Introduction to Tables in BigQuery

In BigQuery, a table is a resource that resides in datasets. It contains records organised in rows with each record has columns (fields). Each table is defined by a schema which contains column names, data types etc.

Data types

|  |  |  |  |
| --- | --- | --- | --- |
| Numeric types | String types | Temporal types | Complex types |
| Int64 | String | Date | Array |
| Float64 | Bytes | Datetime | Struct |
| Numeric | geography | Time |  |
| Bignumeric |  | Timestamp |  |
| bool |  |  |  |

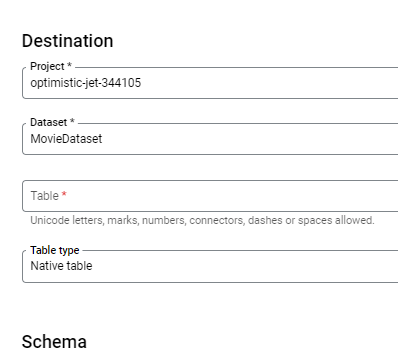
BigQuery supports to querying data that stores in

* BigTable
* Google drive
* Cloud storage

Difference between Traditional RDBMS and BigQuery

|  |  |
| --- | --- |
| Traditional RDBMS storage | BigQuery storage |
| Postgres, MySql etc. stores data in row by row where each row represents a single record. We can call it as record oriented storage. | * Supports Column storage * Data is stored in a columnar format rather than rows. |
| Advantage   * It is easy to change a particular record depending on a condition. But the disadvantage is: we have to check for each row/record for a particular query. | Advantage   * In a transaction, instead of going through each record we can access only those columns that are relevant for a particular query. |

Types of Tables

* Native table
  + Are tables that we import the full data inside Google BigQuery like database system.
  + 
* **External table**
  + Data is not stored inside Google BigQuery, but takes the reference of data from an external source such as Data Lake.
  + Introduction to External data source
* An external data source is a data source that you can query directly from BigQuery, even though the data is not stored in BigQuery storage.
* BigQuery supports the following external data sources:
  + *Bigtable*
  + *Cloud Spanner*
  + *Cloud SQL*
  + *Cloud Storage*
  + *Drive*
* BigQuery has two different mechanisms for querying external data:

1. External tables

* It acts like a standard BigQuery table. The table metadata including schema is stored in BigQuery storage, but the data itself resides in the external source.
* We can use external tables with the following data sources:
  + *Bigtable*
  + *Cloud Storage*
  + *Drive*

1. Federated queries

* A federated query is a way to send a query statement to an external database and get the result back as a temporary table.
* Federated queries use the BigQuery Connection API to establish a connection with the external database.
* We can use federated queries with the following external databases:
  + Cloud Spanner
  + Cloud SQL
  + Advantage
    - Data retrieval is faster because of columnar form and compactness.
* Disadvantage
  + Queries against external tables are comparably slow as compared to native tables if the files are very big.
  + It could be better if the data is split into small files and keep them in a *bucket in* Google Cloud Storage.
* **Internal table**
  + Is a temporary table. We need an internal table when we need to retrieve data from database tables.
  + These tables occupy memory only at run-time and not at the time of their declaration.
  + The size of an internal table is not fixed. It can be changed according to the requirement of the program.
* Advantage
  + Internal tables can be used to hold results which can be used later in the program
  + Accessing or querying is faster for internal tables rather than database table.
  + Example : *Create a list of contact numbers of various customers from several tables.*
    - So the user first creates an internal table, selects the relevant data from customer tables and then places the data in the internal table. Other users can access and use this internal table directly to retrieve the desired information, instead of writing database queries to perform each operation during the run-time of the program.
* Normal table

Normalization is a process to normalize the data in database. This is useful to make database more flexible by eliminating redundancy.

Storage optimization

* Partitioning
  + Divide large tables into smaller chunks or tables which are stored in separate memory locations.
  + Where to use?
    - Having large amount of data with low number of distinct values OR

The cardinality of the number of values in a column or group of columns is large.

* + - Advantage : improves the query performance and controls the cost by reducing number of bytes read by a query.
    - Limitation: slow the query process if we create lots of partitions.
  + BigQuery partitions data based on
    - Ingestion time(when BigQuery ingests the data)
      * Daily partitioning
        + Apply when data is continuously added over time or data is spread out over a wide range of dates.
      * Hourly partitioning
        + High volume of data with timestamp value is less than 6 months.
      * Monthly/yearly partitioning
    - Date/Timestamp column
      * Tables are partitioned based on a TIMESTAMP, DATE, or DATETIME column in the table.
    - Integer range (tables are partitioned based on an integer column)
    - NULL values…(contains row with null values in the partitioning column)
  + Partitioned table pricing
    - When we create and use partition tables in BigQuery, charges are based on how much data is stored in the partitions and the queries do we run against the data.
* Clustering
  + Clustering is applicable for smaller tables or when do we want to apply query statements to a sample of data taken from a large data table.
* Can use both clustering and partitioning altogether
* Clustering can improve the performance of certain types of queries such as queries that use filter clauses and queries that aggregate data.
* Disadvantage
  + To maintain clustering when data is changing. But BigQuery solves this problem by periodically clustering each table.

|  |  |
| --- | --- |
| Partitioning | Clustering |
| Divide large tables into smaller chunks or tables which are stored in separate memory locations. | Clustering is applicable for smaller tables or when do we want to apply query statements to a sample of data taken from a large data table. |
| Limitations | Cases where to prefer clustering over partitions |
| *Time-unit column-partitioned tables are subject to the following limitations:*   * Partitioning column must be either DATE, TIMESTAMP, or DATETIME column. * The partitioning column must be a top-level field. Cannot use a leaf field from a RECORD (STRUCT) as the partitioning column.   *Integer-range partitioned tables are subject to the following limitations:*   * The partitioning column must be an INTEGER column. * The partitioning column must be a top-level field. Cannot use a leaf field from a RECORD (STRUCT) as the partitioning column. | * If partitioning results in a large number of partitions beyond the limits on partitioned tables. |
|  | We can use both partitioning and clustering on the same column.   * Data is first partitioned and then data in each partition is clustered by the clustering columns. |
| Examples | Examples |
|  |  |