
Creating a Data Warehouse Through Joins and Unions

Overview

This project focused on building a data warehouse in BigQuery by combining data from multiple sources:

1. Website ecommerce data
2. Product inventory stock levels and lead times
3. Product review sentiment analysis

The goal was to create a comprehensive view of product performance by joining these datasets and appending historical sales data.

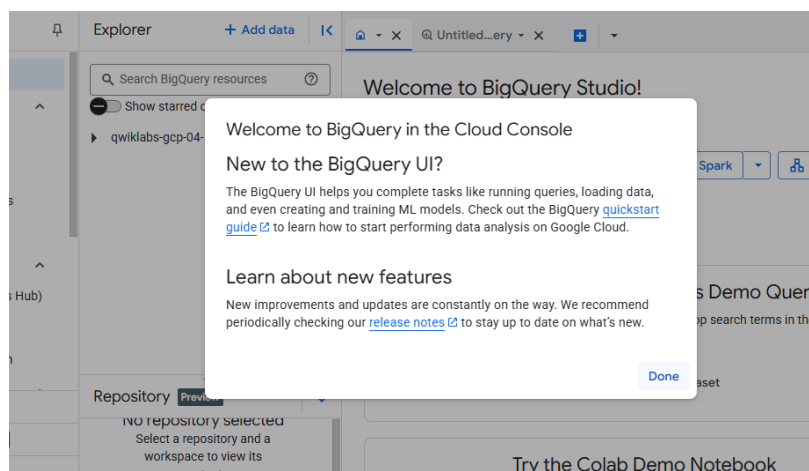
Task 1. Create a new dataset to store your tables

To get started, create a new dataset titled **ecommerce** in BigQuery to store your tables.

1. In the left pane, click on the name of your BigQuery project (qwiklabs-gcp-xxxx).
2. Click on the three dots next to your project name, then select **Create dataset**.

The **Create dataset** dialog opens.

3. Set the **Dataset ID** to ecommerce, leave all other options at their default values.
4. Click **Create dataset**.



Create dataset

Project ID *
qwiklabs-gcp-04-1646a6b9d918 [Change](#)

Dataset ID *

Letters, numbers, and underscores allowed

Location type [?](#)

☐ Region
Specify a region to colocate your datasets with other Google Cloud services.

☒ Multi-region
Allow BigQuery to select a region within a group to achieve higher quota limits.

i Some locations have been restricted due to a policy set by your organization. [Learn more about restricting locations.](#)

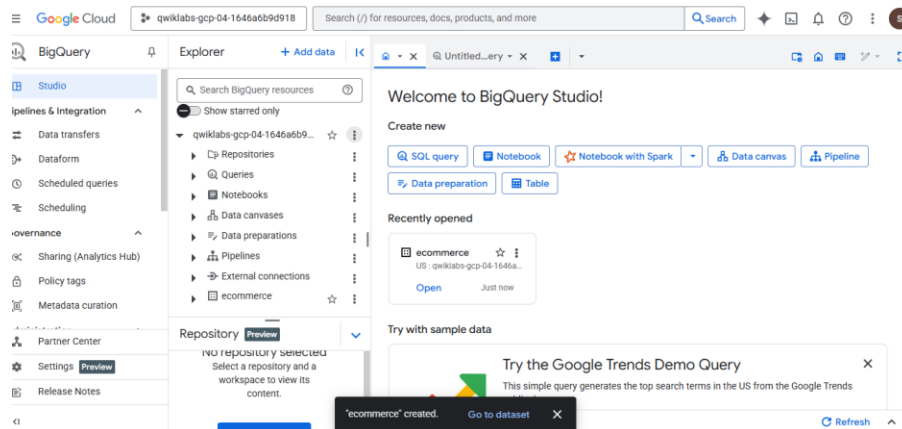
Multi-region *

External Dataset

The selected region supports the following external dataset types: Cloud Spanner

☐ Link to an external dataset [?](#)

[Create dataset](#) [Cancel](#)



Task 2. Explore the product sentiment dataset

Your data science team has run all of your product reviews through the API and provided you with the average sentiment score and magnitude for each of your products.

The project with your marketing team's dataset is **data-to-insights**. BigQuery public datasets are not displayed by default in BigQuery. The queries in this lab will use the data-to-insights dataset even though you cannot see it.

1. First, create a copy of the table that the data science team made so you can read it:

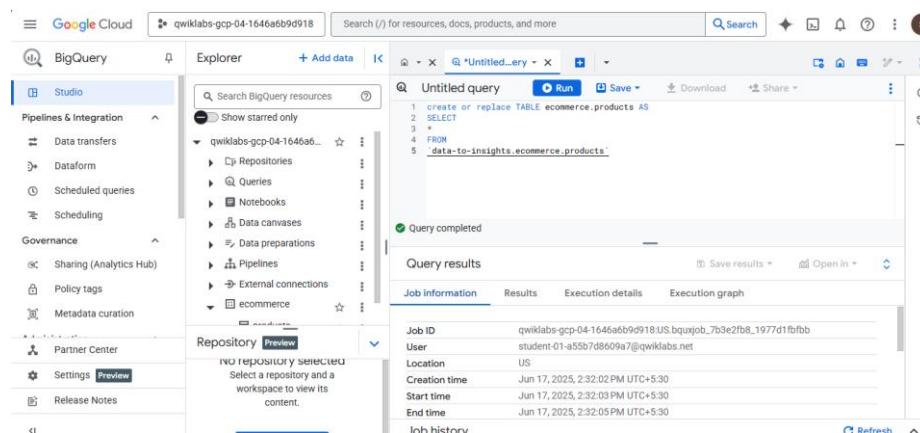
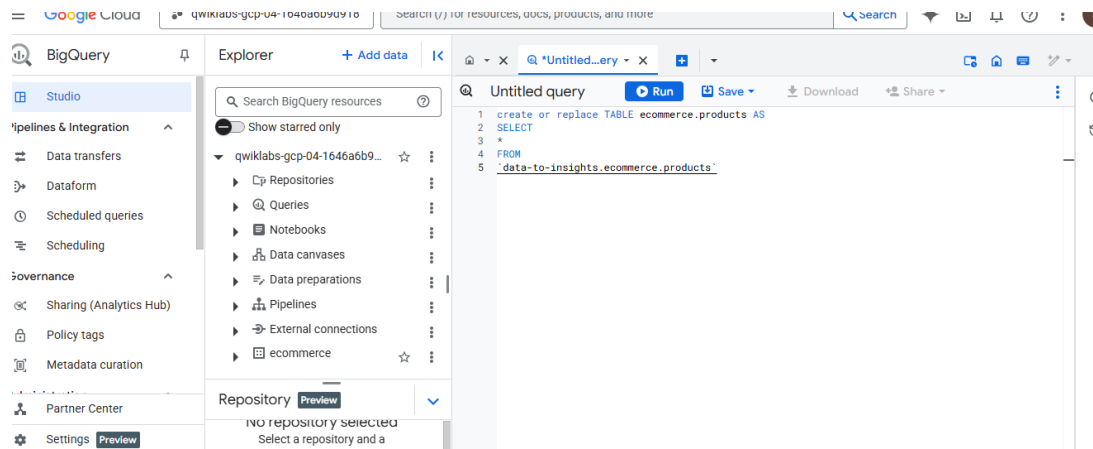
create or replace TABLE ecommerce.products AS

SELECT

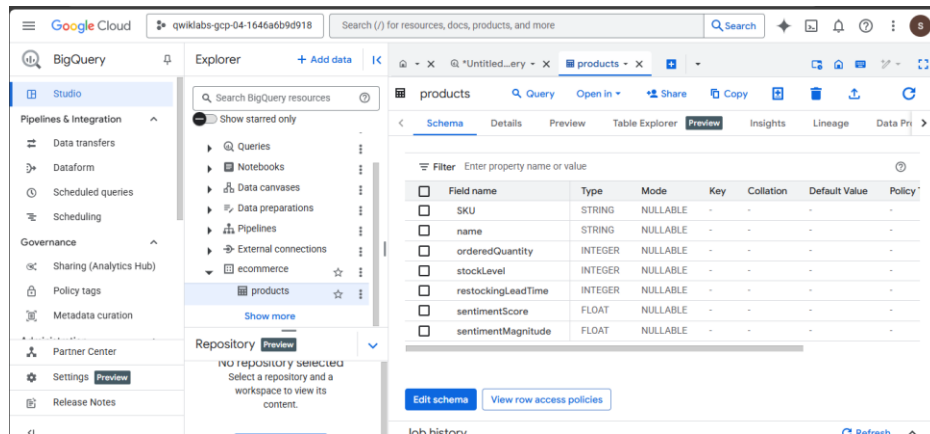
*

FROM

`data-to-insights.ecommerce.products`



2. Click on the **ecommerce** dataset to display the products table.
3. Navigate to the **ecommerce > products** dataset and click the **Preview** tab to see the data.



Create a query that shows the top 5 products with the most positive sentiment

1. In the **Query Editor**, write your SQL query.

SELECT

SKU,

name,

sentimentScore,

sentimentMagnitude

FROM

`data-to-insights.ecommerce.products`

ORDER BY

sentimentScore DESC

LIMIT 5

Query completed

Query results

Save results Open in

| Job information | Results | Chart | JSON | Execution details | Execution graph |
|-----------------|------------------|------------------------------------|----------------|--------------------|-----------------|
| Row | SKU | name | sentimentScore | sentimentMagnit... | |
| 1 | GGOBJGOWUSG69402 | USB wired soundbar - in store o... | 1.0 | 1.0 | |
| 2 | GGOEGOAR013099 | Stylus Pen w/ LED Light | 0.9 | 1.4 | |
| 3 | GGOEGADJ056816 | Men's Watershed Full Zip Hoodi... | 0.9 | 1.4 | |
| 4 | GGOEGADJ056816 | Metal Texture Dollar Bag | 0.9 | 1.4 | |

Results per page: 50 1 - 5 of 5

2. Revise your query to show the top 5 products with the most negative sentiment and filter out NULL values.

SELECT

SKU,

name,

sentimentScore,

sentimentMagnitude

FROM

`data-to-insights.ecommerce.products`

WHERE sentimentScore IS NOT NULL

ORDER BY

sentimentScore

LIMIT 5

The screenshot shows a SQL query editor with the following query:

```

1 SELECT
2   SKU,
3   name,
4   sentimentScore,
5   sentimentMagnitude
6 FROM
7   `data-to-insights.ecommerce.products`
8 WHERE sentimentScore IS NOT NULL
9 ORDER BY
10  sentimentScore
11 LIMIT 5

```

Below the query, the results are displayed in a table:

| Row | SKU | name | sentimentScore | sentimentMagnitude |
|-----|-------------|-----------------------------------|----------------|--------------------|
| 1 | GGOEGAA0098 | 7" Dog Frisbee | -0.6 | 0.2 |
| 2 | GGOEGAA0607 | Women's Convertible Vest-Jack... | -0.5 | 1.8 |
| 3 | GGOEGAA0344 | Women's Vintage Hero Tee Plati... | -0.5 | 1.1 |
| 4 | GGOEGAA0051 | Men's Vintage Hoodie | -0.5 | 1.1 |

Task 3. Join datasets to find insights

Scenario: It's the first of the month and your inventory team has informed you that the orderedQuantity field in the product inventory dataset is out of date. They need your help to query the total sales by product for 08/01/2017 and reference that against the current stock levels in inventory to see which products need to be resupplied first.

Calculate daily sales volume by productSKU

- Create a new table in your **ecommerce** dataset with the below requirements:
 - Title it sales_by_sku_20170801
 - Source the data from data-to-insights.ecommerce.all_sessions_raw
 - Include only distinct results
 - Return productSKU
 - Return the total quantity ordered (productQuantity). Hint: Use a SUM() with a IFNULL condition
 - Filter for only sales on 20170801
 - ORDER BY the SKUs with the most orders first
 - Click on the sales_by_sku table, then click the **Preview** tab.

The screenshot shows a SQL query editor with the following query:

```

1 # pull what sold on 08/01/2017
2 CREATE OR REPLACE TABLE ecommerce.sales_by_sku_20170801 AS
3 SELECT
4   productSKU,
5   SUM(IFNULL(productQuantity,0)) AS total_ordered
6 FROM
7   `data-to-insights.ecommerce.all_sessions_raw`
8 WHERE date = '20170801'
9 GROUP BY productSKU
10 ORDER BY total_ordered DESC #462 skus sold

```

Below the query, a status bar indicates "Query completed".

The "Query results" section shows a message: "This statement created a new table named sales_by_sku_20170801." with a "Go to table" button.

Join sales data and inventory data

- Using a JOIN, enrich the website ecommerce data with the following fields from the product inventory dataset:
 - name
 - stockLevel
 - restockingLeadTime
 - sentimentScore
 - sentimentMagnitude
- Complete the partially written query:

The screenshot shows a SQL query editor with the following query:

```

1 SELECT
2   SKU,
3   name,
4   sentimentScore,
5   sentimentMagnitude
6 FROM
7   `data-to-insights.ecommerce.products`
8 ORDER BY
9   sentimentScore DESC
10 LIMIT 5

```

Below the query, a status bar indicates "Query completed".

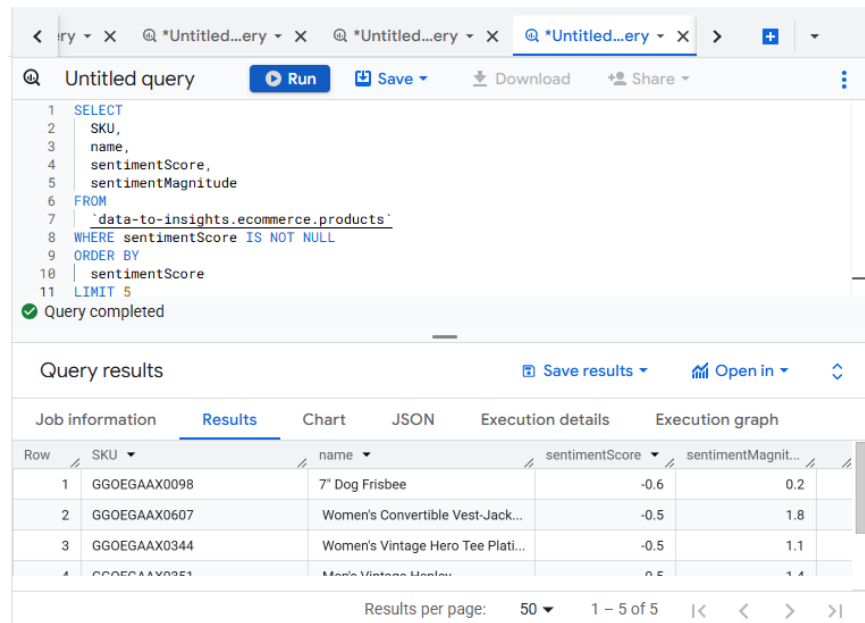
The "Query results" section shows a table with the following data:

| Row | SKU | name | sentimentScore | sentimentMagnit... |
|-----|------------------|------------------------------------|----------------|--------------------|
| 1 | GG0BJGOWUSG69402 | USB wired soundbar - in store o... | 1.0 | 1.0 |
| 2 | GG0EG0AR013099 | Stylus Pen w/ LED Light | 0.9 | 1.4 |
| 3 | GG0EGADJ056816 | Men's Watershed Full Zip Hoodi... | 0.9 | 1.4 |
| 4 | GG0EG0AR013099 | Model Traction Pillow Box | 0.9 | 1.4 |

Results per page: 50 | 1 - 5 of 5

3. Modify the query you wrote to now include:

- A calculated field of $(total_ordered / stockLevel)$ and alias it "ratio". **Hint:** Use `SAFE_DIVIDE(field1,field2)` to avoid dividing by 0 errors when the stock level is 0.
- Filter the results to only include products that have gone through 50% or more of their inventory already at the beginning of the month



The screenshot shows a SQL query editor with a query that selects product details and sentiment data. The query is as follows:

```
1 SELECT
2   SKU,
3   name,
4   sentimentScore,
5   sentimentMagnitude
6 FROM
7   `data-to-insights.ecommerce.products`
8 WHERE sentimentScore IS NOT NULL
9 ORDER BY
10  sentimentScore
11 LIMIT 5
```

Below the query, a status bar indicates "Query completed". The results are displayed in a table with the following columns: Row, SKU, name, sentimentScore, and sentimentMagnit... (truncated). The results are as follows:

| Row | SKU | name | sentimentScore | sentimentMagnit... |
|-----|--------------|-----------------------------------|----------------|--------------------|
| 1 | GGOEGAAX0098 | 7" Dog Frisbee | -0.6 | 0.2 |
| 2 | GGOEGAAX0607 | Women's Convertible Vest-Jack... | -0.5 | 1.8 |
| 3 | GGOEGAAX0344 | Women's Vintage Hero Tee Plati... | -0.5 | 1.1 |
| 4 | GGOEGAAX0251 | Men's Vintage Harley... | -0.5 | 1.4 |

At the bottom of the results table, there is a pagination bar showing "Results per page: 50" and "1 - 5 of 5".

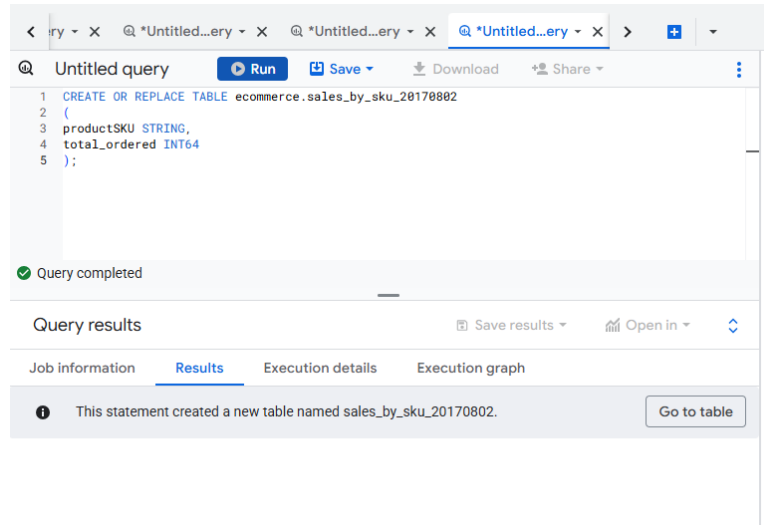
Task 4. Append additional records

Your international team has already made in-store sales on 08/02/2017 which you want to record in your daily sales tables.

Create a new empty table to store sales by productSKU for 08/02/2017

1. For the schema, specify the following fields:

- table name is `ecommerce.sales_by_sku_20170802`
- `productSKU` STRING
- `total_ordered` as an INT64 field



2. Confirm you now have two date-shared sales tables - use the dropdown menu next to the **Sales_by_sku** table name in the table results, or refresh your browser to see it listed in the left menu:

Insert the sales record provided to you by your sales team

4. Confirm the record appears by previewing the table - click on the table name to see the results.

Append together historical data

There are multiple ways to append together data that has the same schema. Two common ways are using UNIONS and table wildcards.

- **Union** is an SQL operator that appends together rows from different result sets.
- **Table wildcards** enable you to query multiple tables using concise SQL statements. Wildcard tables are available only in standard SQL.

1. Write a UNION query that will result in all records from the below two tables:

- ecommerce.sales_by_sku_20170801
- ecommerce.sales_by_sku_20170802

The screenshot shows a query editor with a query that unions two tables. The results table shows the first four rows of the combined data.

```

1 SELECT * FROM ecommerce.sales_by_sku_20170801
2 UNION ALL
3 SELECT * FROM ecommerce.sales_by_sku_20170802

```

Query completed

Query results

| Row | productSKU | totalOrdered |
|-----|----------------|--------------|
| 1 | GGOEGOAQ012899 | 456 |
| 2 | GGOEGDHC074099 | 334 |
| 3 | GGOEGOCB017499 | 319 |
| 4 | GGOEGGCC077999 | 290 |

Results per page: 50 1 - 50 of 463

What is a pitfall of having many daily sales tables? You will have to write many UNION statements chained together.

A better solution is to use the table wildcard filter and `_TABLE_SUFFIX` filter.

- Write a query that uses the (*) table wildcard to select all records from `ecommerce.sales_by_sku_` for the year 2017.

`SELECT * FROM ecommerce.sales_by_sku_2017*`

The screenshot shows a query editor with a query that uses a wildcard to select all records from sales tables for the year 2017. The results table shows the first nine rows of the data.

```

1 SELECT * FROM `ecommerce.sales_by_sku_2017*`

```

Query completed

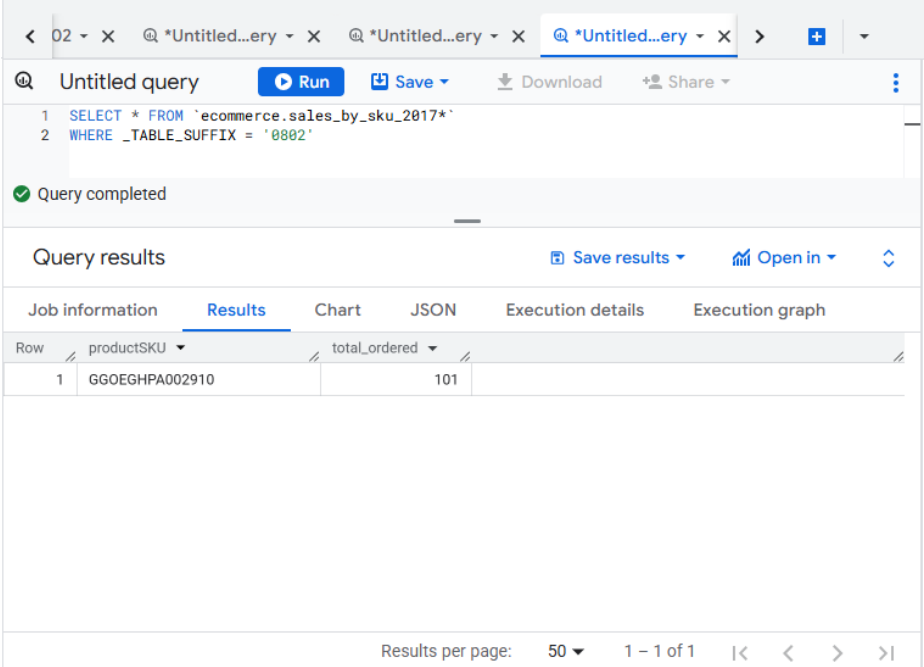
Query results

| Row | productSKU | totalOrdered |
|-----|----------------|--------------|
| 1 | GGOEGOAQ012899 | 456 |
| 2 | GGOEGDHC074099 | 334 |
| 3 | GGOEGOCB017499 | 319 |
| 4 | GGOEGGCC077999 | 290 |
| 5 | GGOEGFYQ016599 | 253 |
| 6 | GGOEGOCB078299 | 250 |
| 7 | GGOEGHPJ080310 | 189 |
| 8 | GGOEADHH073999 | 167 |
| 9 | GGOFRGAX00037 | 146 |

Results per page: 50 1 - 50 of 463

Modify the previous query to add a filter to limit the results to just 08/02/2017

```
SELECT * FROM ecommerce.sales_by_sku_2017* WHERE _TABLE_SUFFIX = '0802'
```



The screenshot shows the Google BigQuery web interface. At the top, there's a tab bar with several tabs, including 'Untitled query'. Below the tabs, the query editor shows the following SQL query:

```
1 SELECT * FROM `ecommerce.sales_by_sku_2017*`  
2 WHERE _TABLE_SUFFIX = '0802'
```

Below the query editor, a green checkmark indicates 'Query completed'. The 'Query results' section is active, showing a table with the following data:

| Row | productSKU | total_ordered |
|-----|----------------|---------------|
| 1 | GG0EGHPA002910 | 101 |

At the bottom of the interface, there's a footer showing 'Results per page: 50' and '1 - 1 of 1'.

Key Learnings

1. **Data Exploration:** Effectively analyzed product sentiment data to understand customer perceptions
2. **SQL Joins:** Successfully combined sales and inventory data to create actionable insights
3. **Data Appending:** Implemented both UNION and table wildcard techniques for combining historical data
4. **Inventory Management:** Developed queries to identify products needing urgent restocking

Conclusion

This project successfully demonstrated how to create a comprehensive data warehouse in BigQuery by joining disparate datasets and appending historical records. The resulting tables provide valuable insights into product performance, inventory needs, and customer sentiment - enabling data-driven decision making for the ecommerce business.

