Exploring an Ecommerce Dataset using SQL in Google BigQuery

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
```

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"

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`.

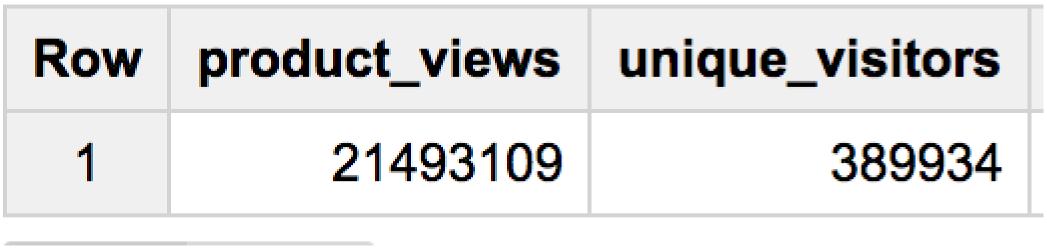
1. Click **Compose Query**.
2. Write this query in the editor:

```
#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 Query**. Read the results to view the number of unique visitors.

Results

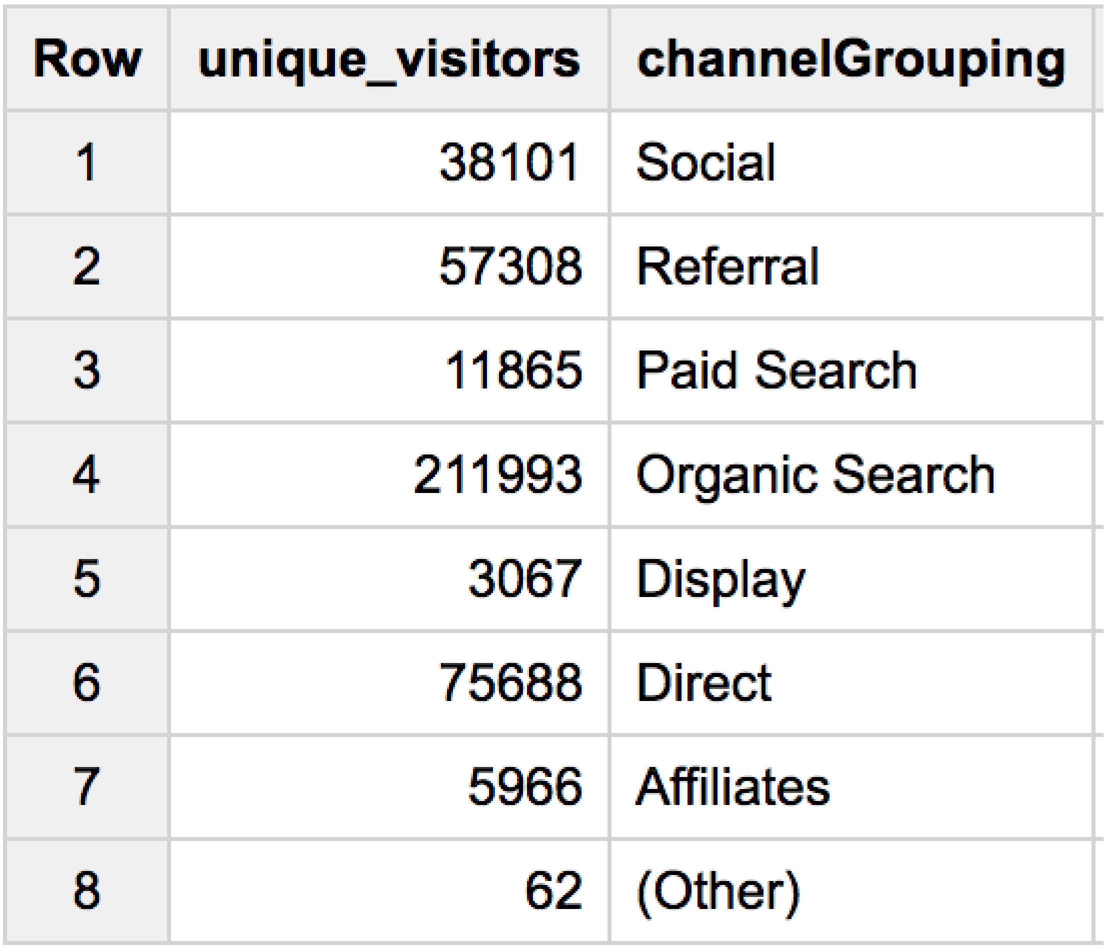


Row	product_views	unique_visitors
1	21493109	389934

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

```
#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)

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

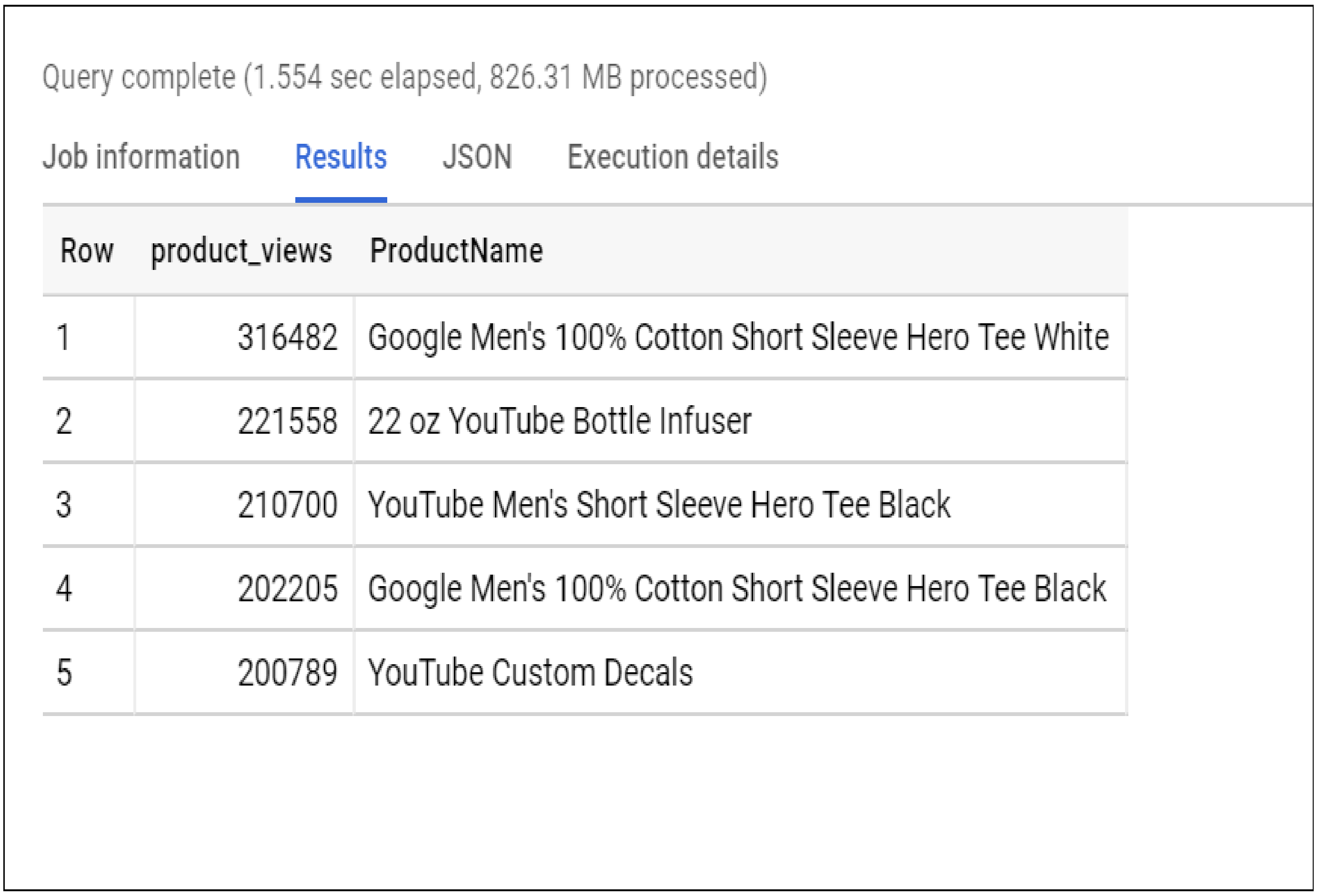
```
#standardSQL
SELECT
  (v2ProductName) AS ProductName
FROM `data-to-insights.ecommerce.all_sessions`
GROUP BY ProductName
ORDER BY ProductName
```

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;
```

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

Bonus: Now 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.

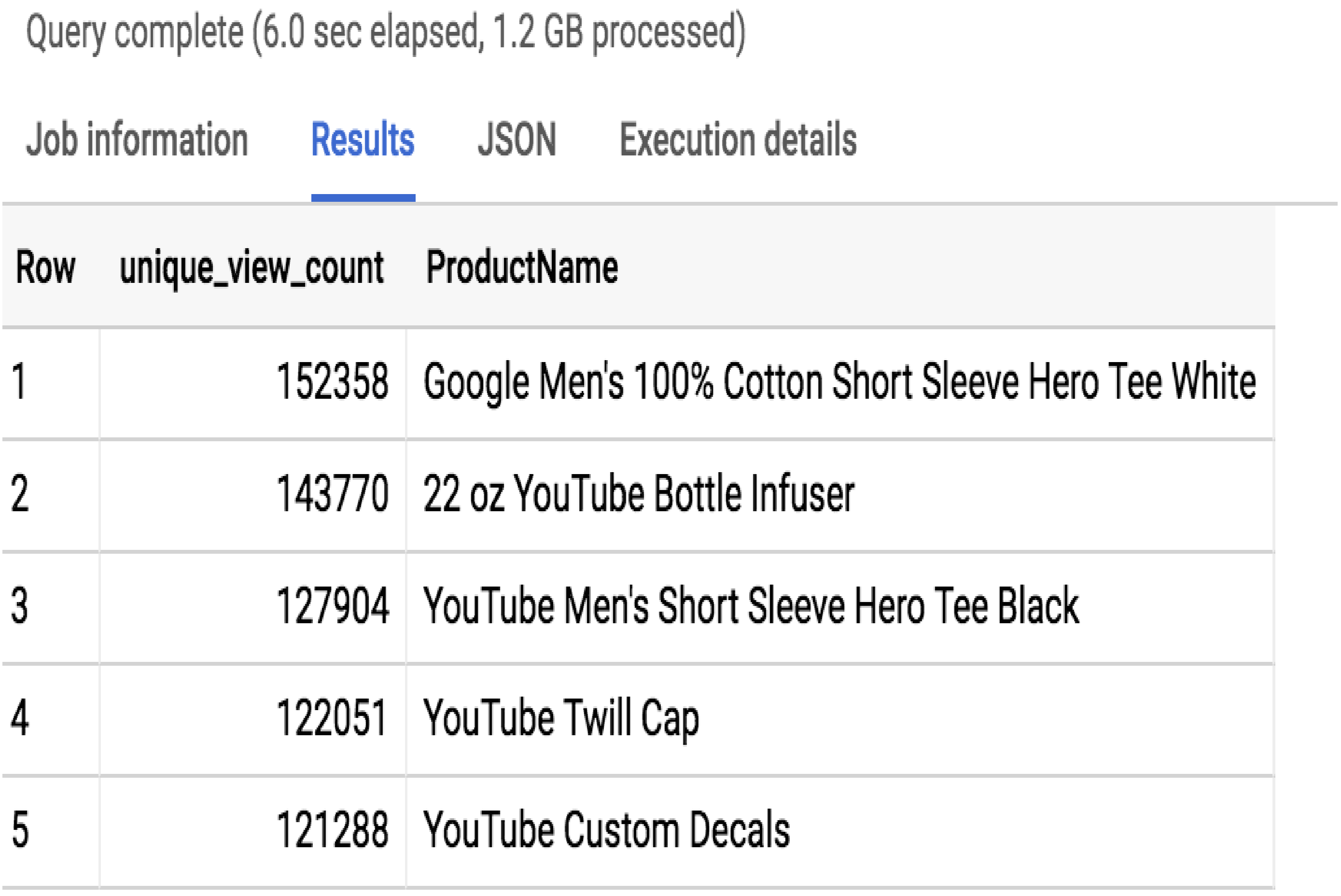
```
WITH unique_product_views_by_person AS (
-- find each unique product viewed by each visitor
SELECT
 fullVisitorId,
 (v2ProductName) AS ProductName
FROM `data-to-insights.ecommerce.all_sessions`
WHERE type = 'PAGE'
GROUP BY fullVisitorId, v2ProductName )


-- 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
```

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

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

```
#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;
```

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)`).

```
#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;
```

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

Practice with SQL

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

Challenge 1: Calculate a conversion rate

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

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?

Complete the following partial query:

```
#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:

```
#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:

```
#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

```
#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, sessionQu

alityDim
HAVING
  checkout_progress = '3' # 3 = added to cart
  AND (transaction_revenue = 0 OR transaction_revenue IS NULL)
```