Google BigQuery
Lab 2

1 BigQuery documentation

Main official documentation page for BigQuery.

Top level overview of BigQuery

Basics of organization of resources in BigQuery, including datasets and tables.

Basics of working with BigQuery via the console UI.

Quick start guide for working with loading and querying data as well as querying public datasets.

Official reference for BigQuery SQL query syntax

2 Lab setup with project, dataset and tables

We will be using the same project: `first-bigquery-project` and dataset (`first_dataset`) as well as tables (`samplesales`, `salestime`, `salary`) that we created in a previous lab and build from there.

3 Basic Data Manipulation Language statements

Up to this point of time, we have created a complete table using an uploaded CSV and performed queries of varying complexity to extract and summarize data from this table. Here, we will look at some of the most common operations you can perform on a table once it is created from an uploaded CSV (or any of the other formats available such as Avro, Parquet or ORC).

In practice, most of the time, you will perform any required modifications on the table data outside of Google BigQuery (for example in a Google Sheet or Excel spreadsheet) and then reimport it as table

into Google BigQuery, as this is usually simpler than modifying the table directly inside Google BigQuery using SQL statements. We will demonstrate this here for the purpose of completeness.

3.1 Using INSERT to add new rows to table

We will work with the `salary` table in `first_dataset`. If you have not created this table yet, create it using the instructions from the previous lab.

At the start, if you do a preview of this table it should only have 5 rows in it:

Schema Details Preview Table Explorer Preview

Row	EmployeeID	Name	MonthlySalary	DaysWorked
1	2	Muthu	0	0
2	4	Jane	0	0
3	3	Ah Chong	1000	10
4	1	Ali	2000	20
5	5	Matthew	8000	40

You can add a single new row by at the end of the table by specifying the columns of the new row and the values for these columns:

```
INSERT INTO first_dataset.salary(EmployeeID, Name,
MonthlySalary, DaysWorked)
VALUES(6, 'Peter', 10000, 20);
```

Now if you close the table preview and open it again in the console UI, you should be able to see the latest added row.
Alternatively, you can just view all the rows in the table with:

```
SELECT * FROM first_dataset.salary;
```

The newly added row may either appear at the top or bottom of the table, depending on which approach you use.

If you are adding in new values into a new row in the same sequence as the original column names, you do not need to specify the column names in your INSERT statement, for e.g.:

```
INSERT INTO first_dataset.salary
VALUES(7, 'Jane', 30000, 5);
```

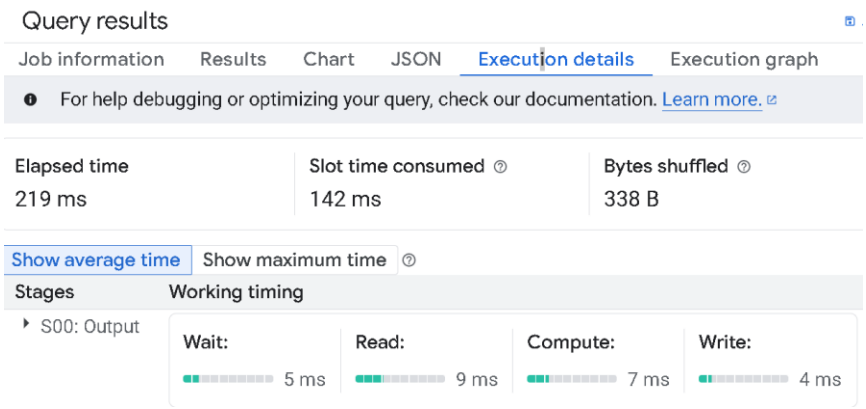
Check that the new row is added with either a new table preview or the SELECT * statement.

You can add multiple new rows with new values by separating the successive value sets with commas, for e.g.

```
INSERT INTO first_dataset.salary
VALUES (8, 'Ahmad', 20000, 2),
(9, 'Siva', 10000, 4),
(10, 'Lim', 40000, 8);
```

Check that the new rows are added with either a new table preview or the SELECT * statement.

You will notice that the job execution time required to insert a few new rows into an existing table is quite long, which is due to the underlying architecture of BigQuery.



BigQuery as a cloud data warehouse stores data in a columnar format and distributes it across multiple nodes and storage systems. While this architecture is highly optimized for large-scale reads (e.g., SELECT queries across billions of rows), it's not optimized for low-latency row-level writes, especially for small volumes.
This is why its more efficient to perform your modifications in a tool / application outside of Google BigQuery (for e.g. Excel or Google Sheets) and then import the modified data into a new table in BigQuery when you are done.

You can insert a row with one or more valued that are computed with a subquery that is executed on that table or some existing table, for e.g.

```
INSERT INTO first_dataset.salary
VALUES (11, 'James',

  (SELECT MonthlySalary FROM first_dataset.salary
  WHERE Name = 'Ali'),

  (SELECT DaysWorked FROM
  first_dataset.salary
  WHERE Name = 'Peter')
);
```

Check that the new rows are added with either a new table preview or the SELECT * statement.

3.2 Using UPDATE to update existing row values in a table

We use the UPDATE statement in conjunction with the WHERE clause to specify the specific rows in a table to update.

For e.g. to update the MonthlySalary and DaysWorked column values for employee with the name of Matthew:

```
UPDATE first_dataset.salary
SET MonthlySalary = 12000,
```

```
DaysWorked = 10
WHERE Name = 'Matthew';
```

Check that the specified rows are updated with either a new table preview or the SELECT * statement.

You can also update the values of columns in multiple rows selected via the WHERE clause.

For e.g. to increase the monthly salary of all employees who worked 10 or more days in a month, we can type:

```
UPDATE first_dataset.salary
SET MonthlySalary = CAST((MonthlySalary * 1.5) AS INT64)
WHERE DaysWorked >= 10;
```

The CAST function is necessary to change the result of the expression `MonthlySalary * 1.5` (which will be a float) back to an integer to be stored in the MonthlySalary column, which is originally of type Integer.

Check that the specified rows are updated with either a new table preview or the SELECT * statement.

3.3 Using DELETE to delete existing rows in a table

We use the DELETE statement in conjunction with the WHERE clause to specify the specific rows in a table to delete.

An example to delete a single row returned by a WHERE clause condition:

```
DELETE first_dataset.salary
WHERE Name = 'Muthu';
```

Check that the specified rows are deleted with either a new table preview or the SELECT * statement.

To delete multiple rows returned by a WHERE clause condition:

```
DELETE first_dataset.salary
WHERE DaysWorked < 10;
```

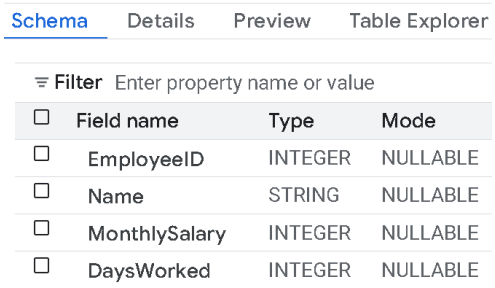
Check that the specified rows are deleted with either a new table preview or the SELECT * statement.

3.4 Using TRUNCATE to empty the contents of a table

To remove all rows from an existing table but preserve the table structure (i.e. the columns and their data types), use:

```
TRUNCATE TABLE first_dataset.salary;
```

Check that all the existing rows within the table have been deleted leaving it empty with either a new table preview or the SELECT * statement. However notice that the table still exists within the dataset and that you can still view its Schema.



To completely remove the table entirely from the dataset, you can use the specified option from the context menu available from the ellipsis menu next to the table in the Explorer view: