Database Technology Investigation: BigQuery

CST 363
Avner Biblarz

February 25, 2026

Glenn Bale Carreon
Jack DePizzo
Tim Shaker

Video Presentation:
https://youtu.be/lI5W1qFrlVI

Introduction

BigQuery is a fully-managed serverless cloud data warehouse developed by Google for enterprise data analytics. Unlike a traditional relational database management system (RDBMS), which is typically optimized for transactional workloads (OLTP), BigQuery is primarily designed for analytical processing (OLAP). Its purpose is to allow users to store and analyze massive datasets using standard SQL, without managing servers or infrastructure. BigQuery is not a replacement for a traditional RDBMS, but rather a platform for data analysis workloads, which can be used in conjunction with other databases, like PostgreSQL or MongoDB.

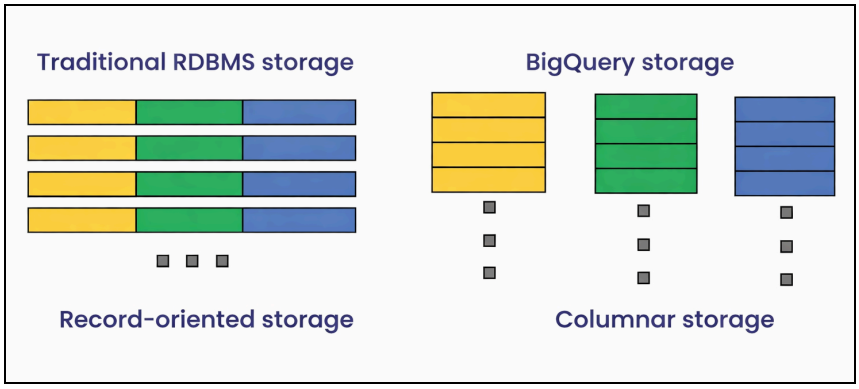
BigQuery Architecture & Core Concepts

The architecture of BigQuery is built on the separation of compute and storage, which is a common characteristic in modern data warehousing. Data is stored in an optimized columnar storage format which helps to improve query performance because only necessary columns are scanned during an execution of a query. Storage in BigQuery is managed independently from compute which helps users and organizations to scale their storage without using additional processing capacity. The separation of storage and compute provides elasticity and cost efficiency since compute resources are allocated dynamically. BigQuery uses distributed, massively parallel processing, also known as MPP, which is responsible for executing queries. Queries are broken into smaller tasks which then get distributed across many different workers that are operating in parallel. These workers process data concurrently and accumulate the results which are returned from a single query. This distributed approach enables BigQuery to scan terabytes of data in a quick manner. Being serverless, users do not need to manage clusters, nodes, and hardware, since Google Cloud handles all the resource allocation and optimization.

It is essential to understand a few core concepts to use BigQuery effectively. In BigQuery, resources are hierarchically organized into projects, datasets, and tables. Google Cloud Platform projects are the top-level container, which control access to resources and billing. Within a project, datasets contain tables and views, while tables store the actual data and schema. BigQuery improves query performance through partitioning and clustering, reducing the amount of data scanned and optimizing distributed execution. Partitioning is the process of dividing tables into segments, based on columns, which allows queries to scan only relevant partitions instead of the whole table, while clustering organizes data within partitions according to its specified columns which improves filtering and aggregation efficiency. These two techniques reduce the amount of data scanned, which improves query speed and decreases the cost of a query at the same time. This helps users and organizations to keep their cloud costs manageable, since pricing in BigQuery relies on the volume of data being processed.

BigQuery vs. Traditional RDBMS

Comparing a traditional relational database management system (RDBMS), like PostgreSQL or MySQL, to BigQuery highlights some significant differences in architecture and intended usage. A traditional RDBMS is row-oriented and optimized for online transactions processing (OLTP), where preserving data consistency during writes is crucial and involves handling high volumes of short, transactional queries like inserts, updates, and single-row lookups. BigQuery, in contrast, is column-oriented to improve query performance for online analytical processing (OLAP) workloads, which tend to read many rows but few columns. Additionally, BigQuery does not enforce table constraints for primary keys and foreign keys, since this would introduce additional complexity and overhead in a distributed system.

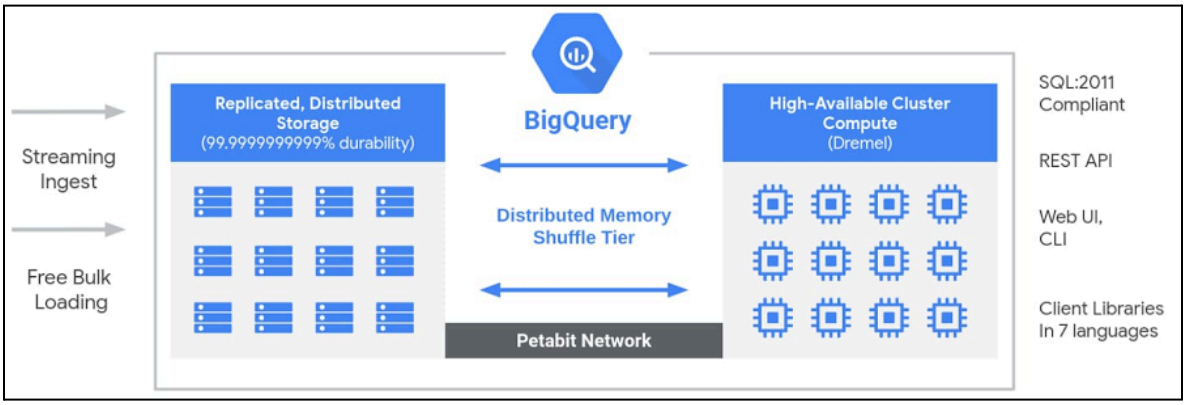


Traditional row-based storage vs. BigQuery's column-based storage

There are also some distinct differences in how BigQuery handles scaling compared to a traditional RDBMS, which tends to scale vertically or have complex requirements for horizontal scaling via sharding. BigQuery scales automatically and scaling is transparent to users, so the size of datasets can increase to petabytes without needing to redesign the database architecture. A traditional RDBMS also typically involves fixed infrastructure costs, whereas BigQuery offers on-demand pricing based on usage, allowing users to only pay for the resources they use, instead of paying for servers that might only be used infrequently.

BigQuery's Core Application

The primary application of BigQuery is large-scale data warehousing and analytics. It is designed to analyze extremely large datasets that would be inefficient or impractical to process in a traditional RDBMS. Consider a modern web application that records user interactions. Each page view, click, and API call might generate an event record. Over time, this system may accumulate billions of rows of data. A traditional RDBMS would struggle to handle a complex query on a dataset of this size, and analysis may involve data from multiple sources. BigQuery's distributed storage and compute model makes it possible to analyze petabytes of data across multiple tables and datasets with speed and efficiency.



BigQuery's serverless model offers extensive data ingestion methods and ways to interact with data

Besides the performance advantages offered by BigQuery's distributed architecture, BigQuery also provides powerful collaborative and data governance capabilities to support modern data teams. BigQuery integrates with Google Cloud's Identity and Access Management (IAM) system, enabling granular permissions at the project, dataset, table, or even column level, allowing organizations to control access to their data, as well as making it easier for them to share their data with the public. Additionally, BigQuery provides a rich feature set and tools, like data canvases, which allow teams to explore their data in an intuitive way, and notebooks, which make it easy to run Python code directly in a BigQuery environment. BigQuery also integrates with other data tools like Google Sheets, which connects with BigQuery to import and sync live data. More recently, AI features powered by Google Gemini have been integrated into BigQuery, allowing users to ask questions about their data with natural language prompts, where answers are linked to clear reasoning based on their data. These features allow organizations to centralize their data in a single platform, while also providing best-in-class tools and features to make the data more useful and accessible.

Appendix

Upserting Data in BigQuery

Background

BigQuery supports primary keys for query optimization, but doesn't enforce primary key constraints. This means that if you insert a record into BigQuery, then insert the same record, or insert an updated version of it, BigQuery won't stop you. This can make it easy for duplicate rows to accidentally end up in datasets, which can decrease the accuracy of data analysis insights, so it's important to avoid unintentional duplicates in data workflows.

An upsert is a common database technique, which creates a record if it doesn't already exist, or updates the record if it does exist, avoiding the creation of a duplicate. A common way to do this in BigQuery is to first load records into a staging table, then perform a SQL MERGE command to create or update the matching records in the target table.

Demo

The repository for this demo contains a Python script that upserts data into BigQuery from CSV files, which is a practical way to load data into BigQuery efficiently, while maintaining data integrity by avoiding duplicates. The script also manages altering BigQuery tables to add new columns for columns that are added to CSV files, using BigQuery's schema auto-detection to infer their types.

Getting Started

To run the Python script, you'll first need to authenticate with Google Cloud. You'll need to do this for both the user account and the application default credentials, which will be used by the BigQuery client library.

```
gcloud auth login
gcloud auth application-default login
```

You'll then need to select a GCP project to use:

```
gcloud config set project YOUR_PROJECT_ID
```

The gcloud project must also have the necessary APIs enabled.

```
gcloud services enable bigquery.googleapis.com
```

You can then install the required dependencies and run the Python script. It's recommended to use a virtual environment:

```
python -m venv .venv
source .venv/bin/activate  # On Windows use `.venv\Scripts\activate`
pip install -r requirements.txt
python upsert_csv.py
```