Troubleshooting Common SQL Errors with BigQuery v1.5 | Google Cloud Skills Boost

Qwiklabs : 10-13 minutes

Overview

BigQuery is Google's fully managed, NoOps, low cost analytics database. With BigQuery you can query terabytes and terabytes of data without having any infrastructure to manage or needing a database administrator. BigQuery uses SQL, and you can take advantage of the pay-as-you-go model. BigQuery allows you to focus on analyzing data to find meaningful insights.

A newly available ecommerce dataset that has millions of Google Analytics records for the Google Merchandise Store has been loaded into BigQuery. You have a copy of that dataset for this lab and will explore the available fields and row for insights.

This lab steps you through the logic of troubleshooting queries. It provides activities within the context of a real-world scenario. Throughout the lab, imagine you're working with a new data analyst on your team, and they've provided you with their queries below to answer some questions on your ecommerce dataset. Use the answers to fix their queries to get a meaningful result.

What you'll do

In this lab, learn how to perform the following tasks:

- Query the data-to-insights public dataset
- Use the BigQuery Query editor to troubleshoot common SQL errors
- Use the Query Validator
- Troubleshoot syntax and logical SQL errors

Setup and requirements

For each lab, you get a new Google Cloud project and set of resources for a fixed time at no cost.

1. Sign in to Qwiklabs using an **incognito window**.
2. Note the lab's access time (for example, `1:15:00`), and make sure you can finish within that time. There is no pause feature. You can restart if needed, but you have to start at the beginning.
3. When ready, click **Start lab**.
4. Note your lab credentials (**Username** and **Password**). You will use them to sign in to the Google Cloud Console.
5. Click **Open Google Console**.

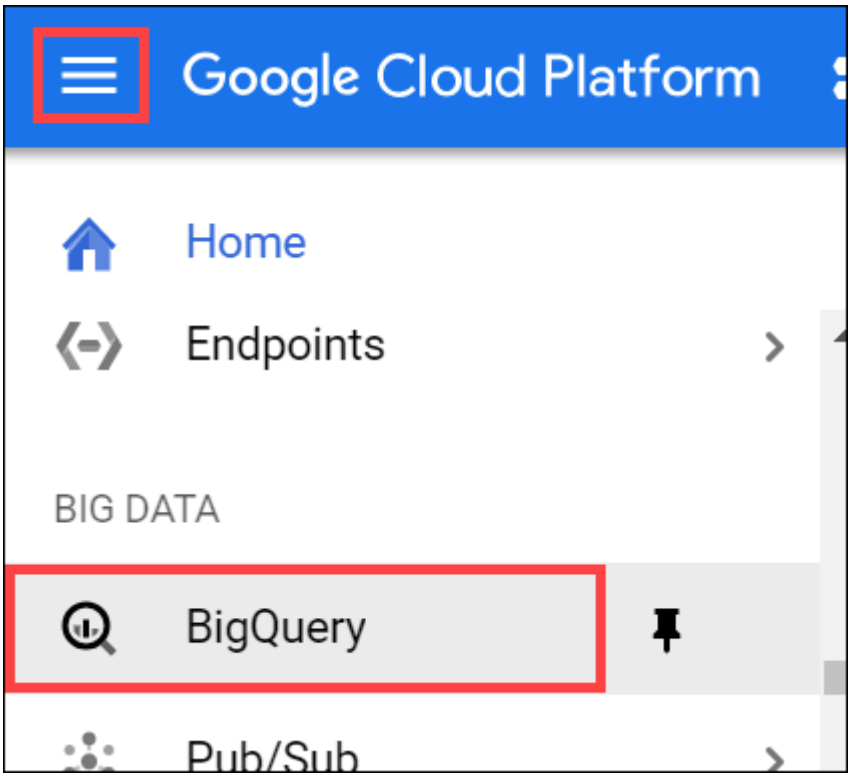
6. Click **Use another account** and copy/paste credentials for **this** lab into the prompts.
   If you use other credentials, you'll receive errors or **incur charges**.

7. Accept the terms and skip the recovery resource page.

Open BigQuery and Pin a project to the Resource tree

In this section, you add the **data-to-insights** project to your environment resources.

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



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

Note: The Welcome to BigQuery in the Cloud Console message box provides a link to the quickstart guide and UI updates.

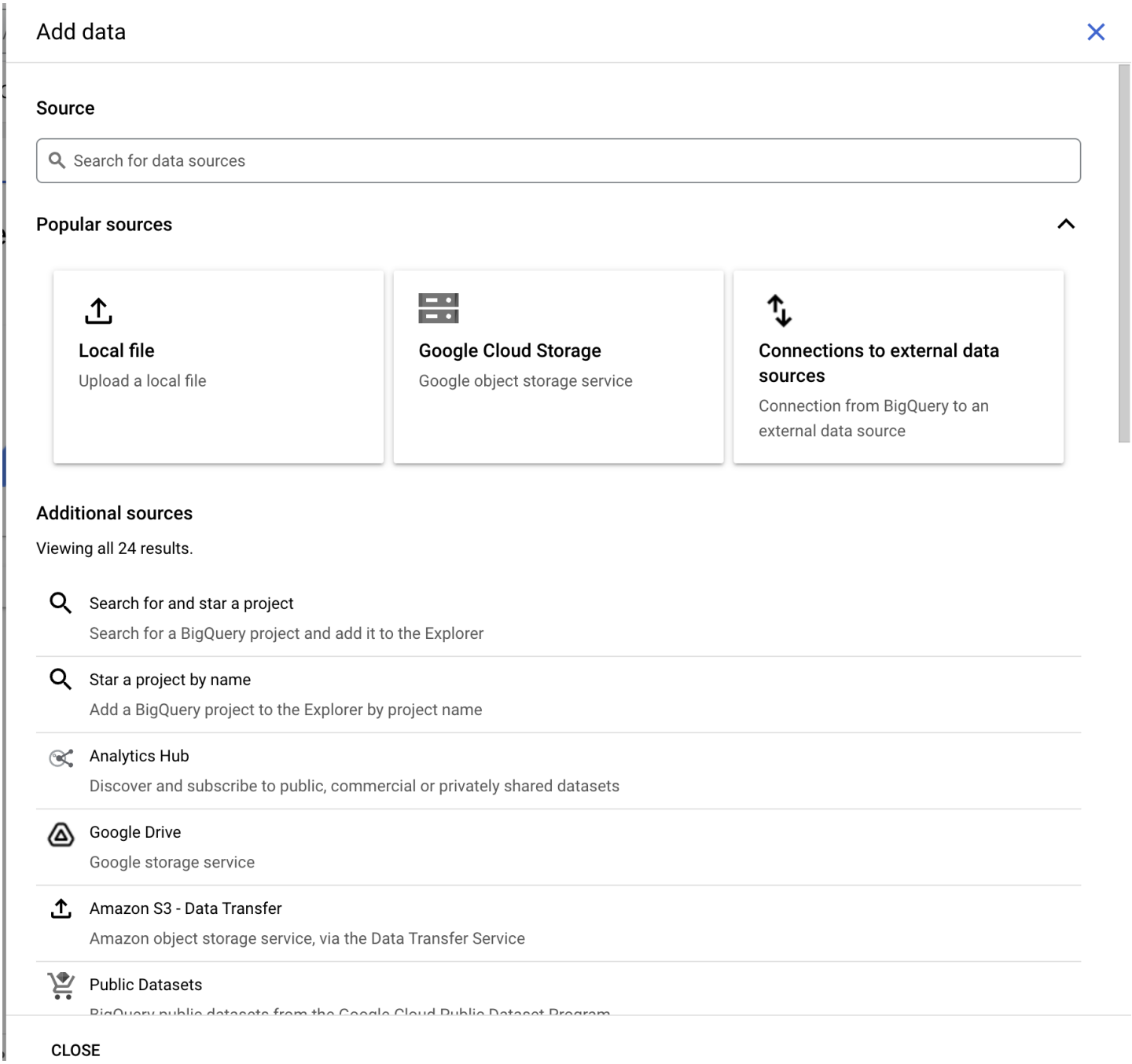
2. Click **Done**.

BigQuery public datasets are not displayed by default in the BigQuery web UI.

To open the public datasets project:

1. Click **+ ADD DATA**.

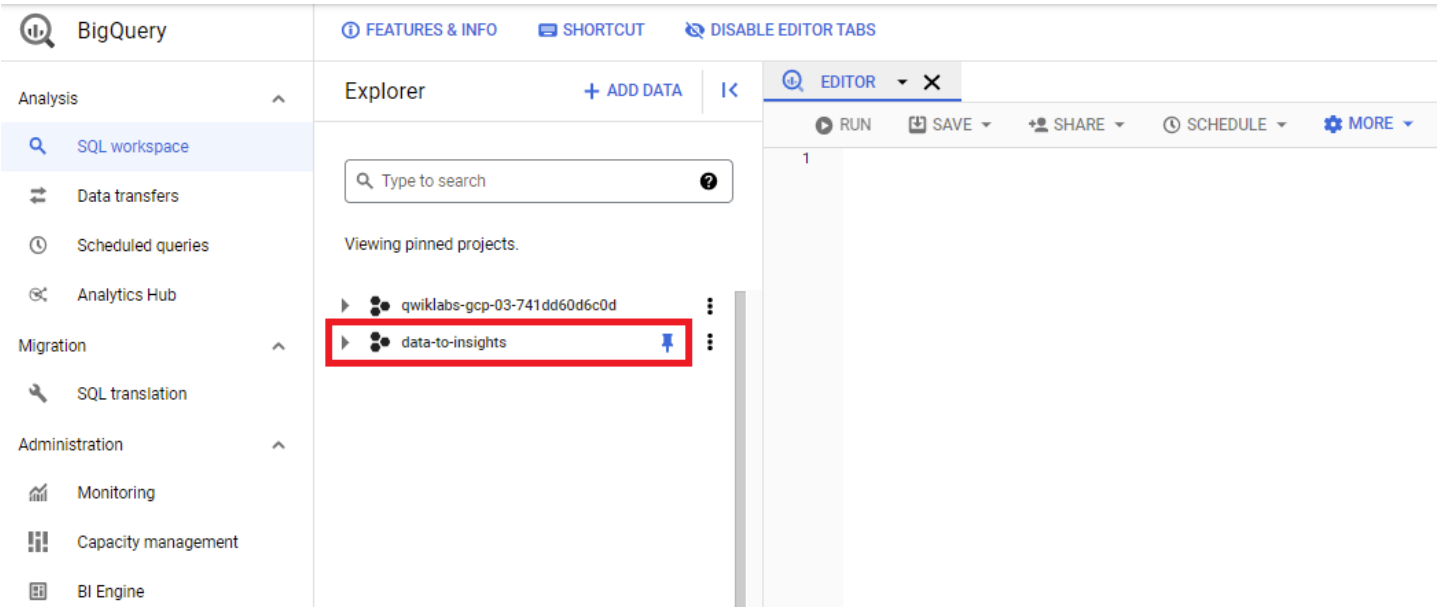
2. Select **Star a project by name**.



3. For **Project name**, enter `data-to-insights`.

4. Click **STAR**.

In the left pane, under **Viewing pinned projects** you will see the **data-to-insights** project pinned.

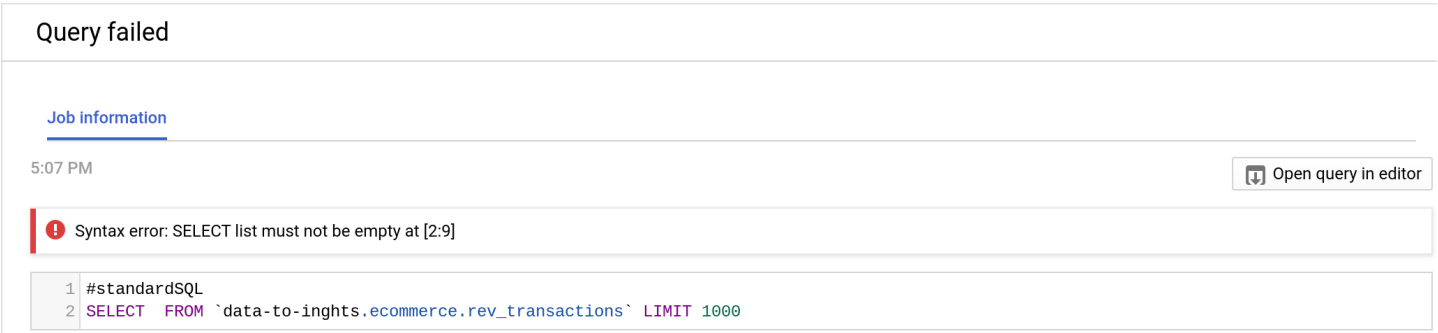


BigQuery Code editor

For each activity in the following sections, this lab provides queries with common errors for you to troubleshoot. The lab directs you what to look at and suggests how to correct the syntax and return meaningful results.

To follow along with the troubleshooting and suggestions, copy and paste the query into the BigQuery EDITOR. If there are errors you see a red exclamation point at the line containing the error and in the query validator (bottom corner).

If you run the query with the errors, the query fails and the error is specified in the Job information.



When the query is error free, you see a green checkmark in the query validator. When you see the green checkmark, click **RUN** to run the query to view what you get for results.

For information about syntax, see Standard SQL Query Syntax.

Find the total number of customers who went through checkout

Your goal in this section is to construct a query that gives you the number of unique visitors who successfully went through the checkout process for your website. The data is in the rev_transactions table which your data analyst team has provided. They have also given you example queries to help you get started in your analysis but you're not sure they're written correctly.

Troubleshoot queries that contain query validator, alias, and comma errors

Look at the below query and answer the following question

#standardSQL SELECT FROM `data-to-inghts.ecommerce.rev_transactions` LIMIT 1000

What about this updated query?

#standardSQL SELECT * FROM [data-to-insights:ecommerce.rev_transactions] LIMIT 1000

What about this query that uses Standard SQL?

#standardSQL SELECT FROM `data-to-insights.ecommerce.rev_transactions`

What about now? This query has a column.

#standardSQL SELECT fullVisitorId FROM `data-to-insights.ecommerce.rev_transactions`

What about now? The following query has a page title.

#standardSQL SELECT fullVisitorId hits_page_pageTitle FROM `data-to-insights.ecommerce.rev_transactions` LIMIT 1000

What about now? The missing comma has been corrected.

#standardSQL SELECT fullVisitorId , hits_page_pageTitle FROM `data-to-insights.ecommerce.rev_transactions` LIMIT 1000

Answer: This returns results, but are you sure visitors aren't counted twice? Also, returning only one row answers the question of how many unique visitors reached checkout. In the next section you find a way to aggregate your results.

Troubleshoot queries that contain logic errors, GROUP BY statements, and wildcard filters

Aggregate the following query to answer the question: How many unique visitors reached checkout?

#standardSQL SELECT fullVisitorId , hits_page_pageTitle FROM `data-to-insights.ecommerce.rev_transactions` LIMIT 1000

What about this? An aggregation function, `COUNT()`, was added.

#standardSQL SELECT COUNT(fullVisitorId) AS visitor_count , hits_page_pageTitle FROM `data-to-insights.ecommerce.rev_transactions`

In this next query, `GROUP BY` and `DISTINCT` statements were added.

#standardSQL SELECT COUNT(DISTINCT fullVisitorId) AS visitor_count , hits_page_pageTitle FROM `data-to-insights.ecommerce.rev_transactions` GROUP BY hits_page_pageTitle

Results

Query results SAVE RESULTS EXPLORE DATA

Query complete (0.4 sec elapsed, 2.4 MB processed)

Job information Results JSON Execution details

Row	visitor_count	hits_page_pageTitle
1	19981	Checkout Confirmation
2	1	2: Checkout Confirmation
3	1	6: Checkout Confirmation
4	1	11: Checkout Confirmation
5	1	Checkout Confirmation - https://shop.googlemerchandisestore.com/ordercompleted.html?vid=20160512512&orderDataId=33312
6	1	Mugs & Cups | Drinkware | Google Merchandise Store
7	1	Checkout Confirmation - https://shop.googlemerchandisestore.com/ordercompleted.html?vid=20160512512&orderDataId=13146
8	1	2 Checkout Confirmation
9	1	Checkout Confirmation - https://shop.googlemerchandisestore.com/ordercompleted.html?vid=20160512512&orderDataId=13522

Great! The results are good, but they look strange. Filter to just "Checkout Confirmation" in the results.

#standardSQL SELECT COUNT(DISTINCT fullVisitorId) AS visitor_count , hits_page_pageTitle FROM `data-to-insights.ecommerce.rev_transactions` WHERE hits_page_pageTitle = "Checkout Confirmation" GROUP BY hits_page_pageTitle

List the cities with the most transactions with your ecommerce site

Troubleshoot ordering, calculated fields, and filtering after aggregating errors

Complete the partially written query:

SELECT geoNetwork_city, totals_transactions, COUNT(DISTINCT fullVisitorId) AS distinct_visitors FROM `data-to-insights.ecommerce.rev_transactions` GROUP BY

Possible solution

#standardSQL SELECT geoNetwork_city, SUM(totals_transactions) AS totals_transactions, COUNT(DISTINCT fullVisitorId) AS distinct_visitors FROM `data-to-insights.ecommerce.rev_transactions` GROUP BY geoNetwork_city

Update your previous query to order the top cities first.

Possible solution

#standardSQL SELECT geoNetwork_city, SUM(totals_transactions) AS totals_transactions, COUNT(DISTINCT fullVisitorId) AS distinct_visitors FROM `data-to-insights.ecommerce.rev_transactions` GROUP

BY geoNetwork_city ORDER BY distinct_visitors DESC

Update your query and create a new calculated field to return the average number of products per order by city.

Possible solution

#standardSQL SELECT geoNetwork_city, SUM(totals_transactions) AS total_products_ordered, COUNT(DISTINCT fullVisitorId) AS distinct_visitors, SUM(totals_transactions) / COUNT(DISTINCT fullVisitorId) AS avg_products_ordered FROM `data-to-insights.ecommerce.rev_transactions` GROUP BY geoNetwork_city ORDER BY avg_products_ordered DESC

Results

Query results SAVE RESULTS EXPLORE DATA

Query complete (1.4 sec elapsed, 2.6 MB processed)

Job information Results JSON Execution details

Row	geoNetwork_city	total_products_ordered	distinct_visitors	avg_products_ordered
1	Jakarta	254	7	36.285714285714285
2	Maracaibo	409	21	19.476190476190474
3	Salem	252	16	15.75
4	Quito	15	1	15.0
5	North Attleborough	13	1	13.0
6	Fort Collins	11	1	11.0
7	Atwater	17	2	8.5
8	Ahmedabad	8	1	8.0
9	Milwaukee	7	1	7.0

Rows per page: 10 1 - 10 of 149 First page Last page

Filter your aggregated results to only return cities with more than 20 avg_products_ordered.

What's wrong with the following query?

#standardSQL SELECT geoNetwork_city, SUM(totals_transactions) AS total_products_ordered, COUNT(DISTINCT fullVisitorId) AS distinct_visitors, SUM(totals_transactions) / COUNT(DISTINCT fullVisitorId) AS avg_products_ordered FROM `data-to-insights.ecommerce.rev_transactions` WHERE avg_products_ordered > 20 GROUP BY geoNetwork_city ORDER BY avg_products_ordered DESC

Possible solution

#standardSQL SELECT geoNetwork_city, SUM(totals_transactions) AS total_products_ordered, COUNT(DISTINCT fullVisitorId) AS distinct_visitors, SUM(totals_transactions) / COUNT(DISTINCT fullVisitorId) AS avg_products_ordered FROM `data-to-insights.ecommerce.rev_transactions` GROUP BY geoNetwork_city HAVING avg_products_ordered > 20 ORDER BY avg_products_ordered DESC

Find the total number of products in each product category

Find the top selling products by filtering with NULL values

What's wrong with the following query? How can you fix it?

#standardSQL SELECT hits_product_v2ProductName, hits_product_v2ProductCategory FROM `data-to-insights.ecommerce.rev_transactions` GROUP BY 1,2

What is wrong with the following query?

#standardSQL SELECT COUNT(hits_product_v2ProductName) as number_of_products, hits_product_v2ProductCategory FROM `data-to-insights.ecommerce.rev_transactions` WHERE hits_product_v2ProductName IS NOT NULL GROUP BY hits_product_v2ProductCategory ORDER BY number_of_products DESC

Update the previous query to only count distinct products in each product category.

Possible solution

#standardSQL SELECT COUNT(DISTINCT hits_product_v2ProductName) as number_of_products, hits_product_v2ProductCategory FROM `data-to-insights.ecommerce.rev_transactions` WHERE hits_product_v2ProductName IS NOT NULL GROUP BY hits_product_v2ProductCategory ORDER BY number_of_products DESC LIMIT 5

Congratulations!

You have troubleshooted and fixed broken queries in BigQuery standard SQL. Remember to use the Query Validator for incorrect query syntax but also to be critical of your query results even if your query executes successfully.

End your lab

When you have completed your lab, click **End Lab**. Google Cloud Skills Boost removes the resources you've used and cleans the account for you.

You will be given an opportunity to rate the lab experience. Select the applicable number of stars, type a comment, and then click **Submit**.

The number of stars indicates the following:

- 1 star = Very dissatisfied
- 2 stars = Dissatisfied
- 3 stars = Neutral
- 4 stars = Satisfied
- 5 stars = Very satisfied

You can close the dialog box if you don't want to provide feedback.

For feedback, suggestions, or corrections, please use the **Support** tab.