Troubleshooting and Solving Data Join Pitfalls

Joining data tables can provide meaningful insight into your dataset. However, when you join your data there are common pitfalls that could corrupt your results. This lab focuses on avoiding those pitfalls. Types of joins:

- *Cross join*: combines each row of the first dataset with each row of the second dataset, where every combination is represented in the output.
- *Inner join*: requires that key values exist in both tables for the records to appear in the results table. Records appear in the merge only if there are matches in both tables for the key values.
- *Left join*: Each row in the left table appears in the results, regardless of whether there are matches in the right table.
- *Right join*: the reverse of a left join. Each row in the right table appears in the results, regardless of whether there are matches in the left table.

Task 1. Create a new dataset to store your tables

In your BigQuery project, create a new dataset titled ecommerce.

Task 2. Pin the lab project in BigQuery

Scenario: Your team provides you with a new dataset on the inventory stock levels for each of your products for sale on your ecommerce website. You want to become familiar with the products on the website and the fields you could use to potentially join on to other datasets.

The project with the new dataset is **data-to-insights**.

1. Click **Navigation menu** ☰ > **BigQuery**.

The Welcome to BigQuery in the Cloud Console message box opens.

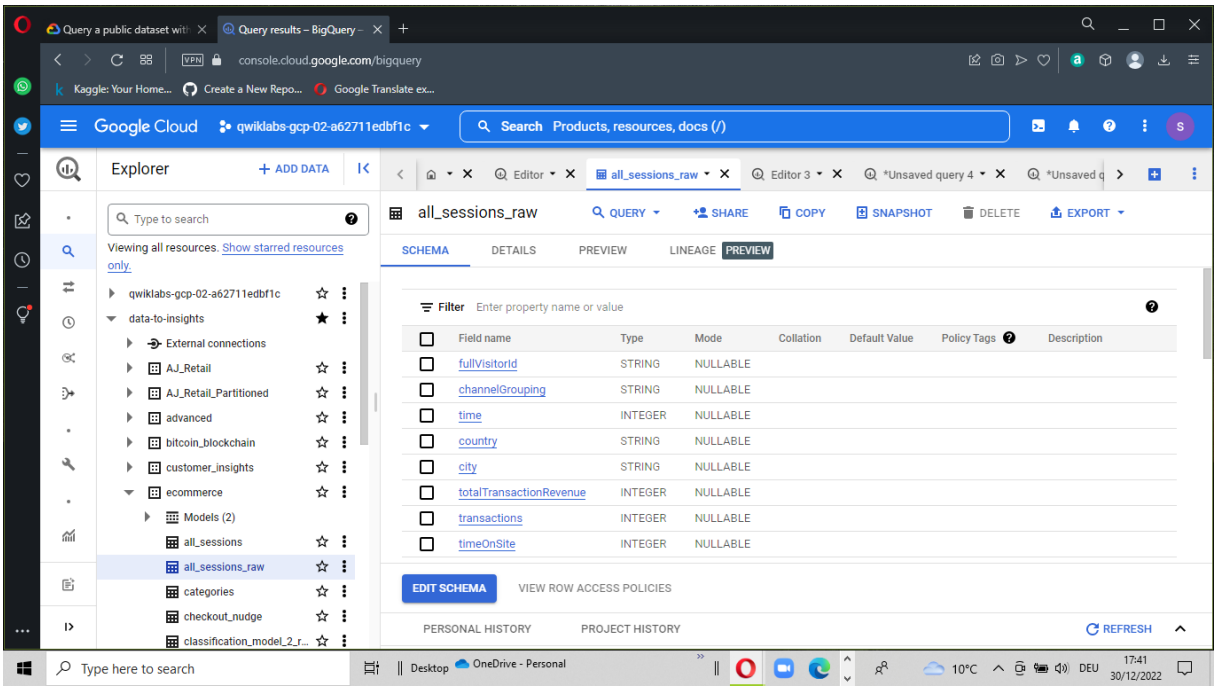
2. Click **Done**.

3. BigQuery public datasets are not displayed by default. To open the public datasets project, copy **data-to-insights**.

4. Click **Add Data** > **Star a project by name** then paste the data-to-insights name.

The data-to-insights project is listed in the Explorer section.

Solution:



Task 3. Identify a key field in your ecommerce dataset

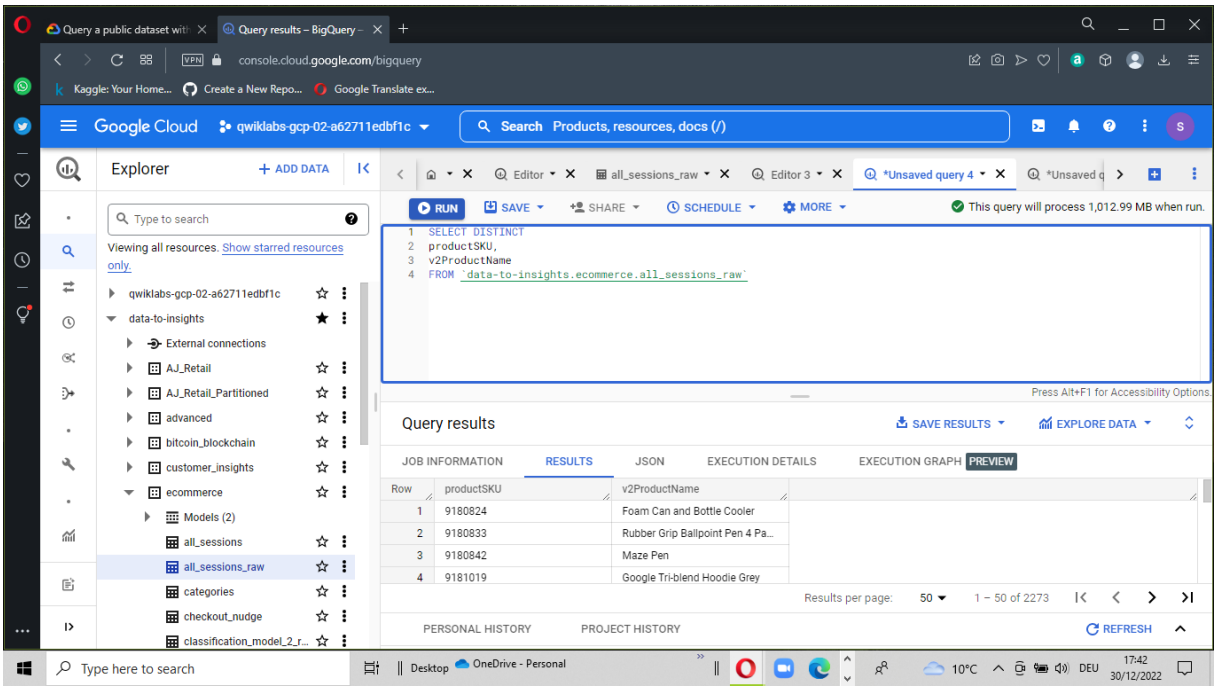
Examine the products and fields further. You want to become familiar with the products on the website and the fields you could use to potentially join on to other datasets.

Examine the records

In this section you find how many product names and product SKUs are on your website and whether either one of those fields is unique.

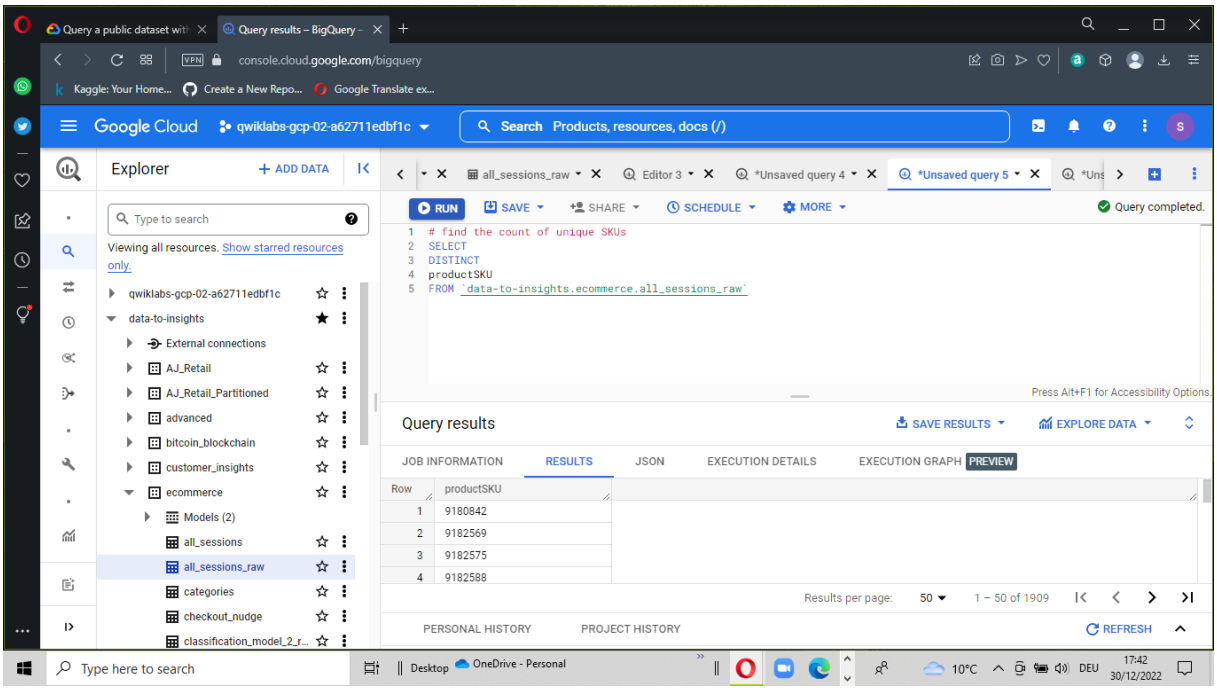
1. Find how many product names and product SKUs are on the website. **Copy and Paste** the below query in bigquery **EDITOR**:

Solution:



2. Clear the previous query and run the below query to list the number of distinct SKUs are listed using DISTINCT.

Solution:

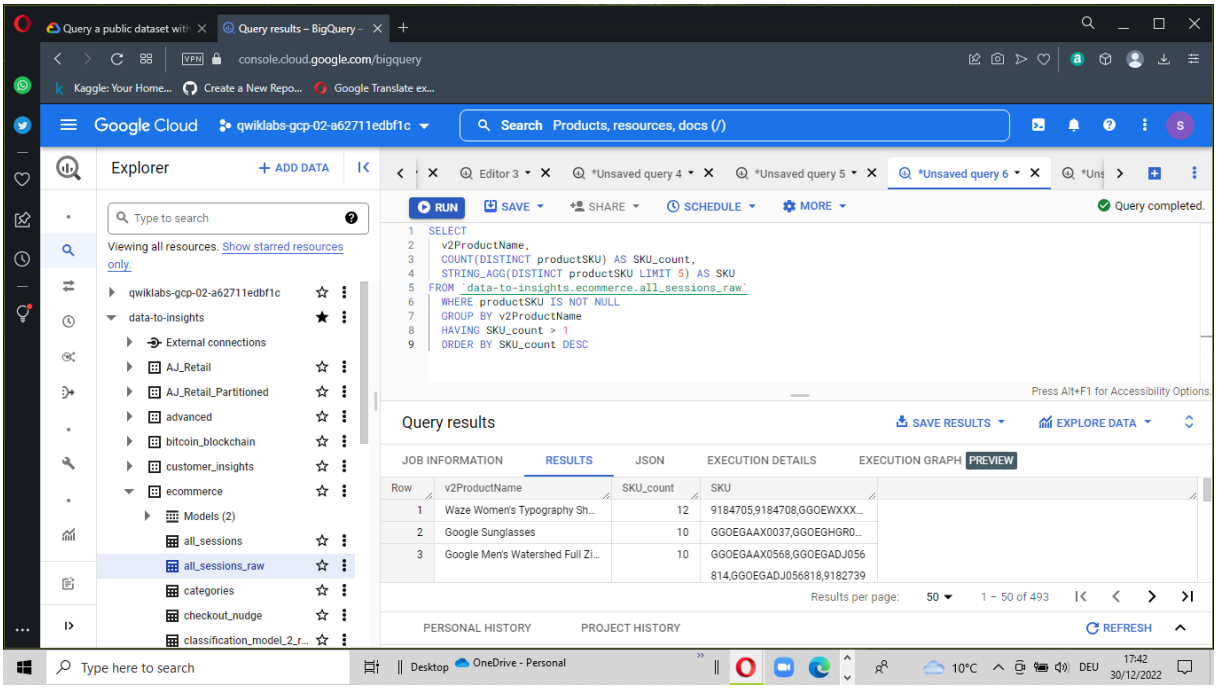


Examine the relationship between SKU & Name

Now determine which products have more than one SKU and which SKUs have more than one Product Name.

1. Clear the previous query and run the below query to determine if some product names have more than one SKU. The use of the STRING_AGG() function to aggregate all the product SKUs that are associated with one product name into comma separated values.

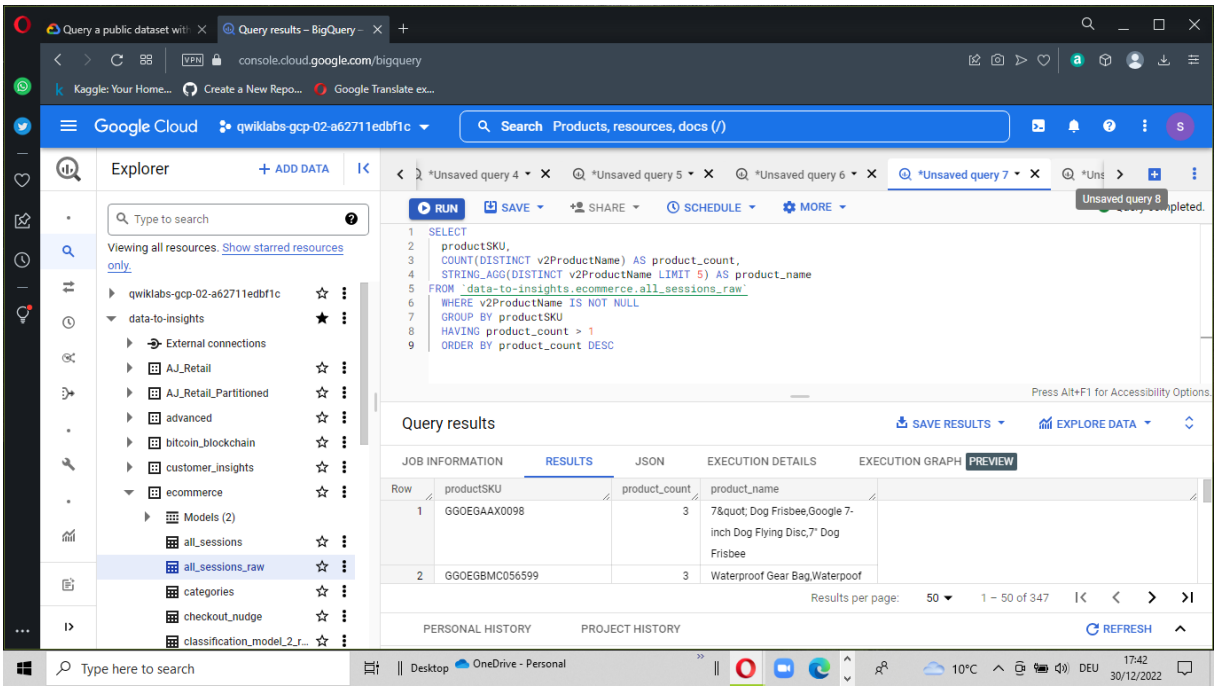
Solution:



The ecommerce website catalog shows that each product name may have multiple options (size, color) -- which are sold as separate SKUs.
So you have seen that 1 Product can have 12 SKUs. What about 1 SKU? Should it be allowed to belong to more than 1 product?

- Clear the previous query and run the below query to find out:

Solution:

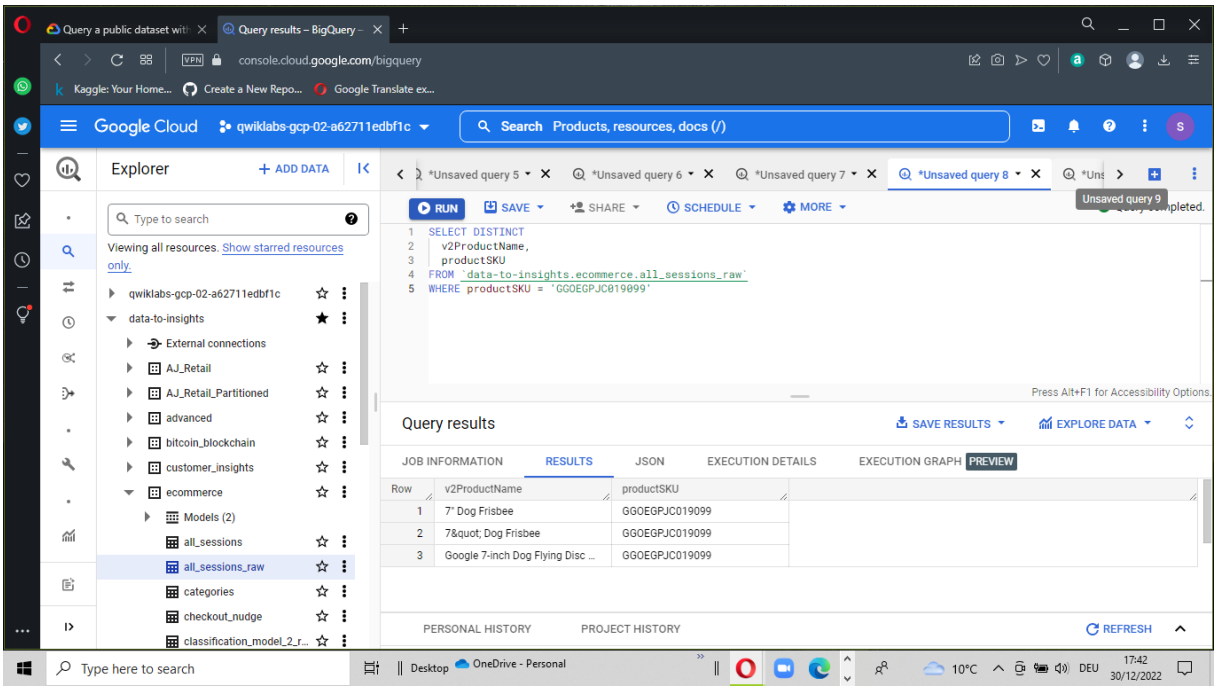


Task 4. Pitfall: non-unique key

In inventory tracking, a SKU is designed to uniquely identify one and only one product. For us, it will be the basis of your JOIN condition when you lookup information from other tables. Having a non-unique key can cause serious data issues as you will see.

1. **Write a query** to identify all the product names for the SKU 'GGOEGPJC019099'

Solution:

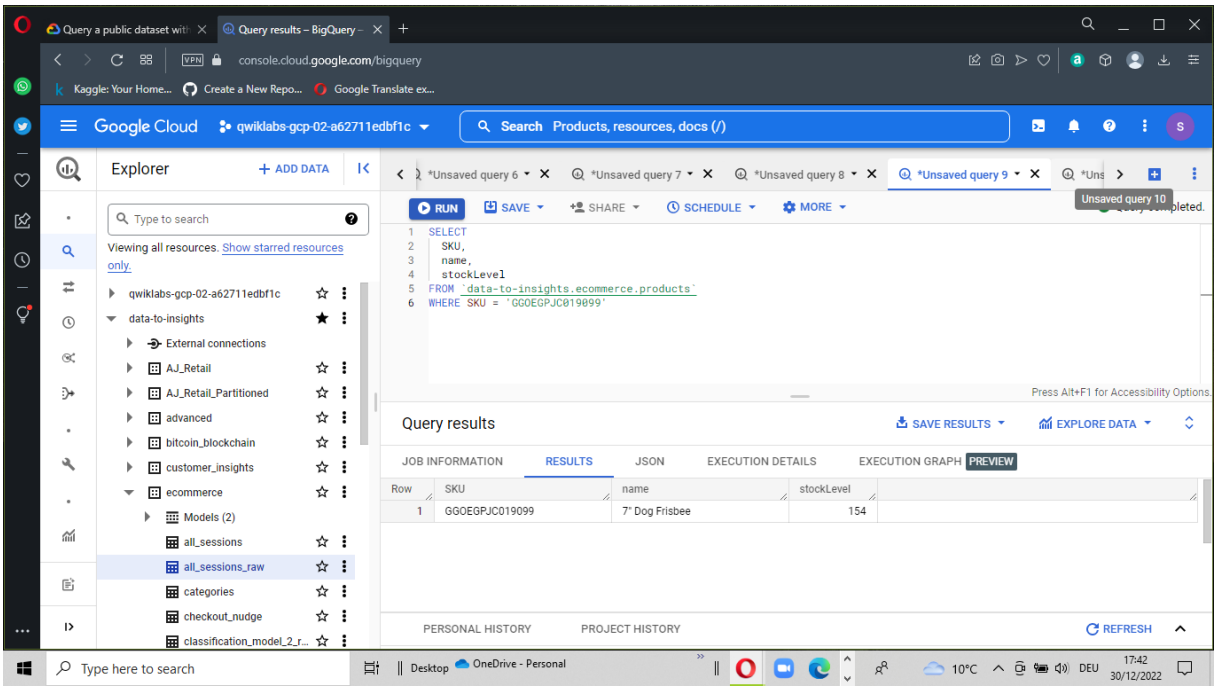


Joining website data against your product inventory list

Now see the impact of joining on a dataset with multiple products for a single SKU. First explore the product inventory dataset (the products table) to see if this SKU is unique there.

- Clear the previous query and run the below query:

Solution:

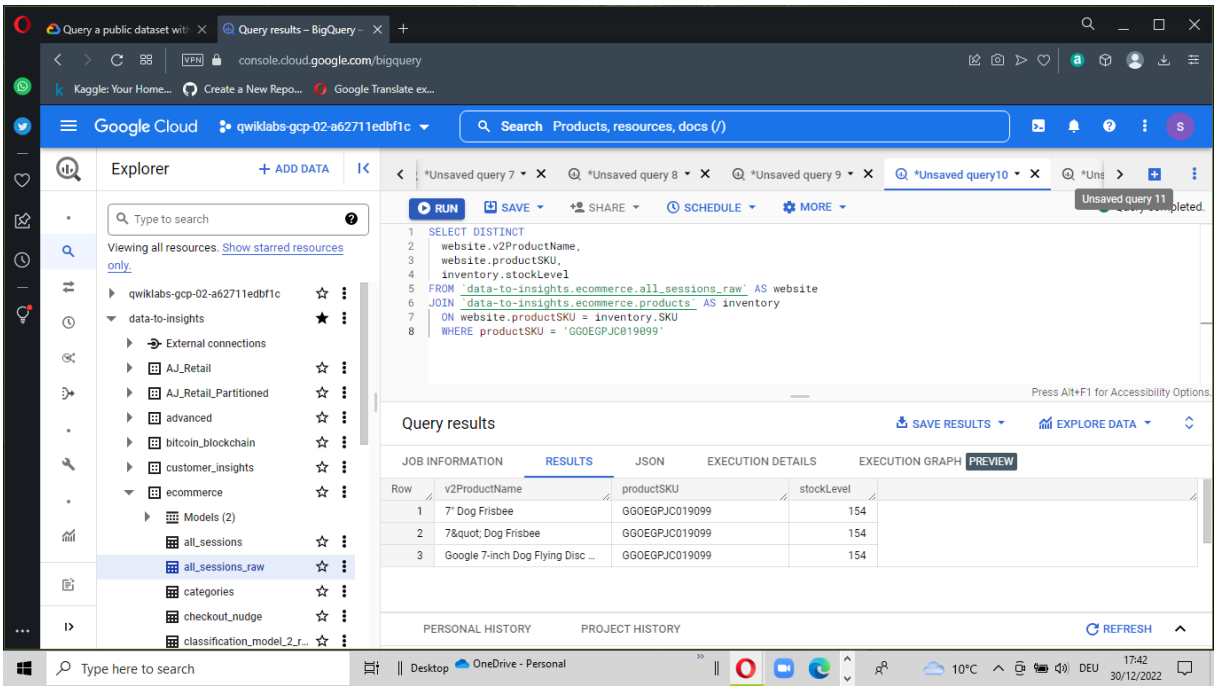


Join pitfall: Unintentional many-to-one SKU relationship

You now have two datasets: one for inventory stock level and the other for our website analytics. JOIN the inventory dataset against your website product names and SKUs so you can have the inventory stock level associated with each product for sale on the website.

1. Clear the previous query and run the below query:

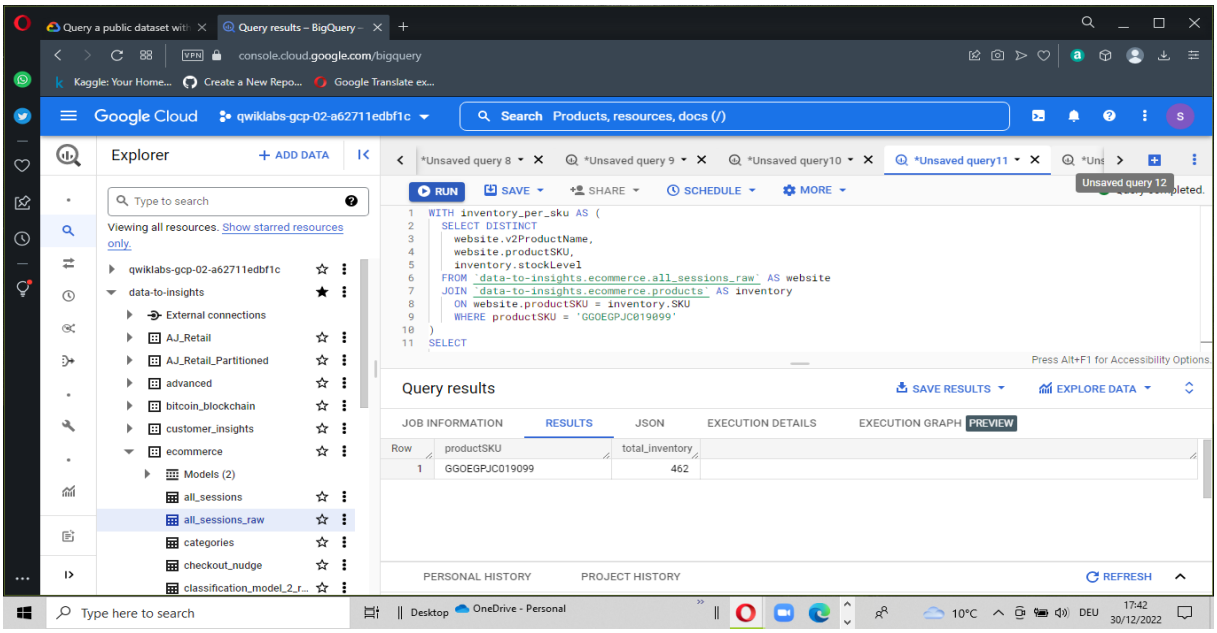
Solution:



Next, expand our previous query to simply SUM the inventory available by product.

2. Clear the previous query and run the below query:

Solution:



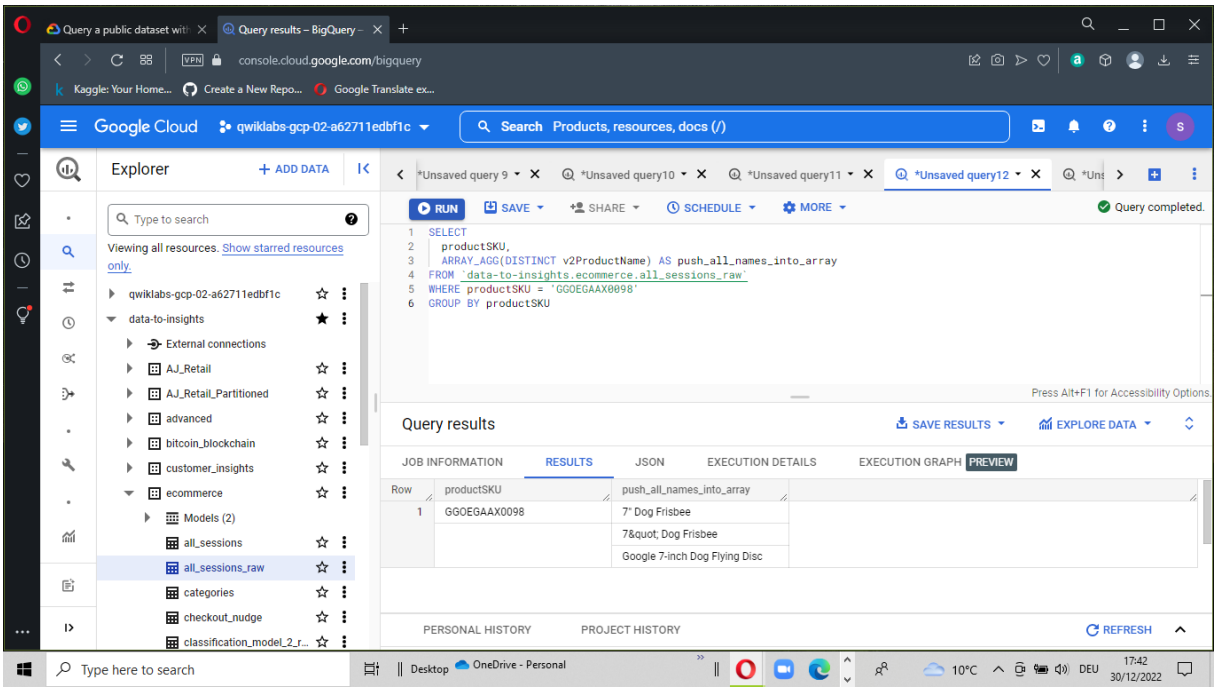
Task 5. Join pitfall solution: use distinct SKUs before joining

What are the options to solve your triple counting dilemma? First you need to only select distinct SKUs from the website before joining on other datasets.

You know that there can be more than one product name (like 7" Dog Frisbee) that can share a single SKU.

1. Gather all the possible names into an array:

Solution:



Now instead of having a row for every Product Name, you only have a row for each unique SKU.

Join pitfall: losing data records after a join

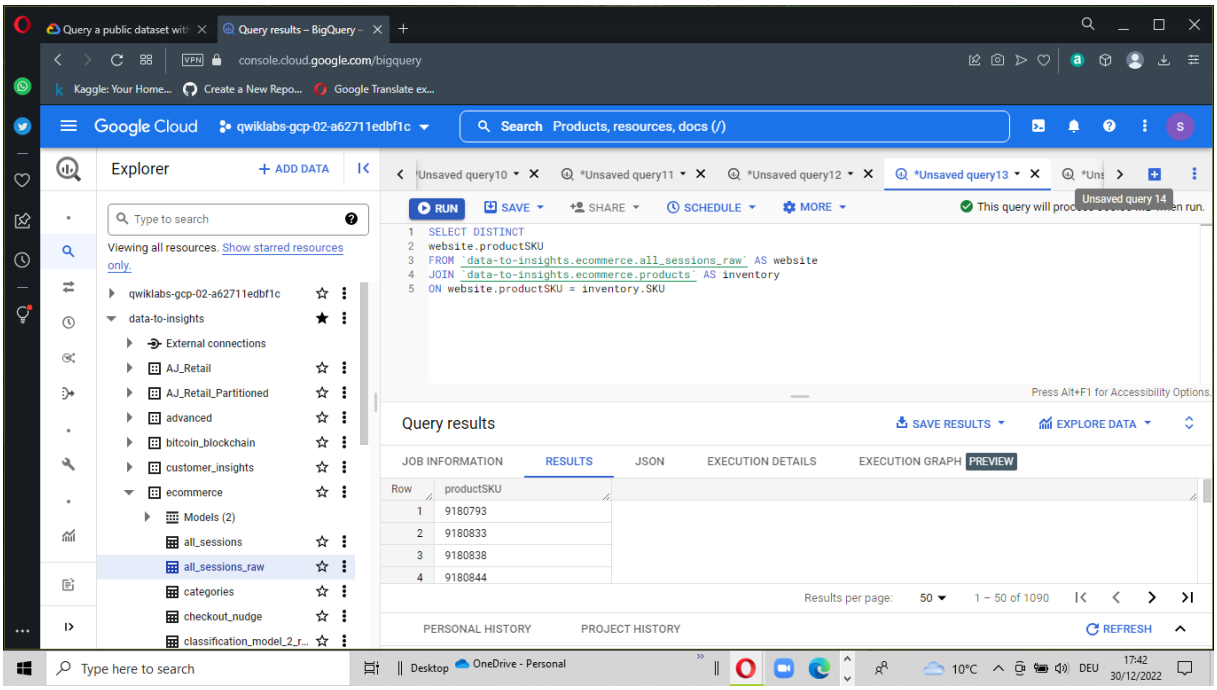
Now you're ready to join against your product inventory dataset again.

1. Clear the previous query and run the below query:

It seems 819 SKUs were lost after joining the datasets Investigate by adding more specificity in your fields (one SKU column from each dataset):

2. Clear the previous query and run the below query:

Solution:



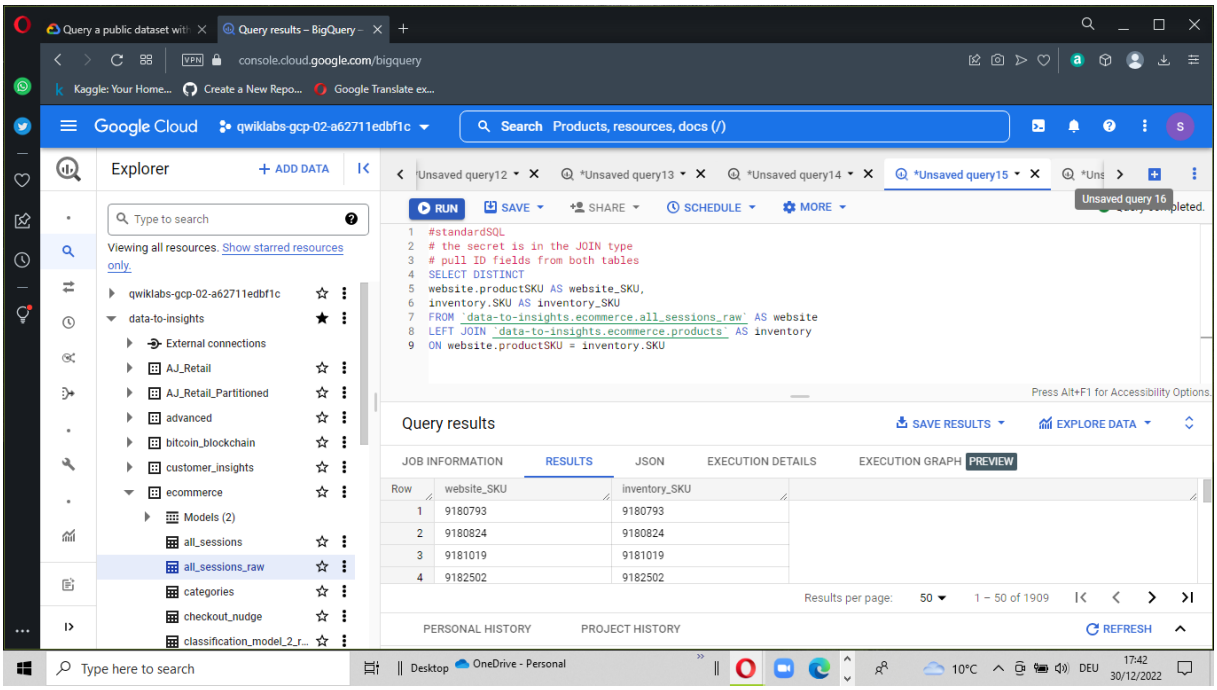
It appears the SKUs are present in both of those datasets after the join for these 1,090 records. How can you find the missing records?

Join pitfall solution: selecting the correct join type and filtering for NULL

The default JOIN type is an INNER JOIN which returns records only if there is a SKU match on both the left and the right tables that are joined.

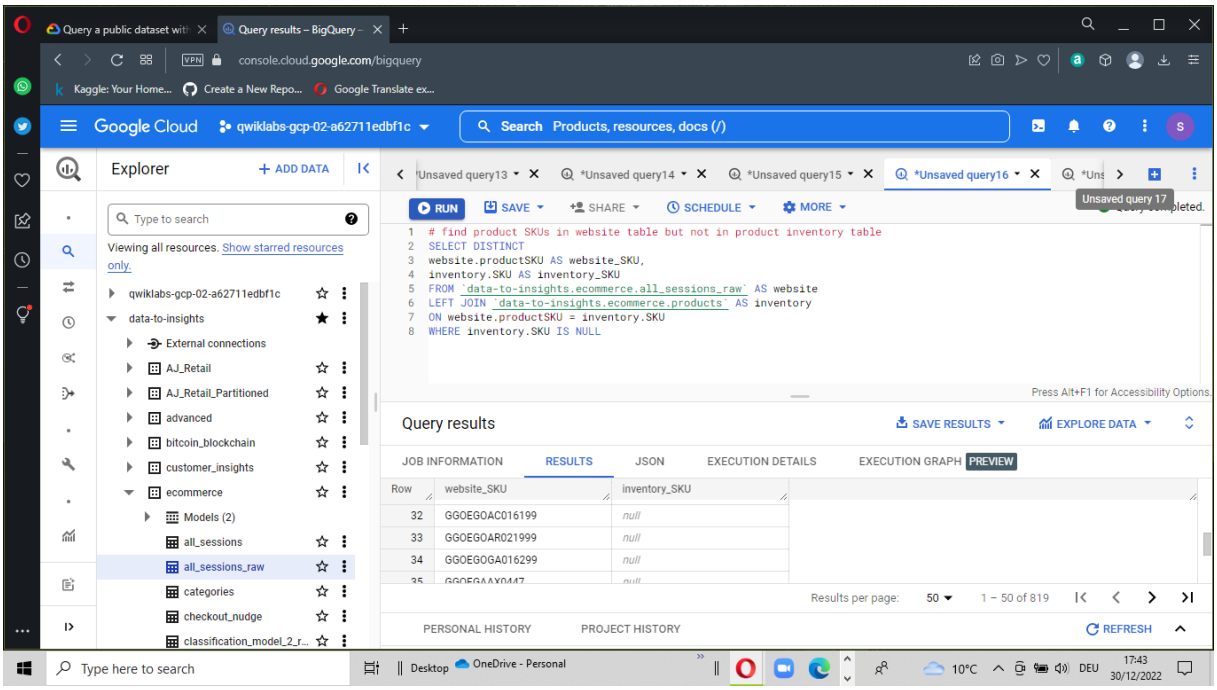
1. **Rewrite the previous query to use a different join type** to include all records from the website table, regardless of whether there is a match on a product inventory SKU record. Join type options: INNER JOIN, LEFT JOIN, RIGHT JOIN, FULL JOIN, CROSS JOIN.

Solution:



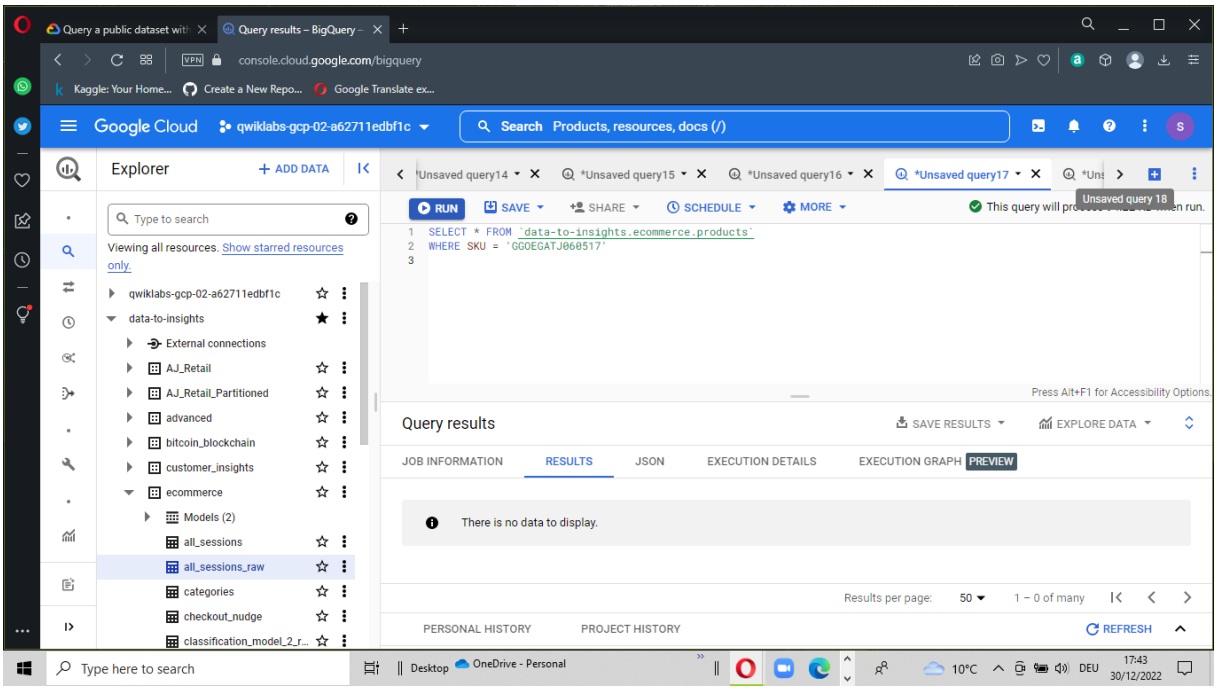
1. **Write a query** to filter on NULL values from the inventory table.

Solution:



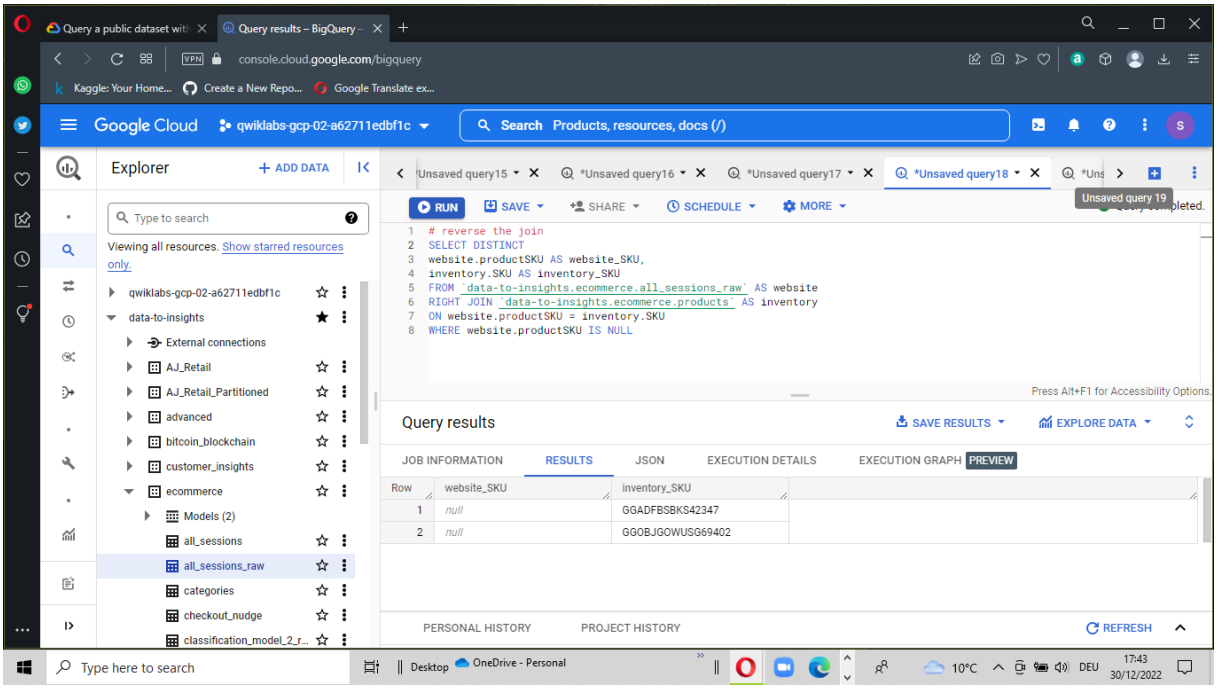
- Clear the previous query and run the below query to confirm using one of the specific SKUs from the website dataset:

Solution:



Write a query using a different join type to investigate.

Solution:

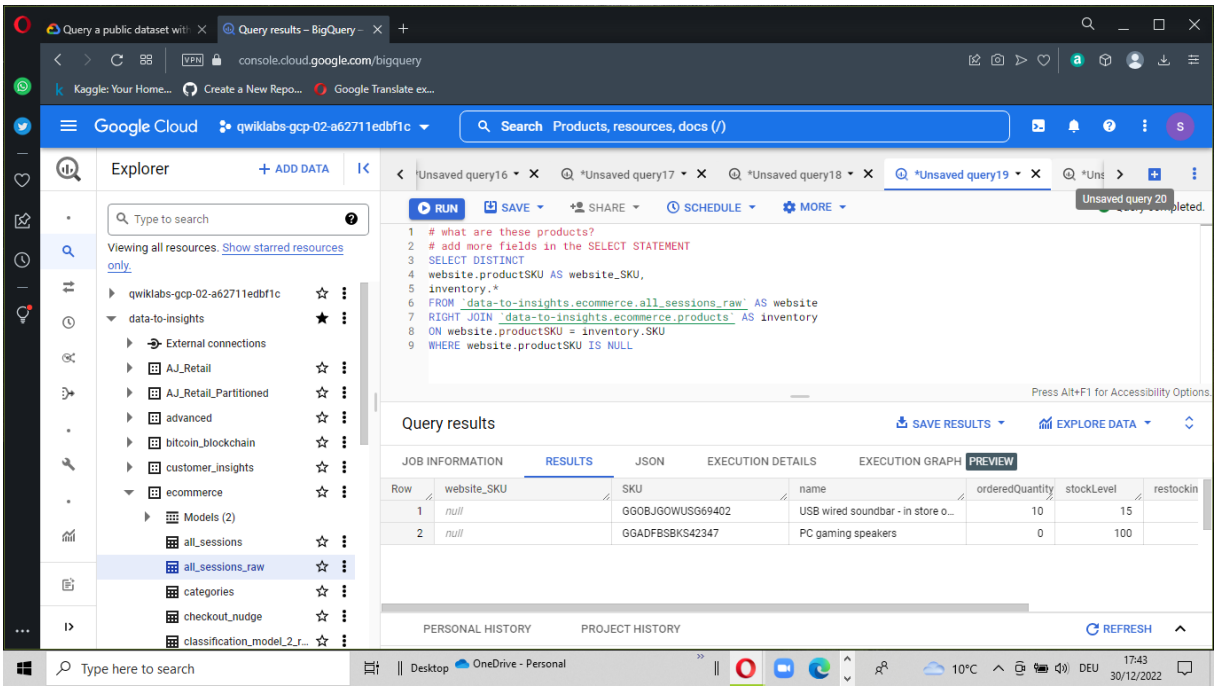


Answer: Yes. There are two product SKUs missing from the website dataset

Next, add more fields from the product inventory dataset for more details.

- Clear the previous query and run the below query:

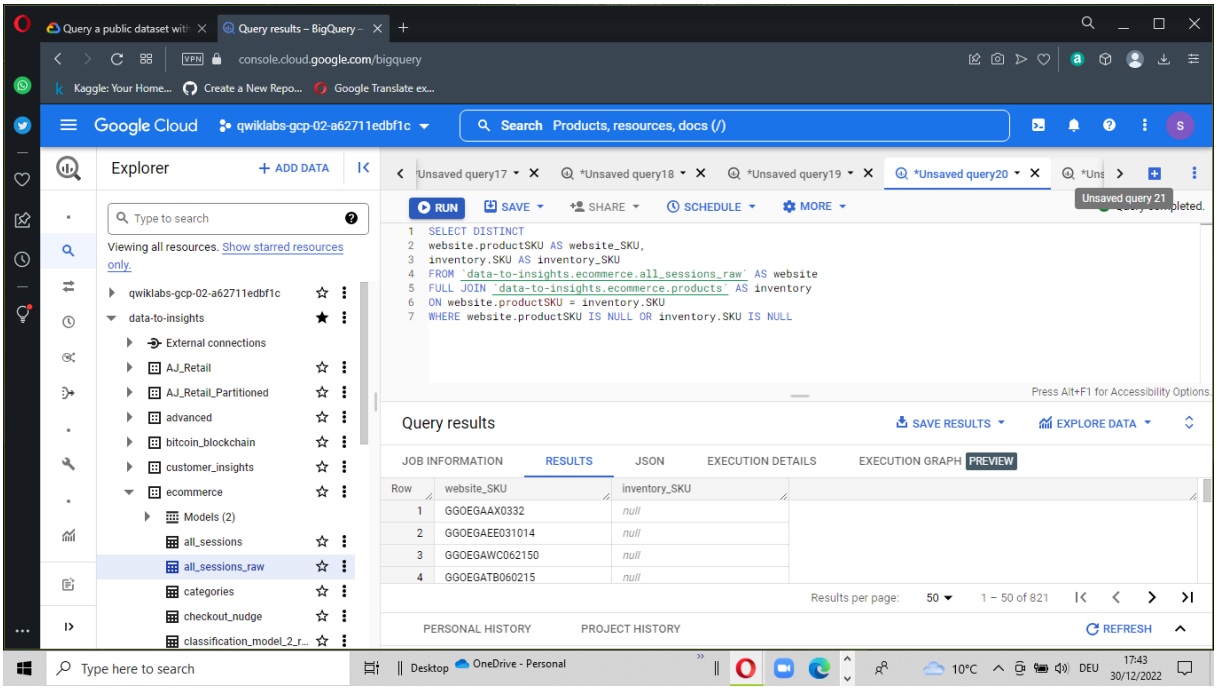
Solution:



What if you wanted one query that listed all products missing from either the website or inventory?

1. Write a query using a different join type.

Solution:



You have your 819 + 2 = 821 product SKUs.

LEFT JOIN + RIGHT JOIN = FULL JOIN which returns all records from both tables regardless of matching join keys. You then filter out where you have mismatches on either side

Join pitfall: unintentional cross join

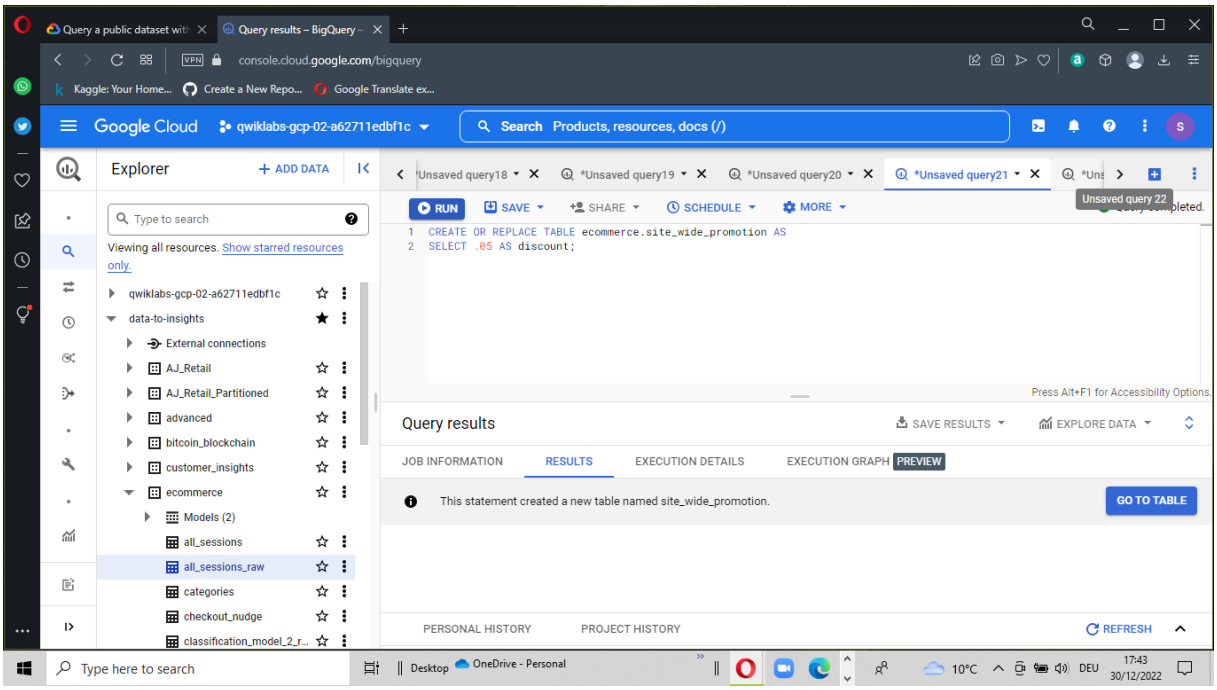
Not knowing the relationship between data table keys (1:1, 1:N, N:N) can return unexpected results and also significantly reduce query performance.

The last join type is the CROSS JOIN.

Create a new table with a site-wide discount percent that you want applied across products in the Clearance category.

1. Clear the previous query and run the below query:

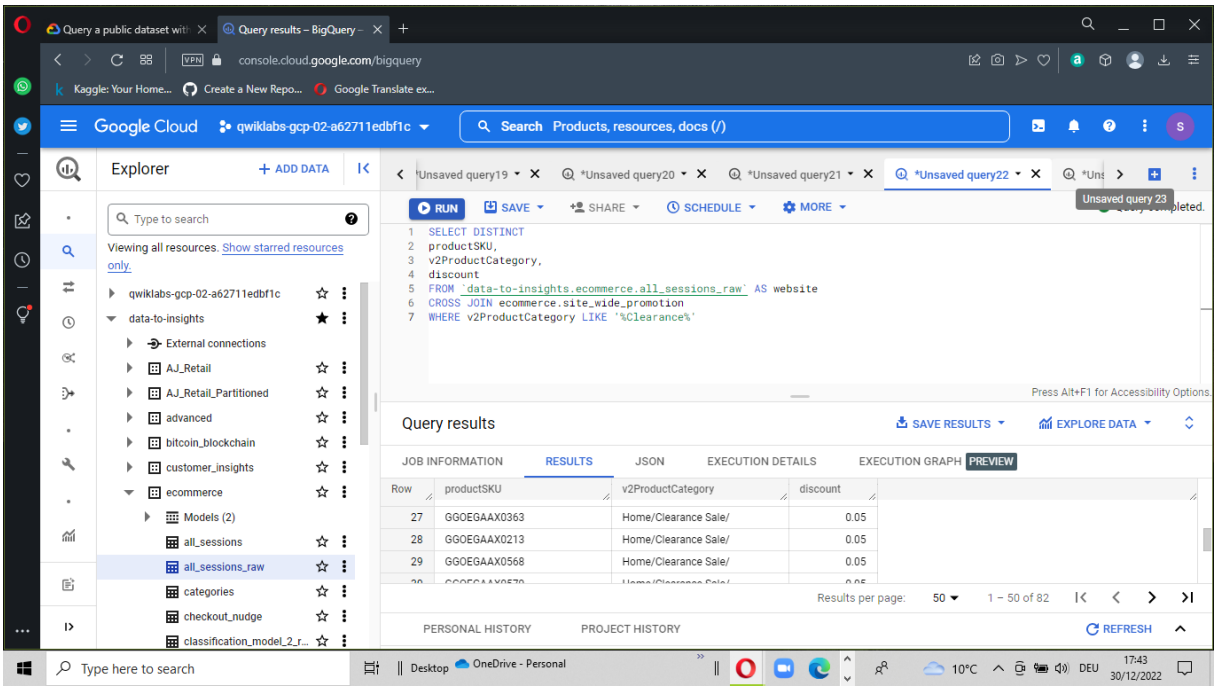
Solution:



In the left pane, site_wide_promotion is now listed in the Resource section under your project and dataset.

2. Clear the previous query and run the below query to find out how many products are in clearance:

Solution:

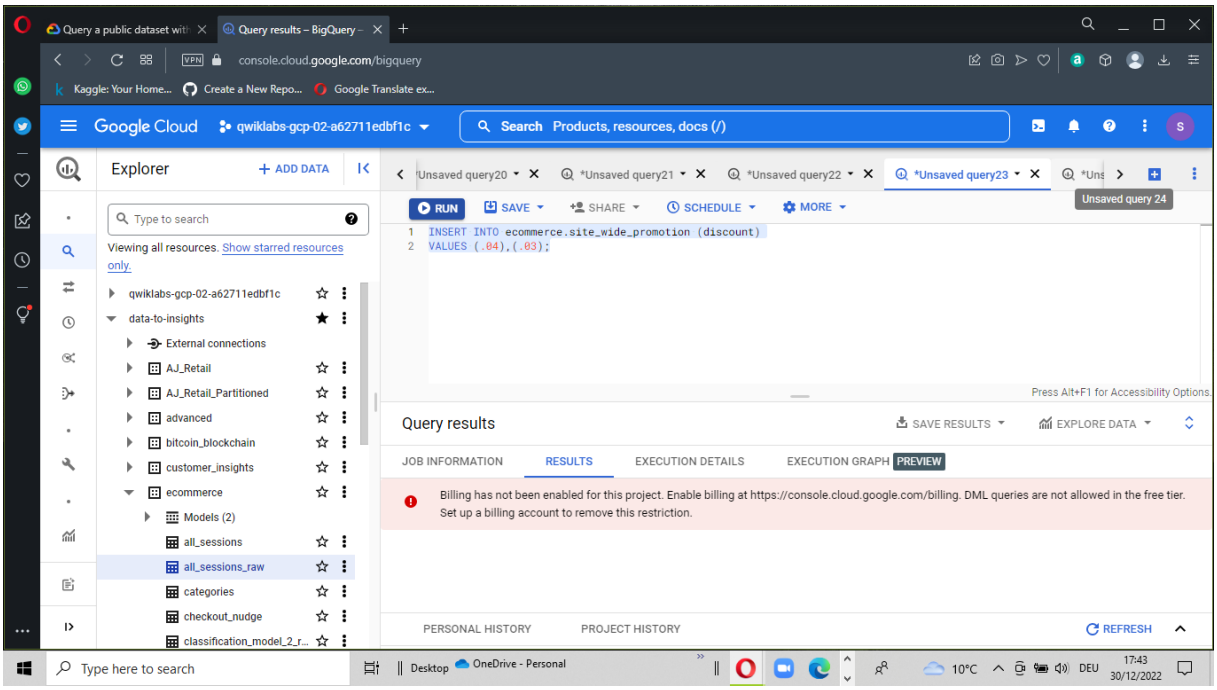


> **Note:** For a CROSS JOIN you will notice there is no join condition (e.g. ON or USING). The field is simply multiplied against the first dataset or .05 discount across all items.

See the impact of unintentionally adding more than one record in the discount table.

3. Clear the previous query and run the below query to insert two more records into the promotion table:

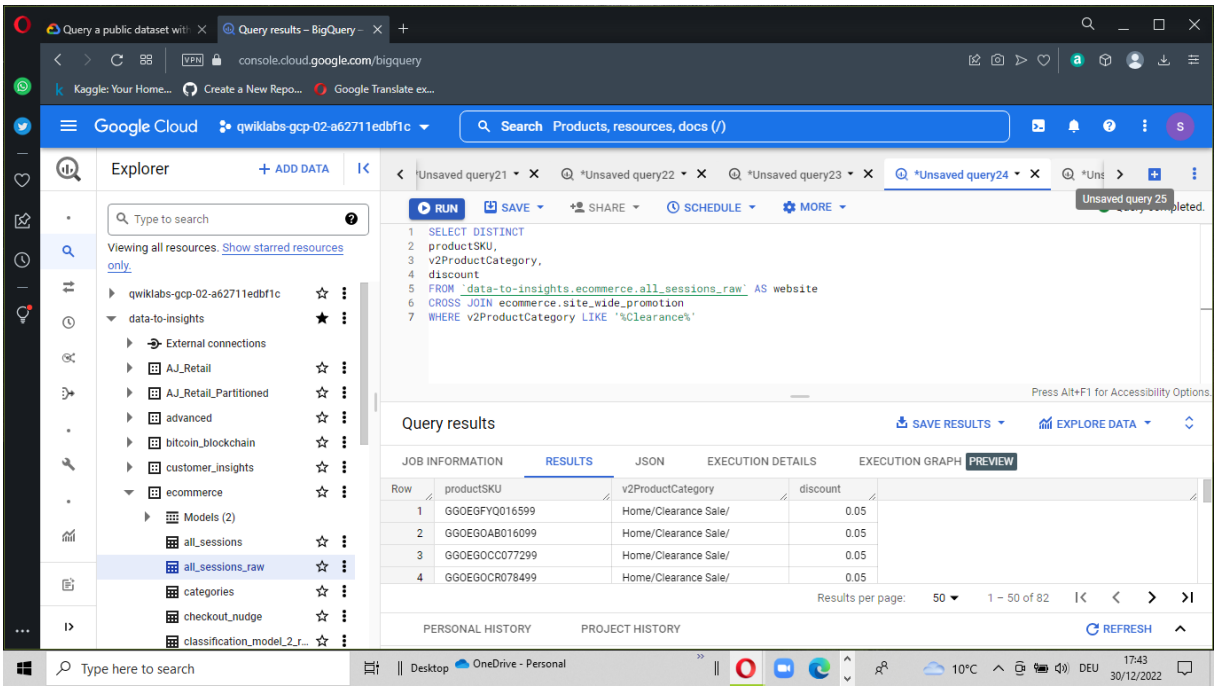
Solution:



Next, view the data values in the promotion table.

4. Clear the previous query and run the below query

Solution:



How many products are returned?

Answer: Instead of 82, you now have 246 returned which is more records than your original table started with.

Now investigate the underlying cause by examining one product SKU.

5. Clear the previous query and run the below query

Solution: