Creating a Data Warehouse Through Joins and Unions

Open the BigQuery console

1. In the Google Cloud Console, select **Navigation menu** > **BigQuery**.

The **Welcome to BigQuery in the Cloud Console** message box opens. This message box provides a link to the quickstart guide and the release notes.

2. Click **Done**.

The BigQuery console opens.

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**.

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`
```

Note: This is only for you to review, the queries in this lab will be using the `data-to-insights` project.

2. Click on the **ecommerce** dataset to display the `products` table.

Examine the data using Preview and Schema tabs

1. Navigate to the **ecommerce** > **products** dataset and click the **Preview** tab to see the data.
2. Click the **Schema** tab.

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

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

Possible solution:

```
SELECT
  SKU,
  name,
  sentimentScore,
  sentimentMagnitude
FROM
  `data-to-insights.ecommerce.products`
ORDER BY
  sentimentScore DESC
LIMIT 5
```

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

Possible solution:

```
SELECT
  SKU,
  name,
  sentimentScore,
  sentimentMagnitude
FROM
  `data-to-insights.ecommerce.products`
WHERE sentimentScore IS NOT NULL
ORDER BY
  sentimentScore
LIMIT 5
```

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

1. 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

Possible solution:

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

2. Click on the `sales_by_sku` table, then click the **Preview** tab.

Next, enrich your sales data with product inventory information by joining the two datasets.

Join sales data and inventory data

1. Using a JOIN, enrich the website ecommerce data with the following fields from the product inventory dataset:

- `name`
- `stockLevel`
- `restockingLeadTime`
- `sentimentScore`
- `sentimentMagnitude`

2. Complete the partially written query:

```
# join against product inventory to get name
SELECT DISTINCT
  website.productSKU,
  website.total_ordered,
  inventory.name,
  inventory.stockLevel,
  inventory.restockingLeadTime,
  inventory.sentimentScore,
  inventory.sentimentMagnitude
FROM
  ecommerce.sales_by_sku_20170801 AS website
  LEFT JOIN `data-to-insights.ecommerce.products` AS inventory

ORDER BY total_ordered DESC
```

Possible solution:

```
# join against product inventory to get name
SELECT DISTINCT
  website.productSKU,
  website.total_ordered,
  inventory.name,
  inventory.stockLevel,
  inventory.restockingLeadTime,
  inventory.sentimentScore,
  inventory.sentimentMagnitude
FROM
  ecommerce.sales_by_sku_20170801 AS website
  LEFT JOIN `data-to-insights.ecommerce.products` AS inventory
  ON website.productSKU = inventory.SKU
ORDER BY total_ordered DESC
```

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

Possible solution:

```
# calculate ratio and filter
SELECT DISTINCT
  website.productSKU,
  website.total_ordered,
  inventory.name,
  inventory.stockLevel,
  inventory.restockingLeadTime,
  inventory.sentimentScore,
  inventory.sentimentMagnitude,

  SAFE_DIVIDE(website.total_ordered, inventory.stockLevel) AS ratio
FROM
  ecommerce.sales_by_sku_20170801 AS website
  LEFT JOIN `data-to-insights.ecommerce.products` AS inventory
  ON website.productSKU = inventory.SKU

# gone through more than 50% of inventory for the month
WHERE SAFE_DIVIDE(website.total_ordered,inventory.stockLevel) >= .50

ORDER BY total_ordered DESC
```

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

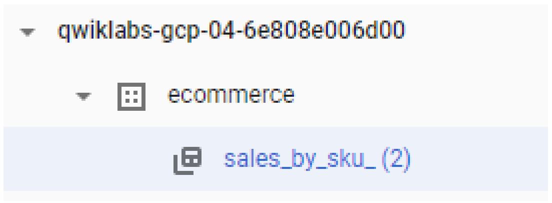
1. For the schema, specify the following fields:

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

Possible solution:

```
CREATE OR REPLACE TABLE ecommerce.sales_by_sku_20170802
(
productSKU STRING,
total_ordered INT64
);
```

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:



3. Insert the sales record provided to you by your sales team:

```
INSERT INTO ecommerce.sales_by_sku_20170802
(productSKU, total_ordered)
VALUES('GGOEGHPA002910', 101)
```

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`

```
SELECT * FROM ecommerce.sales_by_sku_20170801
UNION ALL
SELECT * FROM ecommerce.sales_by_sku_20170802
```

Note: The difference between a `UNION` and `UNION ALL` is that a `UNION` will not include duplicate records.

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.

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

Possible solution:

```
SELECT * FROM `ecommerce.sales_by_sku_2017*`
```

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

Possible solution:

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

Note: Another option to consider is to create a Partitioned Table which automatically can ingest daily sales data into the correct partition.