Exploring an Ecommerce Dataset using SQL in Google BigQuery

experiment Lab schedule 1 hour 30 minutes universal_currency_alt No cost

show_chart Introductory

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 can take advantage of the pay-as-you-go model. BigQuery allows you to focus on analyzing data to find meaningful insights.

We have a newly available ecommerce dataset that has millions of Google Analytics records for the Google Merchandise Store loaded into a table in BigQuery. In this lab, you use a copy of that dataset. Sample scenarios are provided, from which you look at the data and ways to remove duplicate information. The lab then steps you through further analysis the data.

To follow and experiment with the BigQuery queries provided to analyze the data, see Standard SQL Query Syntax.

What you'll do

In this lab, you use BigQuery to:

- Access an ecommerce dataset
- Look at the dataset metadata
- Remove duplicate entries
- Write and execute queries

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**.

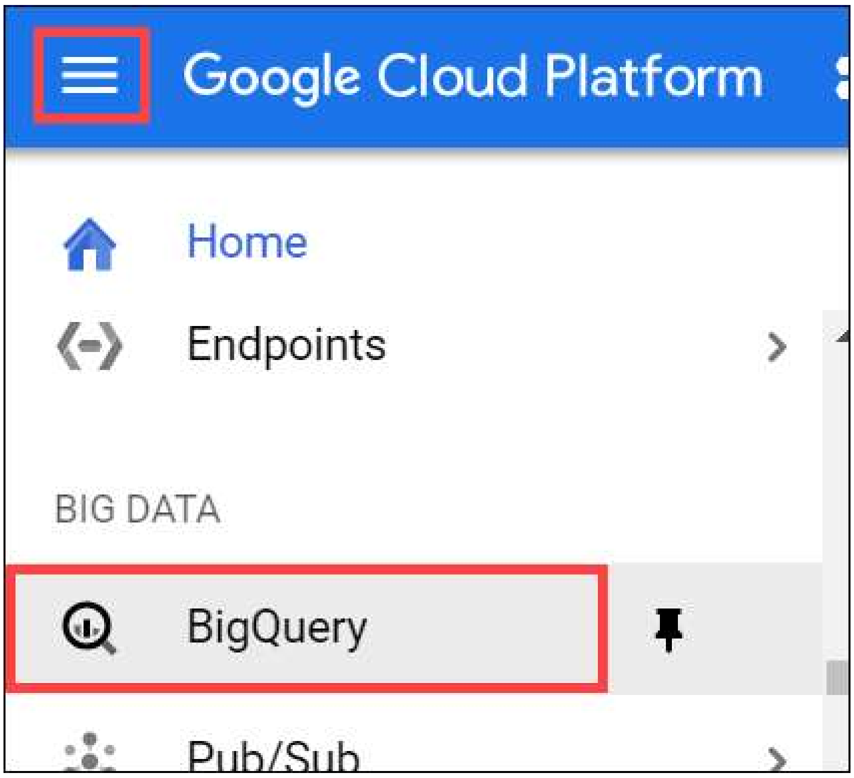
7. Accept the terms and skip the recovery resource page.

Note: Do not click **End Lab** unless you have finished the lab or want to restart it. This clears your work and removes the project.

Task 1. Star the lab project in BigQuery

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 so you'll open the public datasets project.

3. Click + **Add Data**.

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

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

6. Click **Star**.

7. In the explorer pane, you will see the **data-to-insights** project starred.

Task 2. Explore ecommerce data and identify duplicate records

Scenario: Your data analyst team exported the Google Analytics logs for an ecommerce website into BigQuery and created a new table of all the raw ecommerce visitor session data.

Explore the `all_sessions_raw` table data:

1. Expand **data-to-insights** project.

2. Expand **ecommerce**.

3. Click **all_sessions_raw**.

In the right pane, a section opens that provides 3 views of the table data:

- Schema tab: Field name, Type, Mode, and Description; the logical constraints used to organize the data
- Details tab: Table metadata
- Preview tab: Table preview

4. Click the **Details** tab to view the table metadata.

Questions:

Which UI tab will show you the data types?

- ○ Details
- ○ Preview
- check Schema

Submit

○

How many rows are in the dataset?

check Over 21 million

○ 5.63 GB

○ 9990000

Submit

Identify duplicate rows

Seeing a sample amount of data may give you greater intuition for what is included in the dataset. To preview sample rows from the table without using SQL, click the **Preview** tab.

Scan and scroll through the rows. There is no singular field that uniquely identifies a row, so you need advanced logic to identify duplicate rows.

Your query uses the SQL `GROUP BY` function on every field and counts (`COUNT`) where there are rows that have the same values across every field.

- If every field is unique, the `COUNT` will return 1 as there are no other groupings of rows with the exact same value for all fields.
- If there is a row with the same values for all fields, they will be grouped together and the `COUNT` will be greater than 1. The last part of the query is an aggregation filter using `HAVING` to only show the results that have a `COUNT` of duplicates greater than 1.
- Copy and paste the following query into the query **EDITOR**, then **RUN** to find duplicate records across all columns field. If the **EDITOR** tab isn't visible, then click **COMPOSE NEW QUERY**.

```
#standardSQL
SELECT COUNT(*) as num_duplicate_rows, * FROM
`data-to-insights.ecommerce.all_sessions_raw`
GROUP BY
fullVisitorId, channelGrouping, time, country, city,
totalTransactionRevenue, transactions, timeOnSite, pageviews,
sessionQualityDim, date, visitId, type, productRefundAmount,
productQuantity, productPrice, productRevenue, productSKU,
v2ProductName, v2ProductCategory, productVariant, currencyCode,
itemQuantity, itemRevenue, transactionRevenue, transactionId,
pageTitle, searchKeyword, pagePathLevel1, eCommerceAction_type,
eCommerceAction_step, eCommerceAction_option
HAVING num_duplicate_rows > 1;
```

content_c

How many duplicate records are in all_sessions_raw?

○ 434

○ 1,015

check 615

close ~~0~~

Submit

Note: In your own datasets, even if you have a unique key, it is still beneficial to confirm the uniqueness of the rows with COUNT, GROUP BY, and HAVING before you begin your analysis.

Analyze the new all_sessions table

In this section you use a deduplicated table called `all_sessions`.

Scenario: Your data analyst team has provided you with this query, and your schema experts have identified the key fields that must be unique for each record per your schema.

- Run the query to confirm that no duplicates exist, this time in the `all_sessions` table:

```
#standardSQL
# schema: https://support.google.com/analytics/answer/3437719?hl=en
SELECT
fullVisitorId, # the unique visitor ID
visitId, # a visitor can have multiple visits
date, # session date stored as string YYYYMMDD
time, # time of the individual site hit  (can be 0 to many per visitor
session)
v2ProductName, # not unique since a product can have variants like
Color
productSKU, # unique for each product
type, # a visitor can visit Pages and/or can trigger Events (even at
the same time)
eCommerceAction_type, # maps to ‘add to cart', ‘completed checkout'
eCommerceAction_step,
eCommerceAction_option,
  transactionRevenue, # revenue of the order
  transactionId, # unique identifier for revenue bearing transaction
COUNT(*) as row_count
FROM
`data-to-insights.ecommerce.all_sessions`
GROUP BY 1,2,3 ,4, 5, 6, 7, 8, 9, 10,11,12
HAVING row_count > 1 # find duplicates
```

content_c

The query returns zero records.

Note: In SQL, you can GROUP BY or ORDER BY the index of the column like using "GROUP BY 1" instead of "GROUP BY fullVisitorId".

Task 3. Write basic SQL on ecommerce data

In this section, you query for insights on the ecommerce dataset.

Write a query that shows total unique visitors

Your query determines the total views by counting `product_views` and the number of unique visitors by counting `fullVisitorID`.

1. Click **COMPOSE NEW QUERY**.

2. Write this query in the editor:

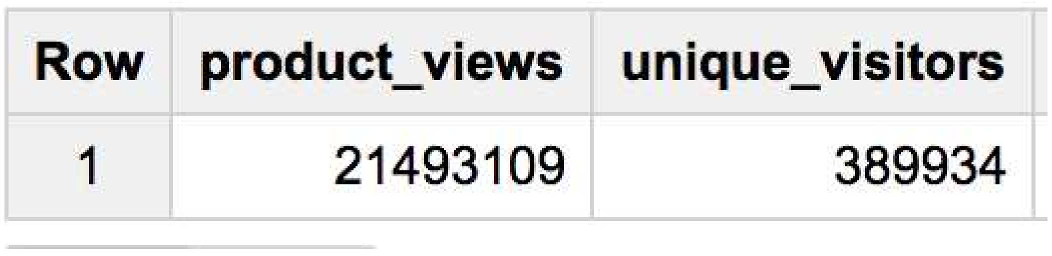
content_c

```
#standardSQL
SELECT
  COUNT(*) AS product_views,
  COUNT(DISTINCT fullVisitorId) AS unique_visitors
FROM `data-to-insights.ecommerce.all_sessions`;
```

3. To ensure that your syntax is correct, click the real-time query validator icon.

4. Click **RUN**. Read the results to view the number of unique visitors.

Results



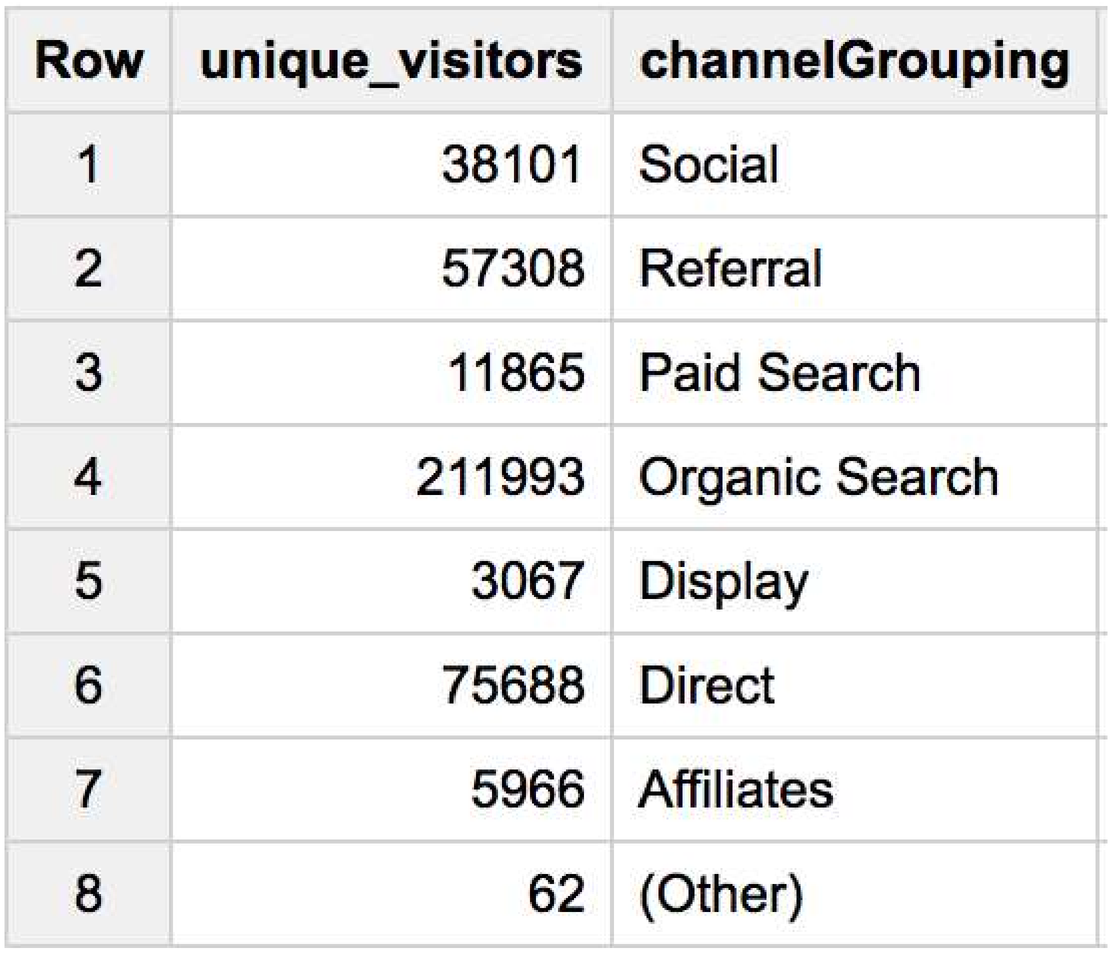
Row	product_views	unique_visitors
1	21493109	389934

5. Now write a query that shows total unique visitors(`fullVisitorID`) by the referring site (`channelGrouping`):

content_c

```
#standardSQL
SELECT
  COUNT(DISTINCT fullVisitorId) AS unique_visitors,
  channelGrouping
FROM `data-to-insights.ecommerce.all_sessions`
GROUP BY channelGrouping
ORDER BY channelGrouping DESC;
```

Results



Row	unique_visitors	channelGrouping
1	38101	Social
2	57308	Referral
3	11865	Paid Search
4	211993	Organic Search
5	3067	Display
6	75688	Direct
7	5966	Affiliates
8	62	(Other)

6. Write a query to list all the unique product names (`v2ProductName`) alphabetically:

```
#standardSQL
SELECT
  (v2ProductName) AS ProductName
FROM `data-to-insights.ecommerce.all_sessions`
GROUP BY ProductName
ORDER BY ProductName
```

content_c

Tip: In SQL, the ORDER BY clause defaults to Ascending (ASC) A to Z. If you want the reverse, try ORDER BY field_name DESC

Which part of the previous query deduplicates the records?

check GROUP BY

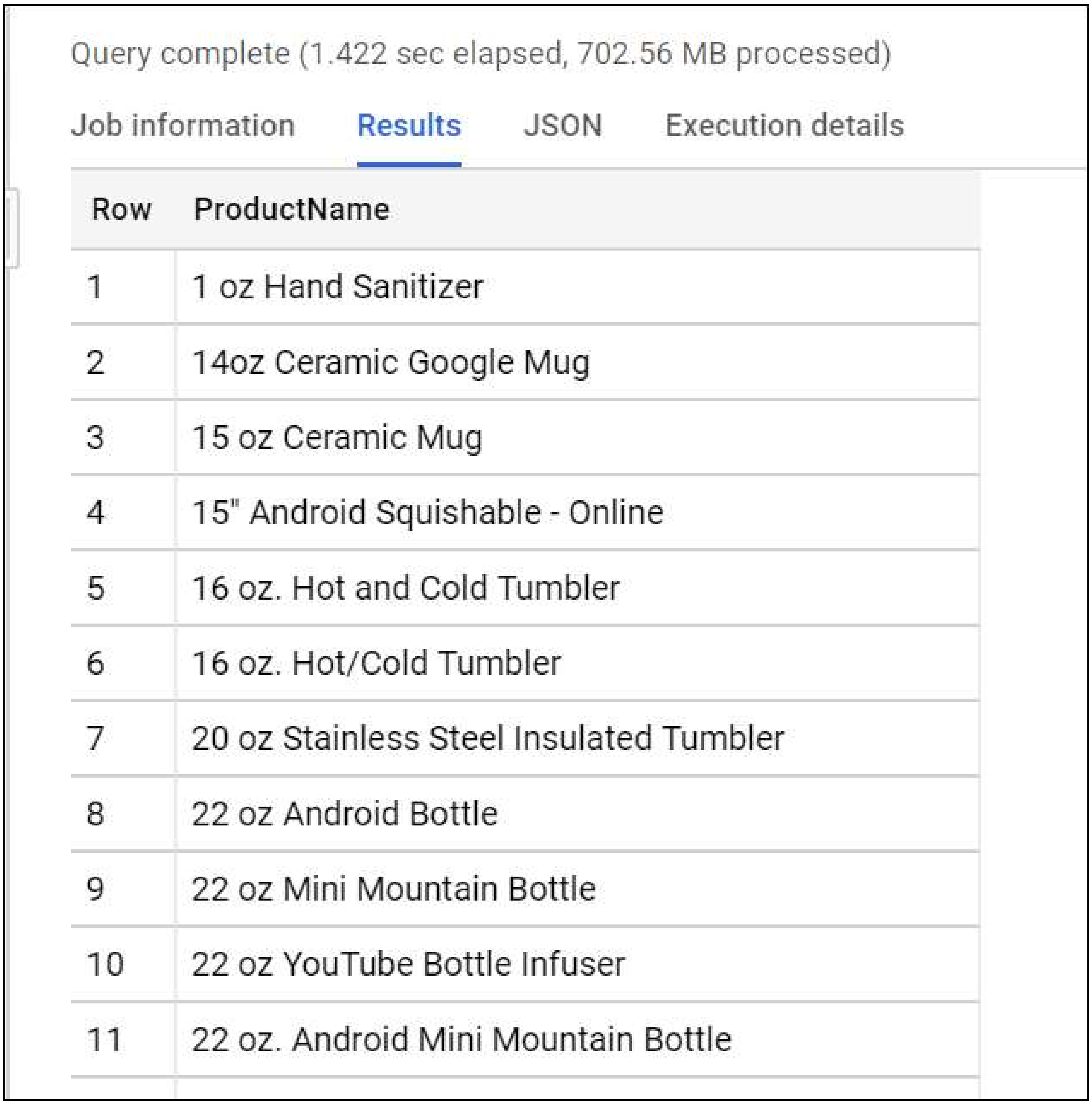
close ~~SELECT~~

○ STANDARD SQL

close ~~ORDER BY~~

Submit

Results



Query complete (1.422 sec elapsed, 702.56 MB processed)

Job information Results JSON Execution details

Row	ProductName
1	1 oz Hand Sanitizer
2	14oz Ceramic Google Mug
3	15 oz Ceramic Mug
4	15" Android Squishable - Online
5	16 oz. Hot and Cold Tumbler
6	16 oz. Hot/Cold Tumbler
7	20 oz Stainless Steel Insulated Tumbler
8	22 oz Android Bottle
9	22 oz Mini Mountain Bottle
10	22 oz YouTube Bottle Infuser
11	22 oz. Android Mini Mountain Bottle

7. This query returns a total of 633 products (rows).

◯

How many distinct product names were returned in total?

◯ 10

check 633

◯ 950

◯ 511

Submit

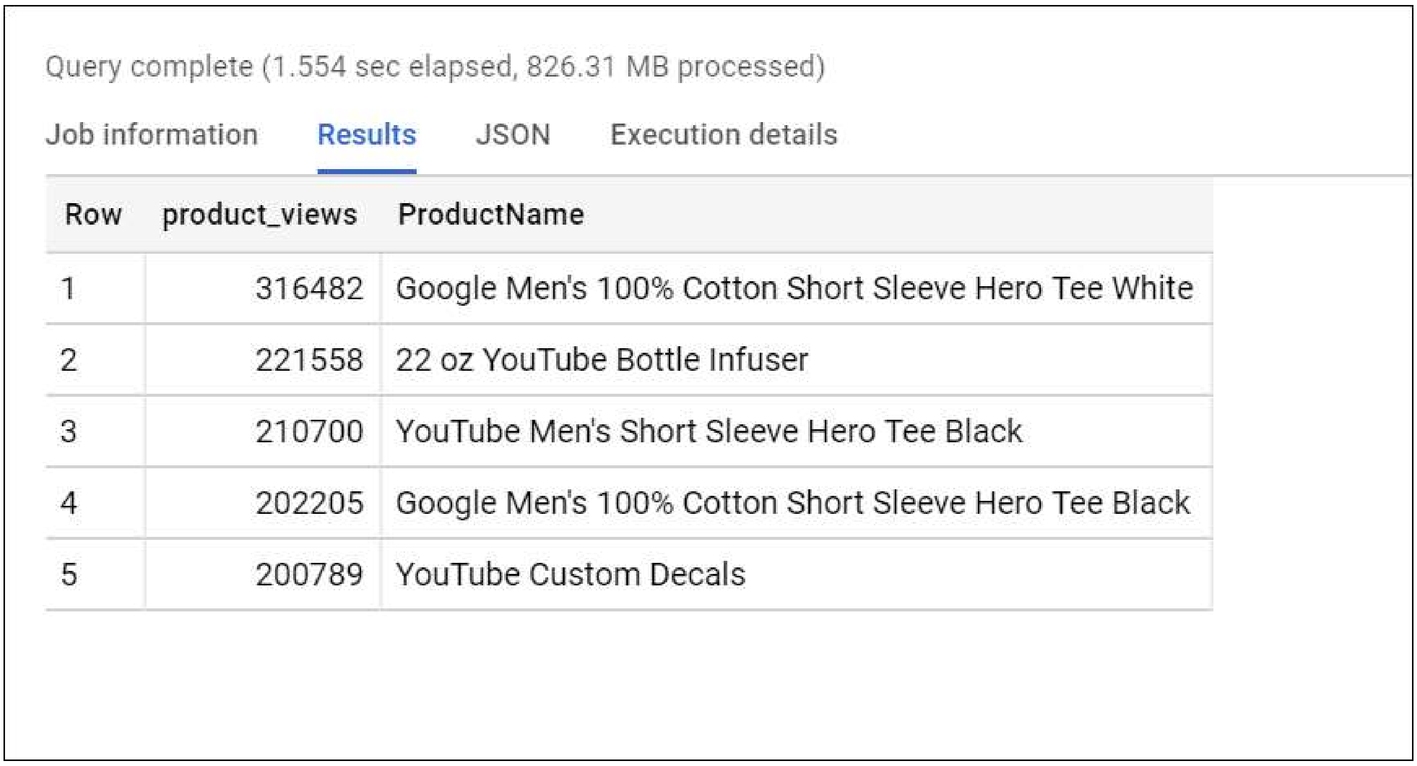
8. Write a query to list the five products with the most views (`product_views`) from all visitors (include people who have viewed the same product more than once). Your query counts number of times a product (`v2ProductName`) was viewed (`product_views`), puts the list in descending order, and lists the top 5 entries:

Tip: In Google Analytics, a visitor can "view" a product during the following interaction types: 'page', 'screenview', 'event', 'transaction', 'item', 'social', 'exception', 'timing'. For our purposes, simply filter for only type = 'PAGE'.

```
#standardSQL
SELECT
  COUNT(*) AS product_views,
  (v2ProductName) AS ProductName
FROM `data-to-insights.ecommerce.all_sessions`
WHERE type = 'PAGE'
GROUP BY v2ProductName
ORDER BY product_views DESC
LIMIT 5;
```

content_c

Results


Query complete (1.554 sec elapsed, 826.31 MB processed)

Row	product_views	ProductName
1	316482	Google Men's 100% Cotton Short Sleeve Hero Tee White
2	221558	22 oz YouTube Bottle Infuser
3	210700	YouTube Men's Short Sleeve Hero Tee Black
4	202205	Google Men's 100% Cotton Short Sleeve Hero Tee Black
5	200789	YouTube Custom Decals

9. Bonus: Refine the query to no longer double-count product views for visitors who have viewed a product many times. Each distinct product view should only count once per visitor.

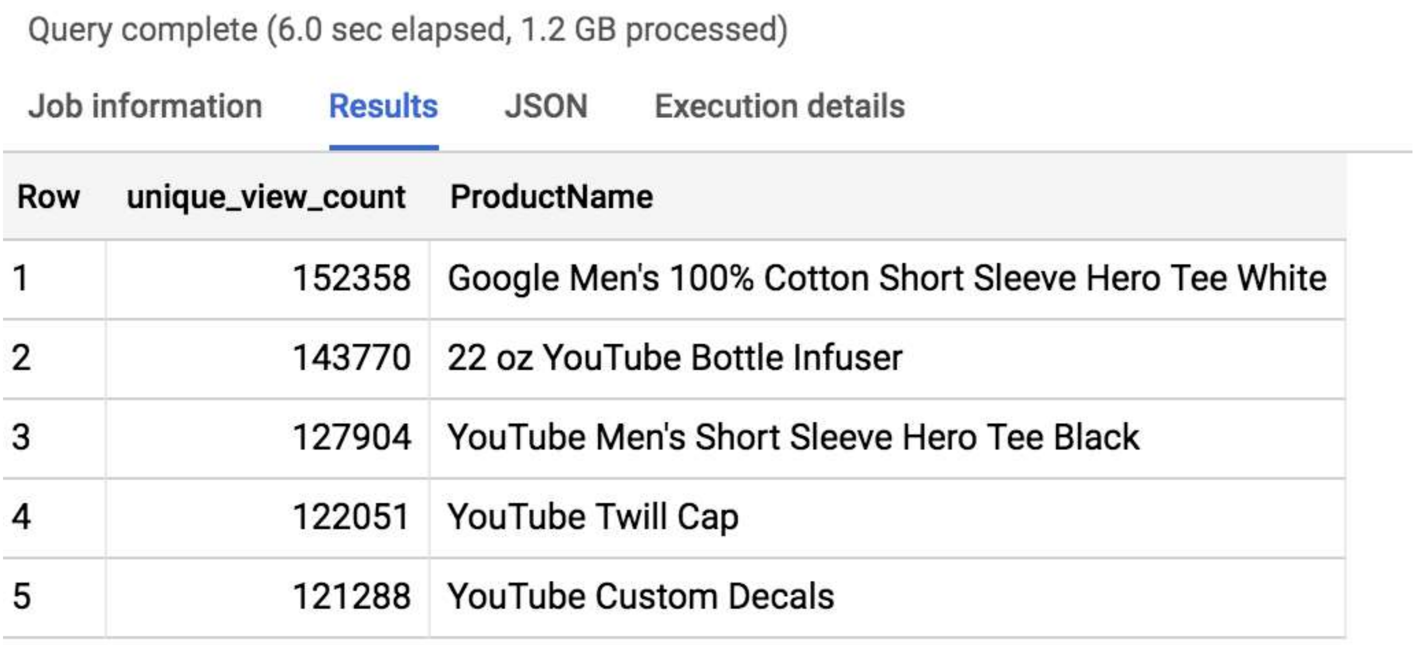
```
WITH unique_product_views_by_person AS (
-- find each unique product viewed by each visitor
SELECT
 fullVisitorId,
 (v2ProductName) AS ProductName
FROM `data-to-insights.ecommerce.all_sessions`
WHERE type = 'PAGE'
GROUP BY fullVisitorId, v2ProductName )


-- aggregate the top viewed products and sort them
SELECT
  COUNT(*) AS unique_view_count,
  ProductName
FROM unique_product_views_by_person
GROUP BY ProductName
ORDER BY unique_view_count DESC
LIMIT 5
```

content_c

Tip: You can use the SQL `WITH` clause to help break apart a complex query into multiple steps. Here we first create a query that finds each unique product per visitor and counts them once. Then the second query performs the aggregation across all visitors and products.

Results


Query complete (6.0 sec elapsed, 1.2 GB processed)

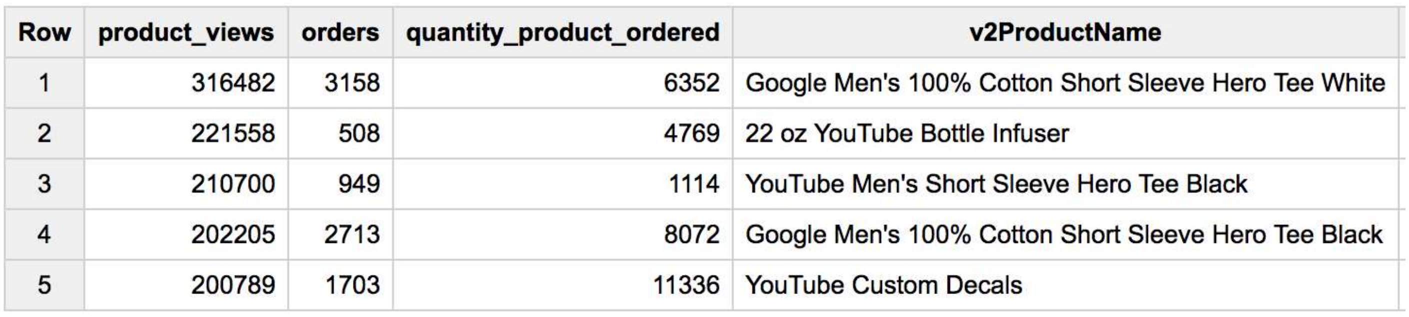
Row	unique_view_count	ProductName
1	152358	Google Men's 100% Cotton Short Sleeve Hero Tee White
2	143770	22 oz YouTube Bottle Infuser
3	127904	YouTube Men's Short Sleeve Hero Tee Black
4	122051	YouTube Twill Cap
5	121288	YouTube Custom Decals

10. Next, expand your previous query to include the total number of distinct products ordered and the total number of total units ordered (`productQuantity`):

content_c

```
#standardSQL
SELECT
  COUNT(*) AS product_views,
  COUNT(productQuantity) AS orders,
  SUM(productQuantity) AS quantity_product_ordered,
  v2ProductName
FROM `data-to-insights.ecommerce.all_sessions`
WHERE type = 'PAGE'
GROUP BY v2ProductName
ORDER BY product_views DESC
LIMIT 5;
```

Results



Row	product_views	orders	quantity_product_ordered	v2ProductName
1	316482	3158	6352	Google Men's 100% Cotton Short Sleeve Hero Tee White
2	221558	508	4769	22 oz YouTube Bottle Infuser
3	210700	949	1114	YouTube Men's Short Sleeve Hero Tee Black
4	202205	2713	8072	Google Men's 100% Cotton Short Sleeve Hero Tee Black
5	200789	1703	11336	YouTube Custom Decals

Questions:

The product with the most views got the most orders.

close ~~True~~

check False

What is the difference between orders and quantity_product_ordered?

check order is the number of orders, quantity_product_ordered is the number of items ordered

○ Nothing, they are the same

○ order is the number of orders, quantity_product_ordered is the number of items available to be ordered

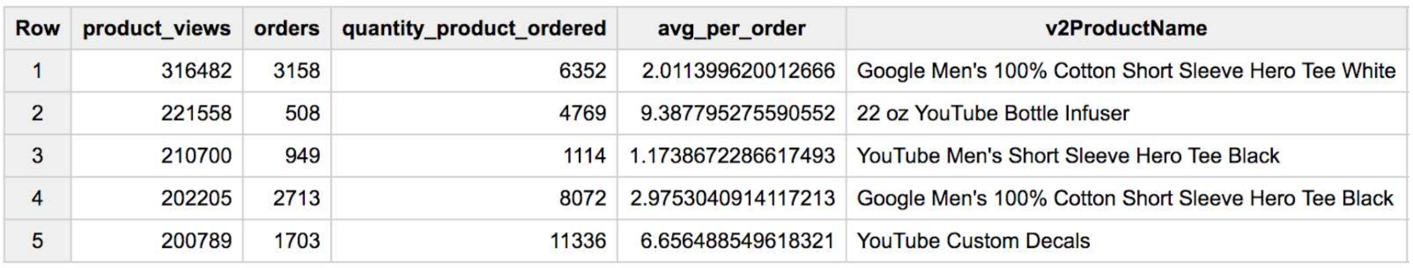
Submit

11. Expand the query to include the average amount of product per order (total number of units ordered/total number of orders, or `SUM(productQuantity)` / `COUNT(productQuantity)`):

content_c

```
#standardSQL
SELECT
  COUNT(*) AS product_views,
  COUNT(productQuantity) AS orders,
  SUM(productQuantity) AS quantity_product_ordered,
  SUM(productQuantity) / COUNT(productQuantity) AS avg_per_order,
  (v2ProductName) AS ProductName
FROM `data-to-insights.ecommerce.all_sessions`
WHERE type = 'PAGE'
GROUP BY v2ProductName
ORDER BY product_views DESC
LIMIT 5;
```

Results



Row	product_views	orders	quantity_product_ordered	avg_per_order	v2ProductName
1	316482	3158	6352	2.011399620012666	Google Men's 100% Cotton Short Sleeve Hero Tee White
2	221558	508	4769	9.387795275590552	22 oz YouTube Bottle Infuser
3	210700	949	1114	1.1738672286617493	YouTube Men's Short Sleeve Hero Tee Black
4	202205	2713	8072	2.9753040914117213	Google Men's 100% Cotton Short Sleeve Hero Tee Black
5	200789	1703	11336	6.656488549618321	YouTube Custom Decals

Question:

What product has the highest avg_per_order?

○ Google Mens 100% Cotton Short Sleeve Hero Tee Black

check 22 oz YouTube Bottle Infuser

○ YouTube Custom Decals

Submit

The 22 oz YouTube Bottle Infuser had the highest avg_per_order with 9.38 units per order.

Task 4. Practice with SQL

Are you ready to put your SQL skills to the test? Try these challenge questions!

Challenge 1: Calculate a conversion rate

1. Write a conversion rate query for products with these qualities:

- More than 1000 units were added to a cart or ordered
- AND are not frisbees

2. Answer these questions:

- How many distinct times was the product part of an order (either complete or incomplete order)?
- How many total units of the product were part of orders (either complete or incomplete)?
- Which product had the highest conversion rate?

3. Complete the following partial query:

content_c

```
#standardSQL
SELECT
  COUNT(*) AS product_views,
  COUNT(productQuantity) AS potential_orders,
  SUM(productQuantity) AS quantity_product_added,
  v2ProductName
FROM `data-to-insights.ecommerce.all_sessions`
WHERE v2ProductName NOT LIKE 'frisbee'
GROUP BY v2ProductName
HAVING quantity_product_added >
ORDER BY conversion_rate
LIMIT 10;
```

Possible solution:

content_c

```
#standardSQL
SELECT
  COUNT(*) AS product_views,
  COUNT(productQuantity) AS potential_orders,
  SUM(productQuantity) AS quantity_product_added,
  (COUNT(productQuantity) / COUNT(*)) AS conversion_rate,
  v2ProductName
FROM `data-to-insights.ecommerce.all_sessions`
WHERE LOWER(v2ProductName) NOT LIKE '%frisbee%'
GROUP BY v2ProductName
HAVING quantity_product_added > 1000
ORDER BY conversion_rate DESC
LIMIT 10;
```

Challenge 2: Track visitor checkout progress

- Write a query that shows the `eCommerceAction_type` and the distinct count of `fullVisitorId` associated with each type.

Possible solution:

content_c

```
#standardSQL
SELECT
  COUNT(DISTINCT fullVisitorId) AS number_of_unique_visitors,
  eCommerceAction_type
FROM `data-to-insights.ecommerce.all_sessions`
GROUP BY eCommerceAction_type
ORDER BY eCommerceAction_type;
```

Bonus

You are given this mapping for the action type: Unknown = 0 Click through of product lists = 1 Product detail views = 2 Add product(s) to cart = 3 Remove product(s) from cart = 4 Check out = 5 Completed purchase = 6 Refund of purchase = 7 Checkout options = 8

- Use a Case Statement to add a new column to your previous query to display the eCommerceAction_type label (such as “Completed purchase”).

Possible solution:

content_c

```
#standardSQL
SELECT
  COUNT(DISTINCT fullVisitorId) AS number_of_unique_visitors,
  eCommerceAction_type,
  CASE eCommerceAction_type
  WHEN '0' THEN 'Unknown'
  WHEN '1' THEN 'Click through of product lists'
  WHEN '2' THEN 'Product detail views'
  WHEN '3' THEN 'Add product(s) to cart'
  WHEN '4' THEN 'Remove product(s) from cart'
  WHEN '5' THEN 'Check out'
  WHEN '6' THEN 'Completed purchase'
  WHEN '7' THEN 'Refund of purchase'
  WHEN '8' THEN 'Checkout options'
  ELSE 'ERROR'
  END AS eCommerceAction_type_label
FROM `data-to-insights.ecommerce.all_sessions`
GROUP BY eCommerceAction_type
ORDER BY eCommerceAction_type;
```

What percent of visitors who added something to their cart completed a purchase?

Answer: 19988 / 56010 = .3568 or 35.68%

Challenge 3: Track abandoned carts from high quality sessions

- Write a query using aggregation functions that returns the unique session IDs of those visitors who have added a product to their cart but never completed checkout (abandoned their shopping cart).

Possible solution:

```
#standardSQL
# high quality abandoned carts
SELECT
  #unique_session_id
  CONCAT(fullVisitorId,CAST(visitId AS STRING)) AS unique_session_id,
  sessionQualityDim,
  SUM(productRevenue) AS transaction_revenue,
  MAX(eCommerceAction_type) AS checkout_progress
FROM `data-to-insights.ecommerce.all_sessions`
WHERE sessionQualityDim > 60 # high quality session
GROUP BY unique_session_id, sessionQualityDim
HAVING
  checkout_progress = '3' # 3 = added to cart
  AND (transaction_revenue = 0 OR transaction_revenue IS NULL)
```

content_c

End your lab

When you have completed your lab, click **End Lab**. Qwiklabs 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.