Table Partitioning Strategy

Objective

Whenever possible, all tables in BigQuery should be partitioned and/or clustered to improve query performance and to enable cost savings.

No partitioning of tables is required in the ***Staging*** dataset, however ***most*** or all tables within the ***ODS*** dataset should be partitioned and clustered for performance and costs.

The partitioning strategy for initial loading and merging should be configured to use the ingestion date, with clustering on the filename.

For curated tables and aggregations, the tables should be partitioned on a date or timestamp column based on the most applicable business use case. The column chosen for partitioning must be the column that would be most suitable for the majority of business use cases, though in general it should be ***TS_UTC*** (utc timestamp). Determining this column can be done by determining which column would be used within the ***WHERE*** clause in the query to filter data by date.

This column must be included in all ***VIEWS*** that reference the table, and must be in the description of the table, and communicated to the ***Product*** team members so they use this column for the ***date*** filtering option in their applications (Qlik, Tableau, Dataiku).

Clustering of columns can be done using the same methodology based on evaluating which columns would be most used for general business use cases.

It should be noted that partitioning and clustering is not initially required. It is possible to easily "convert" or create a partitioned and clustered table from a table that has not been configured. See converting tables section below.

Google Reference

As with all things IT related, they change over time. This document references information taken from Google's Support pages. Always think to verify the source documentation as it may be more up to date than this document.

For partitioning options please refer to this site: https://cloud.google.com/bigquery/docs/partitioned-tables#types_of_partitioning

For Clustering, please refer to this site: https://cloud.google.com/bigquery/docs/clustered-tables#when_to_use_clustering

Partitioning Overview

The idea of partitioning tables is a method of dividing up data within a larger table, into smaller "partition" tables. Partitioning a table separates data on the physical level. Doing so will improve query performance time when doing a where clause on the partition date.

Partitioning by time column (***Creation_date***) by day

The time column used should reflect the most useful business use case that would be used in a filter in a report (in a production table). For intermediate tables, it would those that would be commonly used in the ***WHERE*** clause when generating the production data, ie. SELECT X, Y from table where Creation_date < 2 months || or where Creation_date = "20180303"

In either case, the amount of data scanned will correspond to the physical amount of data in each partition. In the case where the table is not partitioned, the complete table would be scanned.

Partitioning Configuration Options

As previously stated, there are few reasons not to use partitioning or clustering in a table, but before implementing partitioning or clustering, it's necessary to understand the data that will be contained in the table and which (time) column is the best business case to partition on, or which column(s) the data would be better clustered on.

https://cloud.google.com/bigquery/docs/partitioned-tables#when_to_use_partitioning

Ingestion Time Partition (automatic)

Data is partitioned on a pseudo column called _***PARTITIONTIME***. This column is not visible from the table schema, but it is present. Very useful when receiving data from upstream systems or streaming systems where we want to be able to get data based on a load time into the database, and not from a date or time from within a column.

Time Unit Column Partitioning

This option should be used on modeled data based on date for a specific business use case. We partition the table on a column that contains a date, and then over a specific unit of time (day, month, year)

Example

where date(meta_Business_date) < DATE_SUB(CURRENT_DATE(), INTERVAL 2 MONTH)

Integer Range Partitioning

Possibly more limiting in terms of configuration, and might not be the most adapted. It requires knowledge of the range of possible values in a column, which might not always be known.

There is a limitation of 10,000 ranges for integer partitions.

Require Partition Filter

Just because a table is ***partitioned*** does not require the users or any systems to generate queries that filter the data in the ***WHERE*** clause in a query. On tables with a small volume of data, there will be little cost savings or performance improvement when using filtering on the partition column. However, requiring the partition column in the ***WHERE*** clause can lead to improved cost savings and performance increase on tables with a large volume of data, or even small tables that are queried frequently.

Special Note: While introducing this requirement in intermediary tables should not cause problems as data engineer applications and developers should be able to adapt their queries to always include the partition filter. However not all DataViz or Data Science tools may be compatible with this option so it must be evaluated for production tables, or at least provided in the documentation so they can adapt their applications.

Partition Expiration

In some cases, (business / security / GDPR) it is not necessary to keep data beyond a specific amount of time. Tables that are partitioned (with the exception of those partitioned on an integer range) can be configured to be deleted after a certain amount of days.

Limitations

Partition Strategy is Key, this will depend on the amount of history, and the types of partitions that we need.

4000 partitions in each partition table. This is a hard limit that cannot be increased.

If partitioning by day (4000/365) that gives us about 11 years.

This must be considered when

Ingestion Partitioning: Normally only for new data.

This gives us 11 years into the future, assuming Google doesn't increase this in the future.

Clustering

When there are a lot of values in a column (like a primary key) that could be grouped together to improve query response time or save costs. Clustering doesn't promise cost saving, but does offer performance because data is sorted or "clustered" on the column value.

Partitioning by time column (***Creation_date***) and clustering on column ***Tags***

The difference now is that the data is sorted in the partitions on the Tags column, which ***can*** speed up query times and sometimes improve cost saving.

Another example taken from Google's support site:

Table Sharding

In the majority of cases, it is preferred to use a type of partitioning on a table. There does exist another “legacy” feature in BigQuery which is table sharding. This is where multiple tables are regrouped together using a common table prefix, with the table suffix being a date.

gcp_project_id.bigquery_dataset.sharded_table_20230401 for example. With ***sharded_table_*** being the prefix and ***20230401*** being the suffix.

This could be useful for keeping a history of full table load as part of a data engineering pipeline.

There is a limitation of only ***1000*** sharded tables that may be queried at a time, whereas with a regular partition table it’s possible to have up to ***4000*** partitions.

Converting Tables

During the initial configuration of a project, it may not be possible to have a clear path on what columns it is best to partition the data or which column to cluster the data on. However it’s possible to easily recreate tables that are not partitioned or clustered into partitioned and clustered tables.

Note: This isn’t a true table conversion, The original table is not being modified in any way. It is creating a new table, with partitioning or clustering options. This requires scanning the entire table and will incur the costs for reading all the data, so it is best to implement a partitioning and clustering strategy before the table size grows very large.

Non Clustered Table to Clustered Table

```
CREATE OR REPLACE TABLE

`prj-do-domain-poc.ds_sample.ODS_0000_F006_F_D_IDBS_EWB_CORE_ENTITIES_clustered`

CLUSTER BY

  meta_Run_id,

  meta_filename AS

SELECT

  *

FROM

  `prj-do-domain-poc.ds_sample.ODS_0000_F006_F_D_IDBS_EWB_CORE_ENTITIES`
```

Non Partitioned to Partitioned

```
CREATE OR REPLACE TABLE

  `prj-do-domain-poc.ds_sample.ODS_0000_F006_F_D_IDBS_EWB_CORE_ENTITIES_partitioned`

PARTITION BY

  DATE(meta_Business_date)

 AS

SELECT

  *

FROM

  `prj-do-domain-poc.ds_sample.ODS_0000_F006_F_D_IDBS_EWB_CORE_ENTITIES`
```

Non Partitioned/Clustered to Partitioned & Clustered

```
CREATE OR REPLACE TABLE

`prj-do-domain-poc.ds_sample.ODS_0000_F006_F_D_IDBS_EWB_CORE_ENTITIES_clustered_partitioned`

PARTITION BY

  DATE(meta_Business_date)

CLUSTER BY

  meta_Run_id,

  meta_filename AS

SELECT

  *

FROM

  `prj-do-domain-poc.ds_sample.ODS_0000_F006_F_D_IDBS_EWB_CORE_ENTITIES`
```

Results of different configurations

The following examples are all based on an initial table, creating partitioned, clustered, and partitioned+clustered tables and running the same queries on the tables to show the difference in performance & costs. (Performance gain will be very little as the table is not very large.

Results of different configurations

Querying a regular table that is not partitioned.