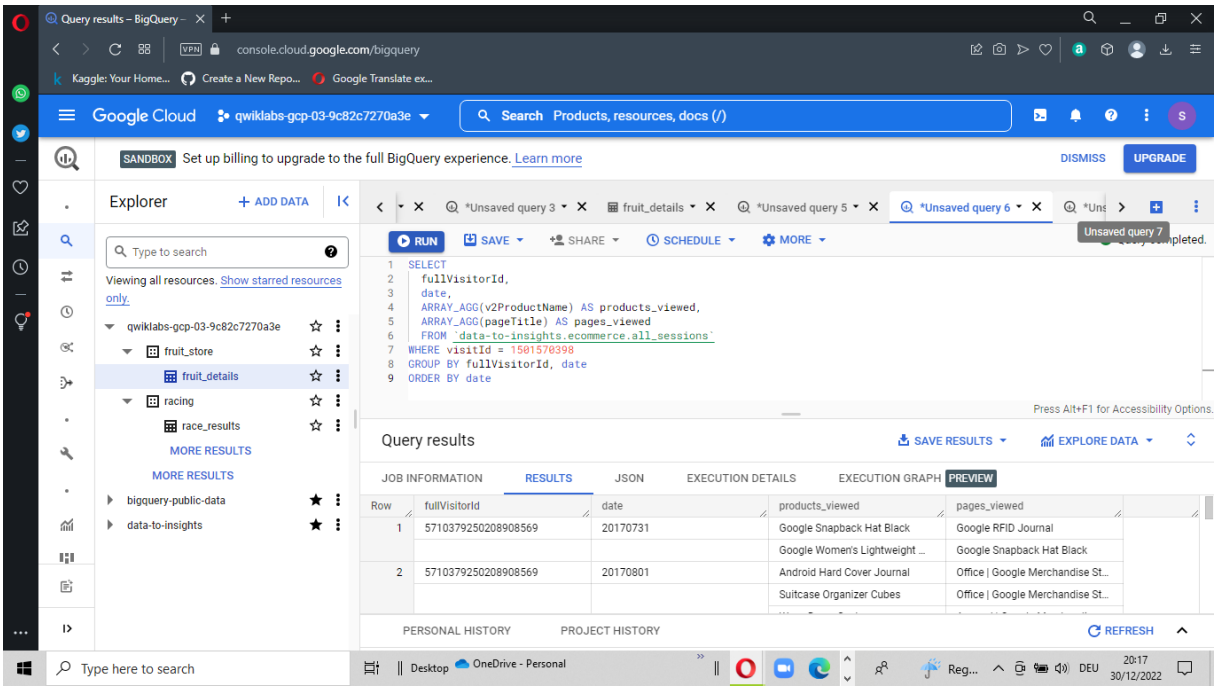
1. **Run** the query to explore this public dataset:

Solution:



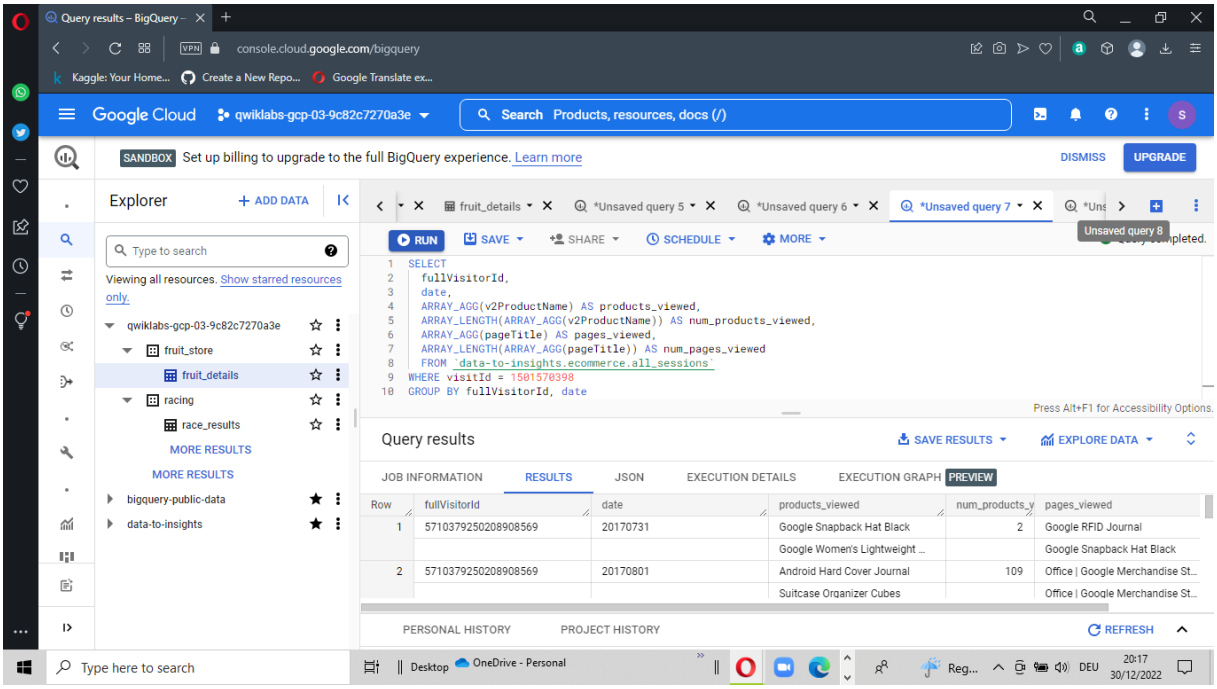
Now, use the ARRAY_AGG() function to aggregate our string values into an array.

Solution:



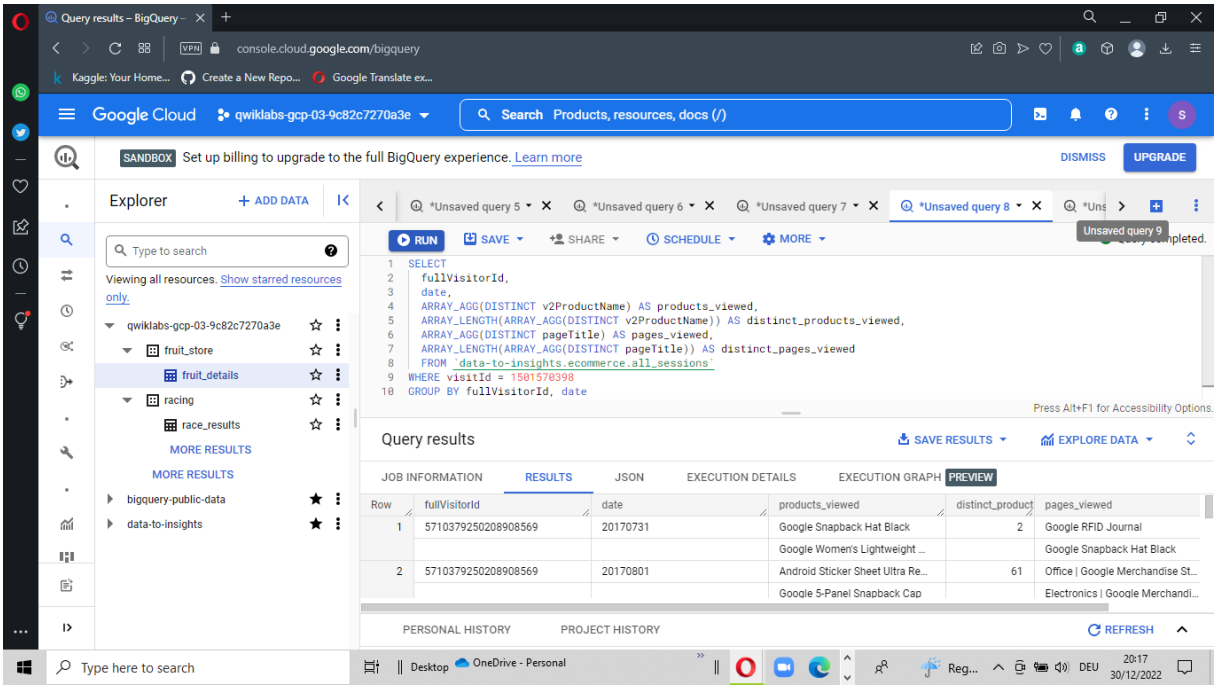
Next, use the ARRAY_LENGTH() function to count the number of pages and products that were viewed:

Solution:



Next, deduplicate the pages and products so you can see how many unique products were viewed by adding DISTINCT to ARRAY_AGG():

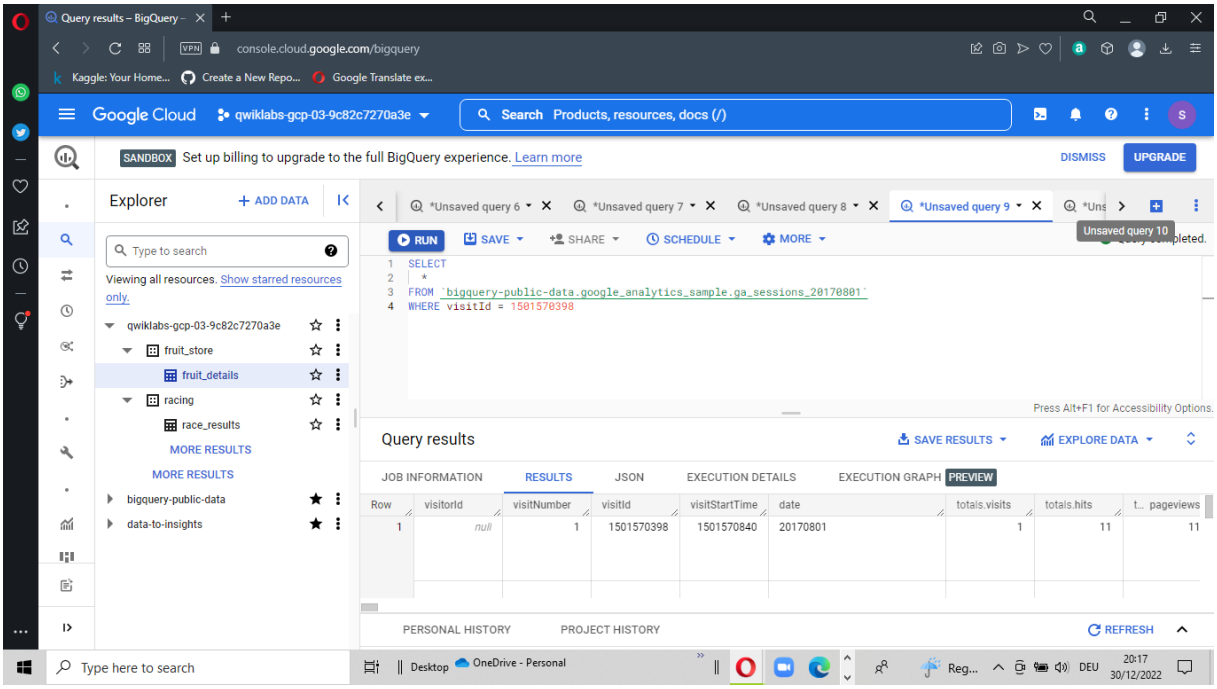
Solution:



Task 4. Querying datasets that already have arrays

The BigQuery Public Dataset for Google Analytics bigquery-public-data.google_analytics_sample has many more fields and rows than our course dataset data-to-insights.ecommerce.all_sessions. More importantly, it already stores field values like products, pages, and transactions natively as ARRAYs.

Solution:



Before you can query REPEATED fields (arrays) normally, you must first break the arrays back into rows.

For example, the array for hits.page.pageTitle is stored currently as a single row like: ['homepage','product page','checkout'] and it needs to be:

['homepage',

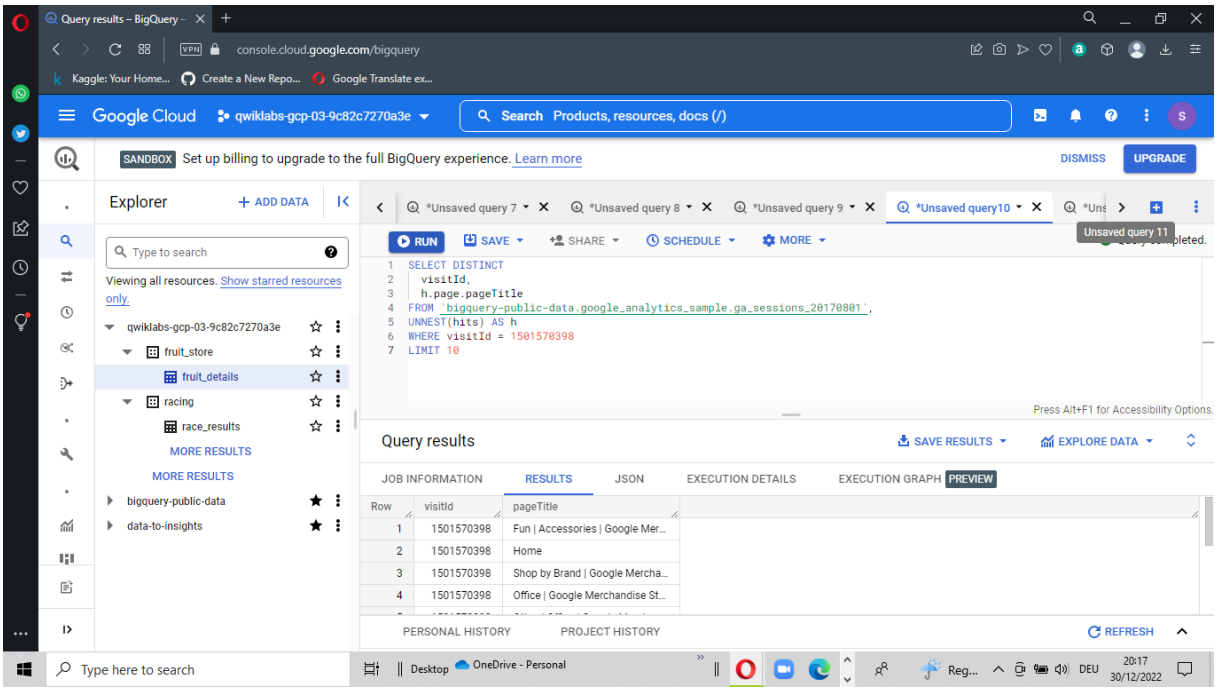
'product page',

'checkout']

How do you do that with SQL?

Answer: Use the UNNEST() function on your array field:

Solution:



Task 5. Introduction to STRUCTs

You may have wondered why the field alias hit.page.pageTitle looks like three fields in one separated by periods. Just as ARRAY values give you the flexibility to *go deep* into the granularity of your fields, another data type allows you to *go wide* in your schema by grouping related fields together. That SQL data type is the STRUCT data type.
The easiest way to think about a STRUCT is to consider it conceptually like a separate table that is already pre-joined into your main table.

A STRUCT can have:

- One or many fields in it
- The same or different data types for each field
- It's own alias

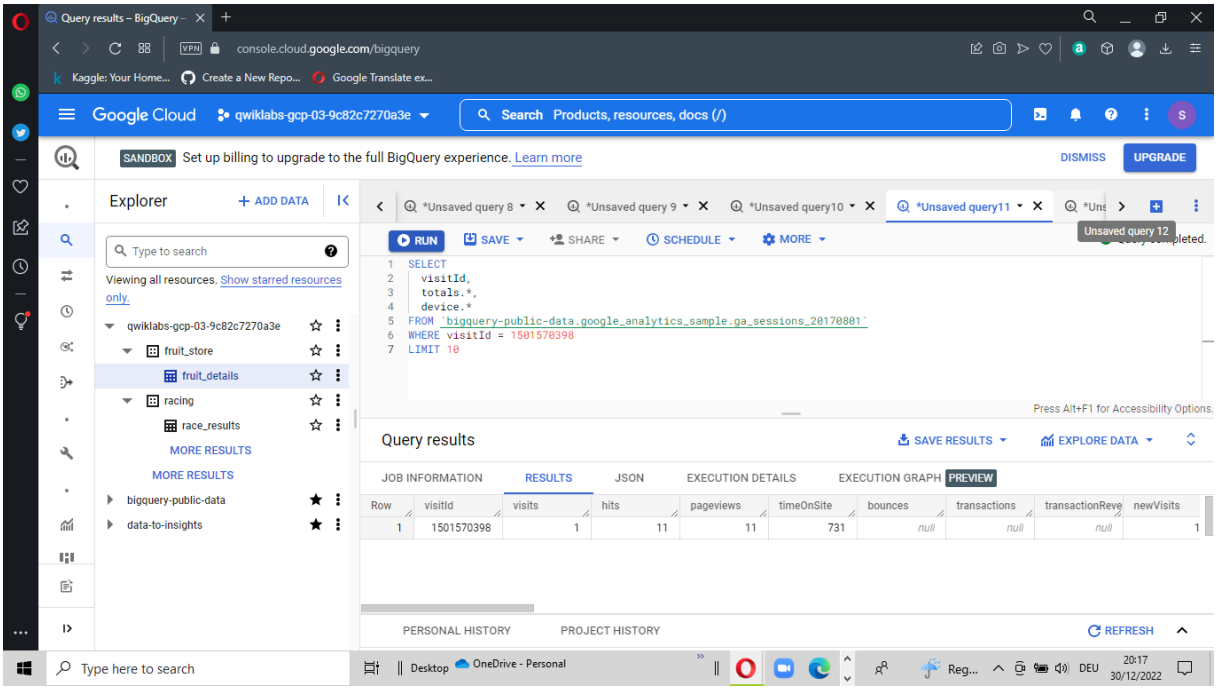
Explore a dataset with STRUCTs

1. To open the **bigquery-public-data** dataset, click **+Add Data** > **Pin a project** > **Enter Project Name**, then write the bigquery-public-data name.
2. Click **Pin**.

The bigquery-public-data project is listed in the Explorer section.

3. Open **bigquery-public-data**.
4. Find and open **google_analytics_sample** dataset.
5. Click the **ga_sessions** table.
6. Start scrolling through the schema and answer the following question by using the find feature of your browser (i.e. CTRL + F).

Solution:



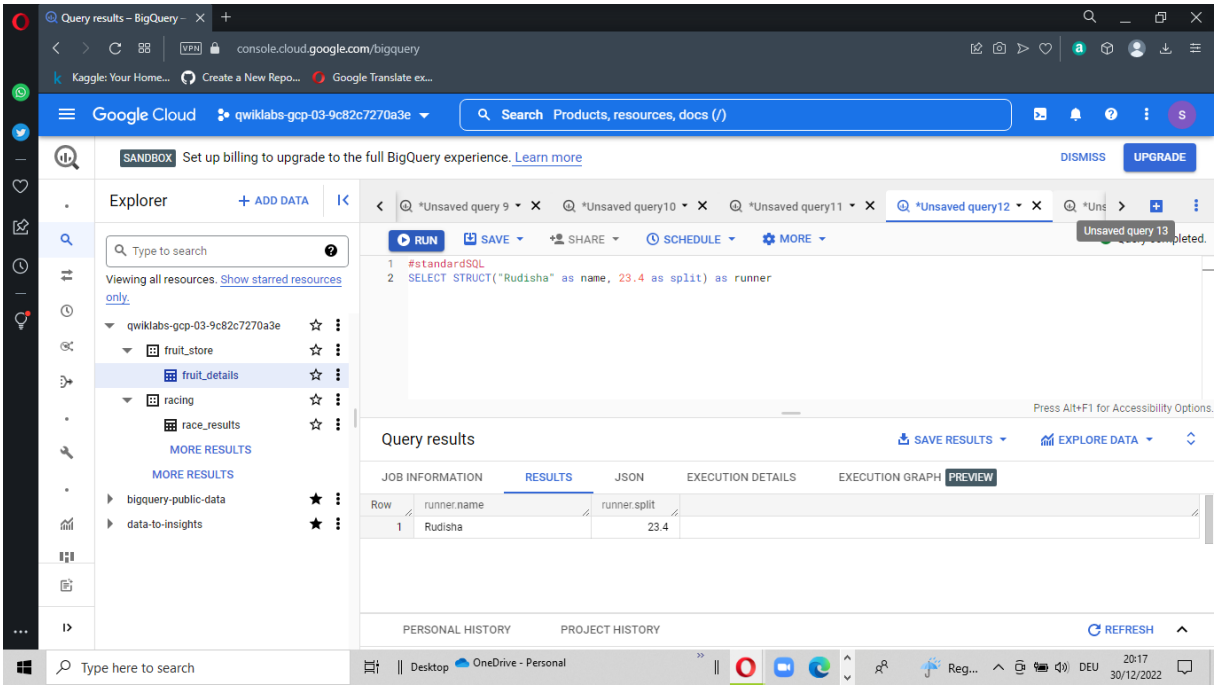
As you can imagine, there is an incredible amount of website session data stored for a modern ecommerce website.

The main advantage of having 32 STRUCTs in a single table is it allows you to run queries like this one without having to do any JOINs:

Task 6. Practice with STRUCTs and arrays:

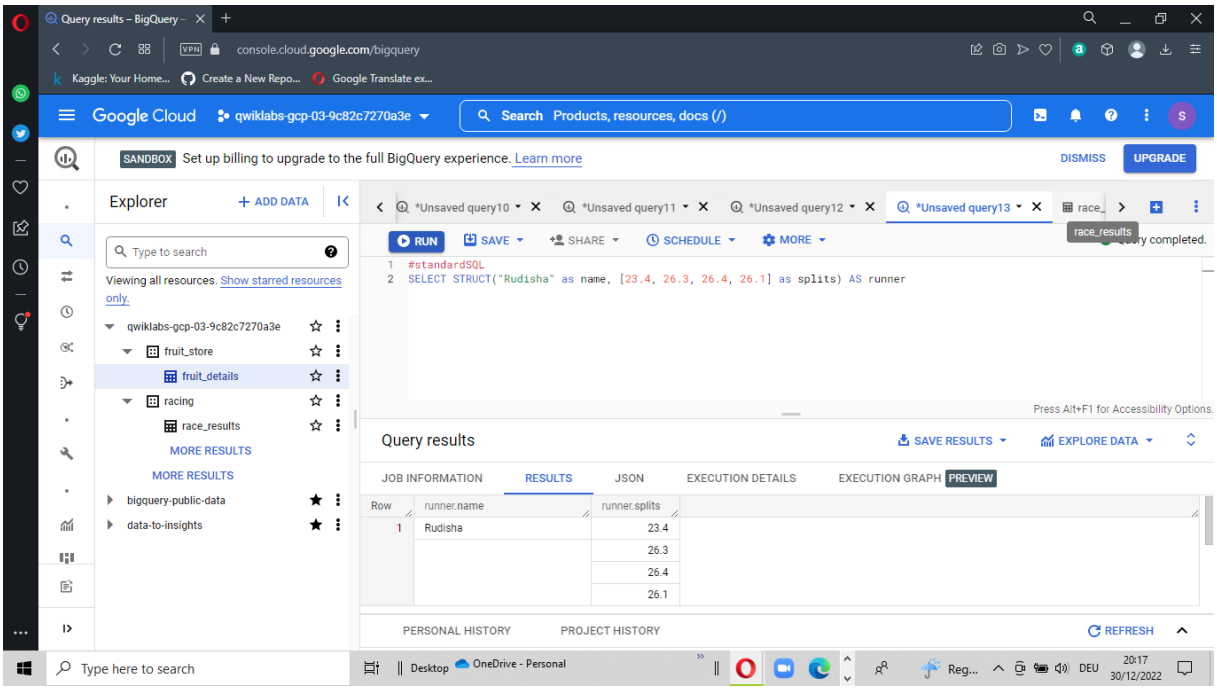
The next dataset will be lap times of runners around the track. Each lap will be called a "split".

1. With this query, try out the STRUCT syntax and note the different field types within the struct container:



With an array of course!

Solution:

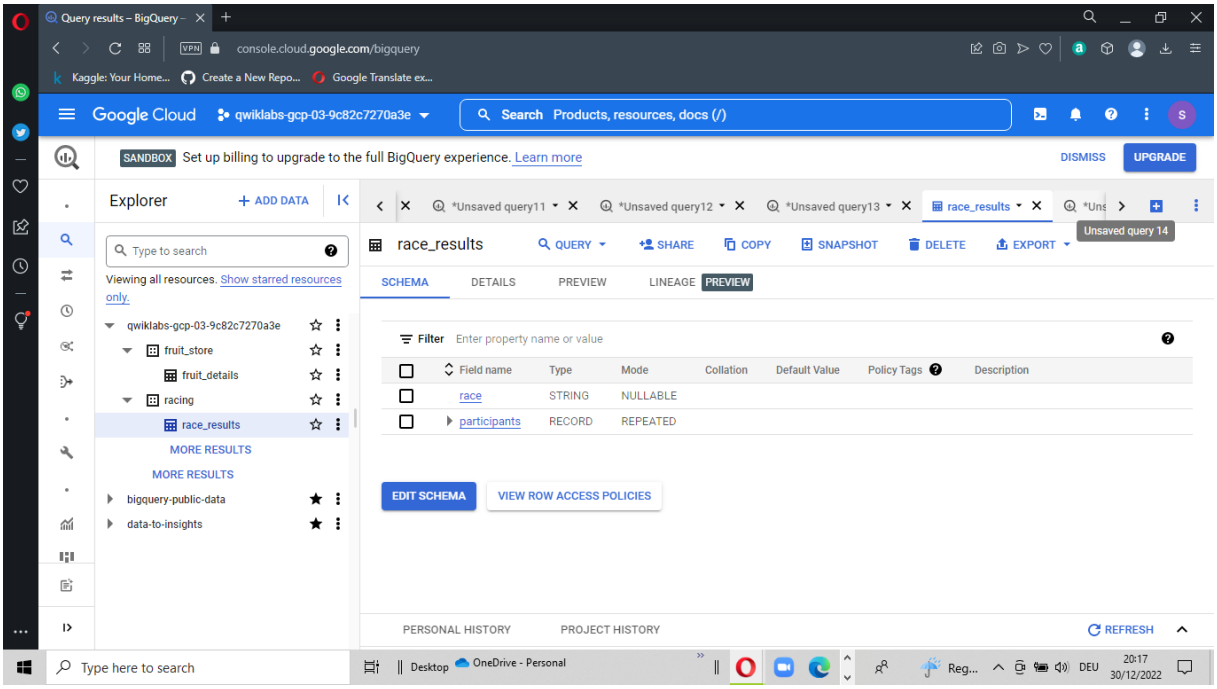


Practice ingesting JSON data

1. Create a new dataset titled racing.
2. Create a new table titled **race_results**.
3. Click on racing dataset and click Create table.

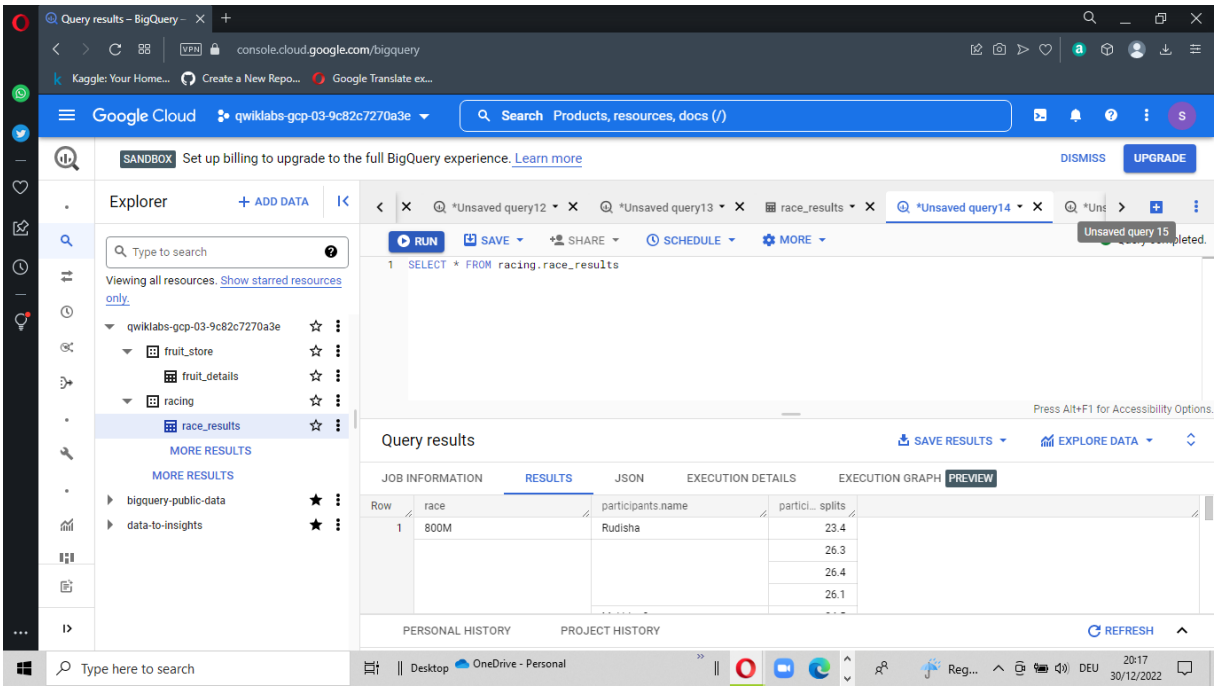
- **Source**: select **Google Cloud Storage** under **Create table from** dropdown.
- **Select file from Cloud Storage bucket**: data-insights-course/labs/optimizing-for-performance/race_results.json
- **File format**: JSONL (Newline delimited JSON)
- In **Schema**, click on **Edit as text** slider and add the following:

Solution:



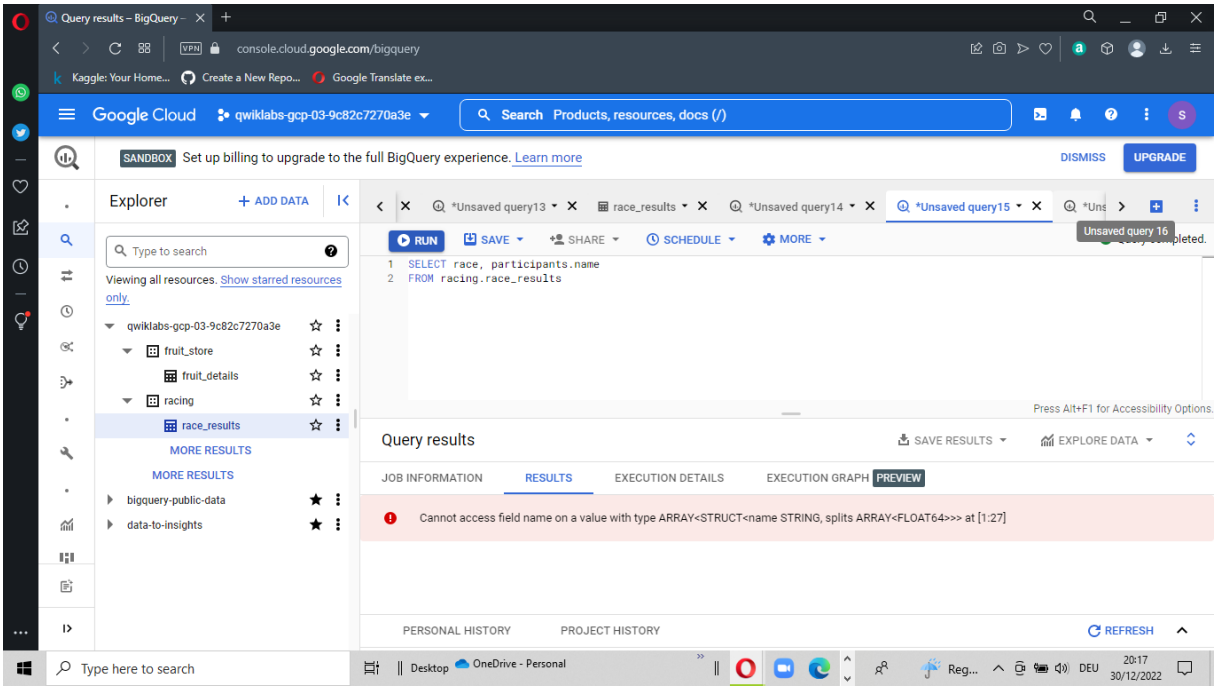
Practice querying nested and repeated fields

1. Let's see all of our racers for the 800 Meter race:



What if you wanted to list the name of each runner and the type of race?

Solution:



In traditional relational SQL, if you had a races table and a participants table what would you do to get information from both tables? You would JOIN them together. Here the participant STRUCT (which is conceptually very similar to a table) is already part of your races table but is not yet correlated correctly with your non-STRUCT field "race".

Can you think of what two words SQL command you would use to correlate the 800M race with each of the racers in the first table?

Answer: CROSS JOIN

Solution:

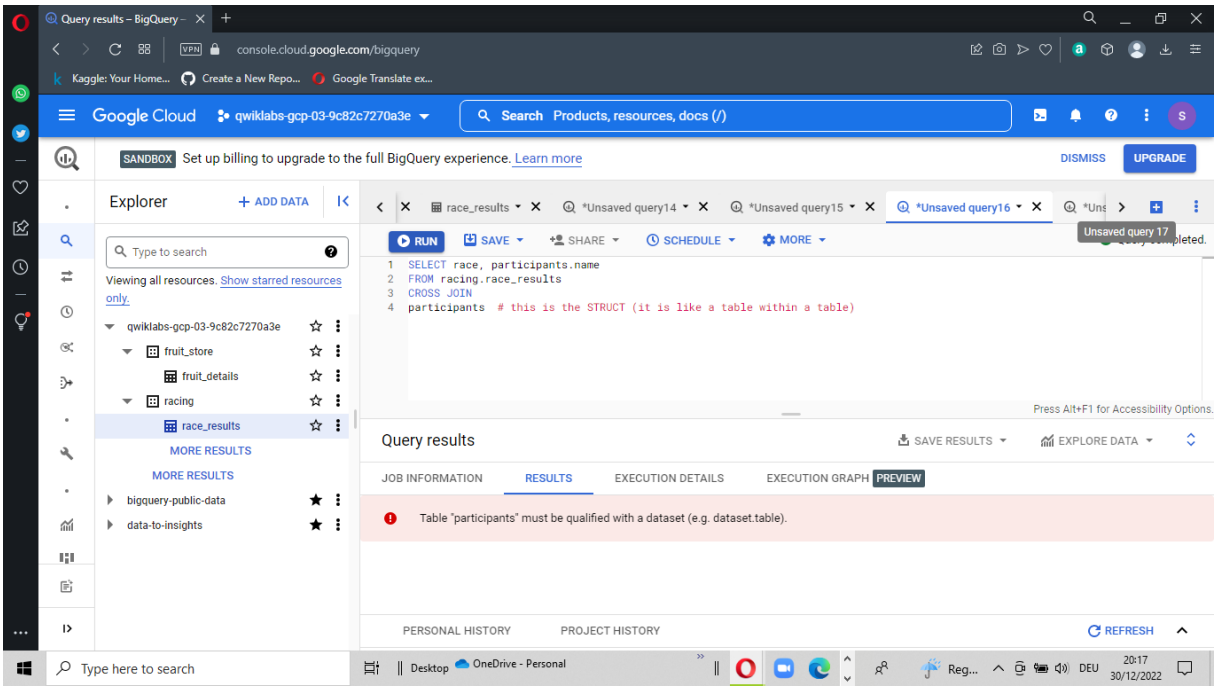
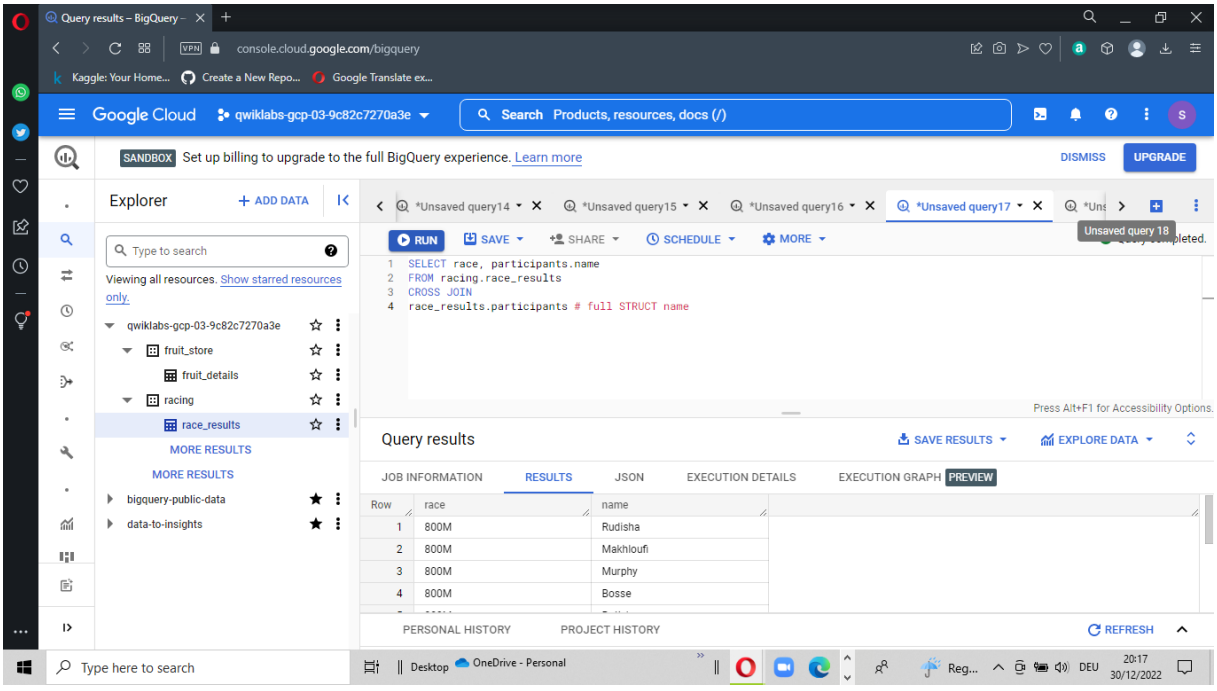


Table name "participants" missing dataset while no default dataset is set in the request.

Even though the participants STRUCT is like a table, it is still technically a field in the racing.race_results table.

Add the dataset name to the query:

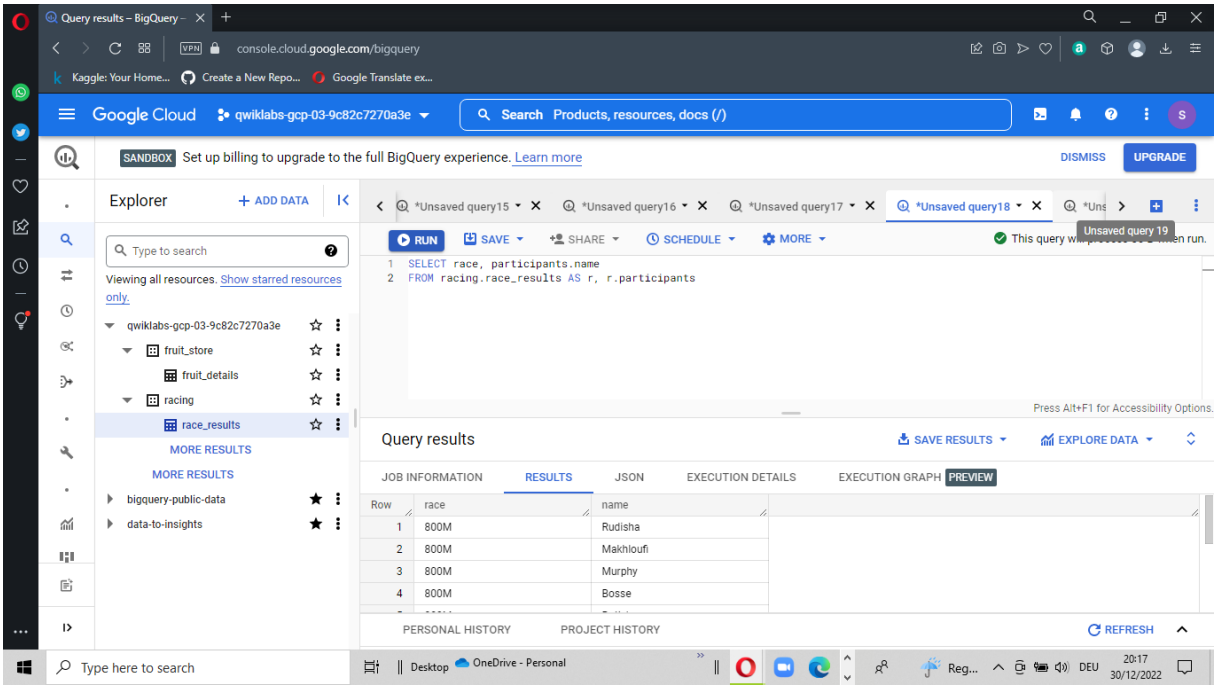
Solution:



simplify the last query by:

- Adding an alias for the original table
- Replacing the words "CROSS JOIN" with a comma (a comma implicitly cross joins)

This will give you the same query result:



If you have more than one race type (800M, 100M, 200M), wouldn't a CROSS JOIN just associate every racer name with every possible race like a cartesian product?

Answer: No. This is a *correlated* cross join which only unpacks the elements associated with a single row. For a greater discussion, see working with ARRAYs and STRUCTs.

Recap of STRUCTs:

- A SQL STRUCT is simply a container of other data fields which can be of different data types. The word struct means data structure. Recall the example from earlier: STRUCT(``"Rudisha" as name, [23.4, 26.3, 26.4, 26.1] as splits``)`` AS runner
- STRUCTs are given an alias (like runner above) and can conceptually be thought of as a table inside of your main table.

- STRUCTs (and ARRAYs) must be unpacked before you can operate over their elements. Wrap an UNNEST() around the name of the struct itself or the struct field that is an array in order to unpack and flatten it.

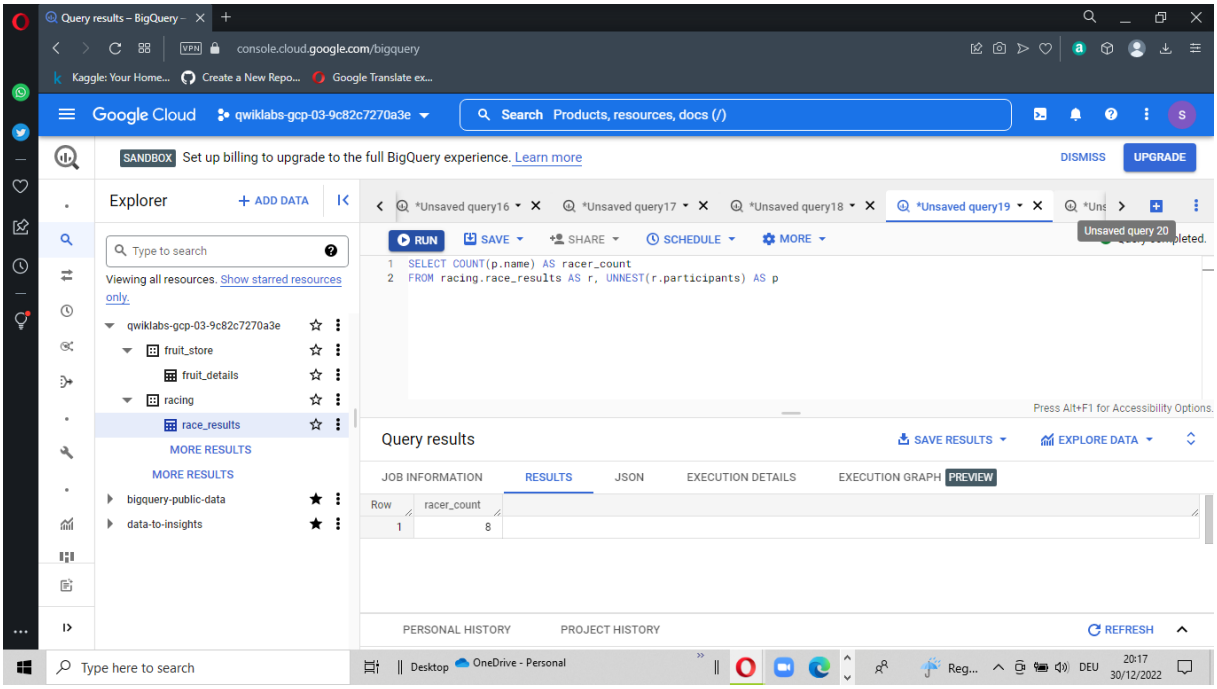
Task 7. Lab question: STRUCT()

Answer the below questions using the racing.race_results table you created previously.

Task: Write a query to COUNT how many racers were there in total.

- To start, use the below partially written query:

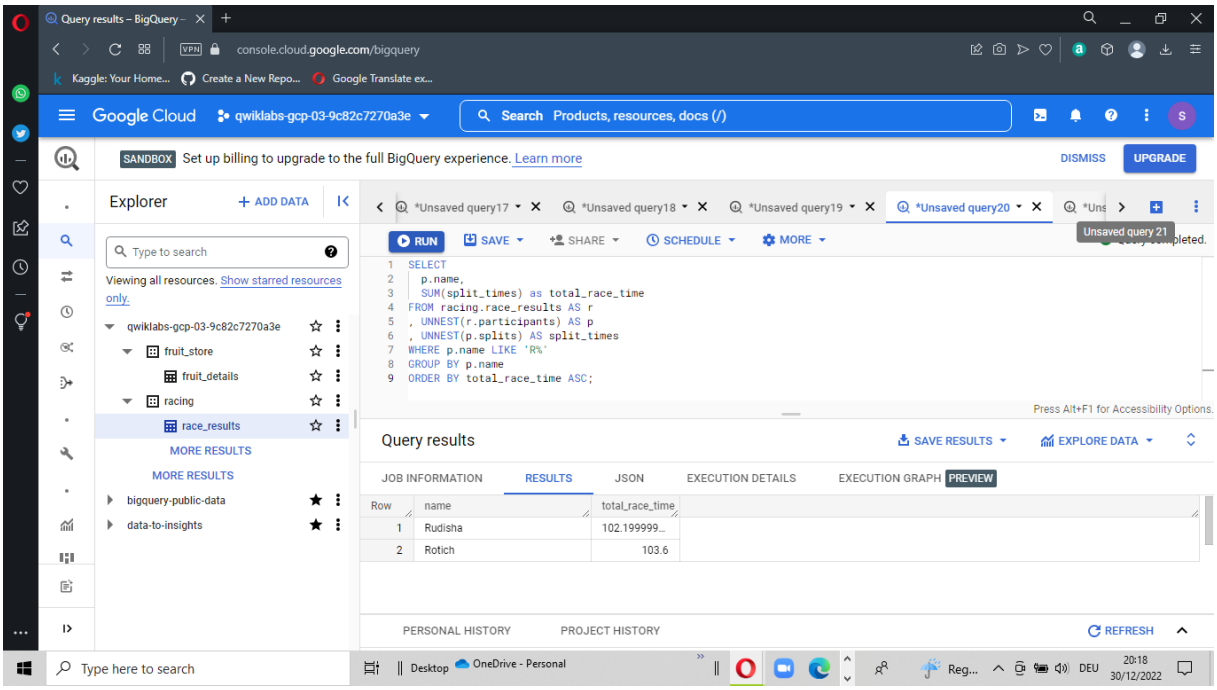
Solution:



Task 8. Lab question: Unpacking arrays with UNNEST( )

Write a query that will list the total race time for racers whose names begin with R. Order the results with the fastest total time first. Use the UNNEST() operator and start with the partially written query below.

Solution:



Task 9. Filtering within array values

You happened to see that the fastest lap time recorded for the 800 M race was 23.2 seconds, but you did not see which runner ran that particular lap. Create a query that returns that result.

Solution: