



## Activity Overview

In this activity, you will use BigQuery to partition data and create an index. Partitions and indexes help you optimize a database by creating shortcuts to specific rows and dividing large datasets into smaller, more manageable tables. By creating partitions and indexes, you can make faster and more efficient databases. This will make it easier to pull your data when you need to analyze or visualize it.

Be sure to complete this activity before moving on. The next course item will provide you with a completed exemplar to compare to your own work. You will not be able to access the exemplar until you have completed this activity.

### Scenario

Review the following scenario. Then complete the step-by-step instructions.

You are a BI analyst for a grocery store chain that monitors dietary trends affecting in-store purchases. Your company wants you to examine which types of Hass avocados are purchased most often. The avocados are categorized as one of four sizes: small, medium, large, and extra large. In addition to the average price and total volume of each avocado, the date of each sale is also recorded.

Using this data, you will create a historical table that illustrates how indexes and partitions work. This will allow you to practice creating partitions and clustered tables and demonstrate how to use them.

Your goal is to use partitions and clusters to answer the following question: What is the distribution of avocado sales from 2015 to 2021?


### Step-By-Step Instructions

Follow the instructions to complete each step of the activity. Then, answer the questions at the end of the activity before going to the next course item to compare your work to a completed exemplar.

#### Part 1: Set up in BigQuery

##### Step 1: Access the data

To use the data for this course item, download the dataset from Kaggle - Avocado Sales 2015-2021 (US centric).

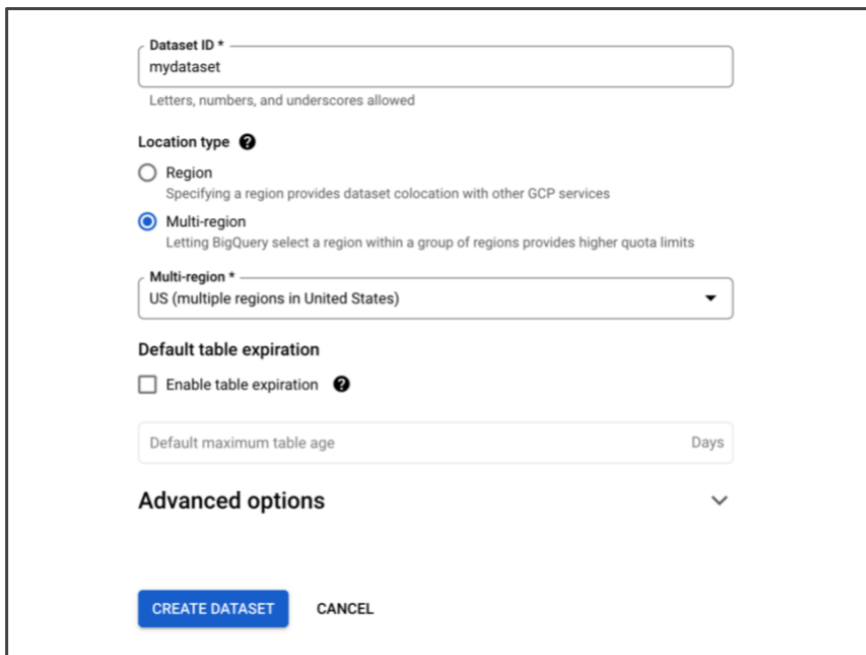
 [avocado cleaned CSV File](#)

## > Step 2: Open the BigQuery console

Navigate to your [BigQuery console](#).

## > Step 3: Create a dataset

In the Explorer menu, find one of your projects. It may be titled "My First Project" or a title you gave it. Click the three-dot icon, then select Create dataset. Fill in "mydataset" for the Dataset ID and set the location to us (multiple regions in United States). Then select Create dataset.



Dataset ID \*

mydataset

Letters, numbers, and underscores allowed

Location type ?

☐ Region  
Specifying a region provides dataset colocation with other GCP services

☒ Multi-region  
Letting BigQuery select a region within a group of regions provides higher quota limits

Multi-region \*

US (multiple regions in United States)

Default table expiration

☐ Enable table expiration ?


Default maximum table age Days

Advanced options

CREATE DATASET CANCEL

## > Step 4: Load the avocado data into a table

Next to mydataset, click the three-dot icon and select Create table.

▶  mydataset



Open

Open in ▶

Create table

Share

Copy ID

Delete

Next, use the Create table from the dropdown menu and select Upload. Choose the CSV file you downloaded earlier in this activity.

## Source

Create table from

Upload

Select file \*

avocado cleaned.csv

File format

CSV

Then, name the table avocado\_base." Make sure the Dataset field reads "mydataset" and the Table type field reads "Native table."

Dataset \*
mydataset

Table \*
avocado\_base

Unicode letters, marks, numbers, connectors, dashes or spaces allowed.

Table type
Native table

n the Schema section of the interface, check the box for Auto detect.  
Then select Create table.

Schema

☒ Auto detect

**i** Schema will be automatically generated.

Partition and cluster settings

Partitioning

No partitioning

Clustering order

Clustering order determines the sort order of the data. Clustering can be used on both partitioned and non-partitioned tables.

Part 2: Create tables with partitions and clusters


>Step 1: Create a table without a partition or cluster

To begin, create a new table without a partition or cluster. This will serve as a baseline to compare to the partitioned and clustered tables. Name it avocados.

Then, in the Editor tab, copy and paste the following SQL code and click Run.

```
1 CREATE TABLE
2   `mydataset.avocados`
3 AS (
4   SELECT
5     *
6   FROM `mydataset.avocado_base`
7 );
```

When you finish running the code, switch to the Results tab. Click Go to table and take note of the Details pane. Save the details for later by taking a screenshot or copying and pasting the information into another document. The dates on your screen might differ, but the table size, long-term storage size, and number of rows should be the same as in the following image.

Table info	
Table ID	my-first-project-379816.mydataset.avocados
Created	Mar 6, 2023, 11:04:27 AM UTC-6
Last modified	Mar 6, 2023, 11:04:27 AM UTC-6
Table expiration	May 5, 2023, 12:04:27 PM UTC-5
Data location	US
Default collation	
Case insensitive	false
Description	
Labels	
Storage info 	
Number of rows	41,025
Total logical bytes	4.37 MB
Active logical bytes	4.37 MB
Long term logical bytes	0 B
Total physical bytes	0 B
Active physical bytes	0 B
Long term physical bytes	0 B
Time travel physical bytes	0 B

>
 Step 2: Create a table with a partition

Next, create a table partitioned by an integer range (the years 2015 through 2022). Name it `avocados_partitioned`. Return to the tab you entered the SQL code into. Delete that code then copy and paste the following SQL code. Click Run.

```

1 CREATE TABLE
2   `mydataset.avocados_partitioned`
3 PARTITION BY
4   RANGE_BUCKET(Year, GENERATE_ARRAY(2015,2022,1))
5 AS (
6   SELECT
7     *
8   FROM `mydataset.avocado_base`
9 );
```

When you finish running the code, switch to the Results tab. Click Go to table and take note of the Details pane. Save the details for later by taking a screenshot or copying and pasting the information into another document. After this activity, you'll compare this to the exemplar.

>
 Step 3: Create a table with a partition and a cluster

Next, create a table partitioned by an integer range and clustered by type. Name it `avocados_clustered`. Return to the tab where you entered the SQL code. Delete that code, then copy and paste the following SQL code. Click Run.

```

1 CREATE TABLE
2   `mydataset.avocados_clustered`
3 PARTITION BY
4   RANGE_BUCKET(Year, GENERATE_ARRAY(2015,2022,1))
5 CLUSTER BY
6   type
7 AS (
8   SELECT
9     *
10  FROM `mydataset.avocado_base`
11 );
```

When you finish running the code, switch to the Results tab. Click Go to table and take note of the Details pane. Save the details for later by taking a screenshot or copying and pasting the information into another document. After this activity, you'll compare this to the exemplar.

Part 3: Query the tables and compare performance

>
 Step 1: Query the table without a partition or cluster

Delete the code in the Editor tab, then copy and paste the following code. Click Run to query `avocados`—the table without partition or cluster.

```

1 SELECT
```

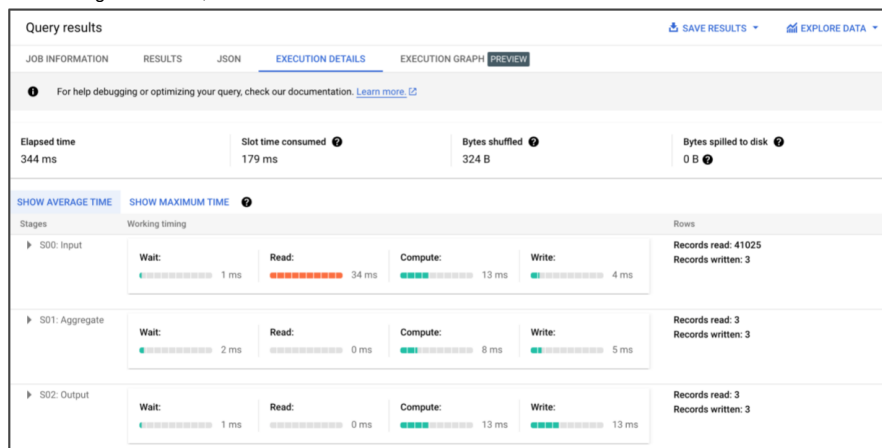
```

2      year,
3      COUNT(*) AS number_avocados,
4      SUM(TotalVolume) AS sum_totalVolume,
5      SUM(AveragePrice) AS sum_AveragePrice
6 FROM `mydataset.avocados`
7 WHERE type = 'organic'
8 GROUP BY year
9 ORDER BY year ASC;

```

When the query has finished running, check the Execution details tab. This explains that the number of records read is the total number of records in the table. In this query, the database processes all records from the table. This is reflected in S00:Input.

Note: The Working timing section on your screen might vary in color or duration. Your SQL query might take longer or shorter to run depending on differing BigQuery engine server speeds. Your screen might not match the following screenshot, but the records read and records written should match with the Rows section.



In the next steps, take note of the S00 and S01 rows as described in the preceding screenshot. You will need to compare these details to the exemplar.

## > Step 2: Query the partitioned table

Delete the code in the Editor tab, then copy and paste the following code. Click Run to query `avocados_partitioned`—the table that is partitioned by an integer range.

```

1 SELECT
2     year,
3     COUNT(*) AS number_avocados,
4     SUM(TotalVolume) as sum_TotalVolume,
5     SUM(AveragePrice) as sum_AveragePrice
6 FROM `mydataset.avocados_partitioned`
7 WHERE type = 'organic'
8 GROUP BY year
9 ORDER BY year ASC;
10

```

When the query has finished running, check the Execution details tab and save a screenshot of it. You'll need to compare these details to the exemplar.

## > Step 3: Query the partitioned and clustered table

Delete the code in the Editor tab, then copy and paste the following code. Click Run to query `avocados_clustered`—the table that is partitioned by an integer range and clustered by type.

```

1 SELECT
2     year,
3     COUNT(*) AS number_avocados,
4     SUM(TotalVolume) as sum_TotalVolume,
5     SUM(AveragePrice) as sum_AveragePrice
6 FROM `mydataset.avocados_clustered`
7 WHERE type = 'organic'
8 GROUP BY year
9 ORDER BY year ASC;
10

```

When the query has finished running, check the Execution details tab and save a screenshot of it. You will need to compare these details to the exemplar.

### What to Include in Your Response

You should record the following in your SQL code results:

A screenshot of the Details pane of the `avocados_partitioned` table

A screenshot of the Details pane of the `avocados_clustered` table

A screenshot of the Execution Details pane of the `avocados_partitioned` table

A screenshot of the Execution Details pane of the `avocados_clustered` table

In addition to this criteria, in a business role you might consider including a report that describes the distribution of avocados over the six-year time period and if there are any relationships between avocado size, type, and total volume sold. You could also share your recommendations based on any trends you find in the data, in order to anticipate future demand.

1. Did you complete this activity? 1 / 1 point

- ☒ Yes  
☐ No

✓ Correct  
Thank you for completing this activity! Using clusters and partitions will help you optimize your database performance. Your data will be faster and simpler to work with in your BI analyses. Please complete the following quiz questions and review the feedback. Then go to the next course item to compare your work to a completed exemplar.

2. How can partitions and indexes help optimize a database? Select all that apply. 1 / 1 point

- ☒ Divide large datasets into smaller, more manageable tables
- ✓ Correct  
Partitions and indexes help optimize a database by creating shortcuts to specific rows and dividing large datasets into smaller, more manageable tables. In addition, partitions and indexes enable faster and more efficient databases, which makes it easier to pull data for analysis or visualization.
- ☒ Enable a faster and more efficient database
- ✓ Correct  
Partitions and indexes help optimize a database by creating shortcuts to specific rows and dividing large datasets into smaller, more manageable tables. In addition, partitions and indexes enable faster and more efficient databases, which makes it easier to pull data for analysis or visualization.
- ☐ Use a system's memory to save time retrieving frequently used data
- ☒ Create shortcuts to specific rows
- ✓ Correct  
Partitions and indexes help optimize a database by creating shortcuts to specific rows and dividing large datasets into smaller, more manageable tables. In addition, partitions and indexes enable faster and more efficient databases, which makes it easier to pull data for analysis or visualization.

3. Fill in the blank: A query on a table with partitions and indexes processes \_\_\_\_\_ a table that is not partitioned or indexed. 1 / 1 point

- ☒ fewer records than  
☐ more records than  
☐ the same number of records as

✓ Correct  
Together or individually, partitions and clusters restructure the data in a way so that only the relevant records are scanned when a query is run. Therefore, a query on a table with partitions and indexes processes fewer records than a table that is not partitioned or indexed, optimizing database performance while minimizing processing costs.