BigQuery for data warehouse practitioners

cloud.google.com/architecture/bigquery-data-warehouse

Updated June 2021

This document explains how to use BigQuery as a data warehouse. It maps common data warehouse concepts to those in BigQuery, and describes how to perform standard data warehousing tasks in BigQuery. This document is intended for people who manage data warehouses and big data systems.

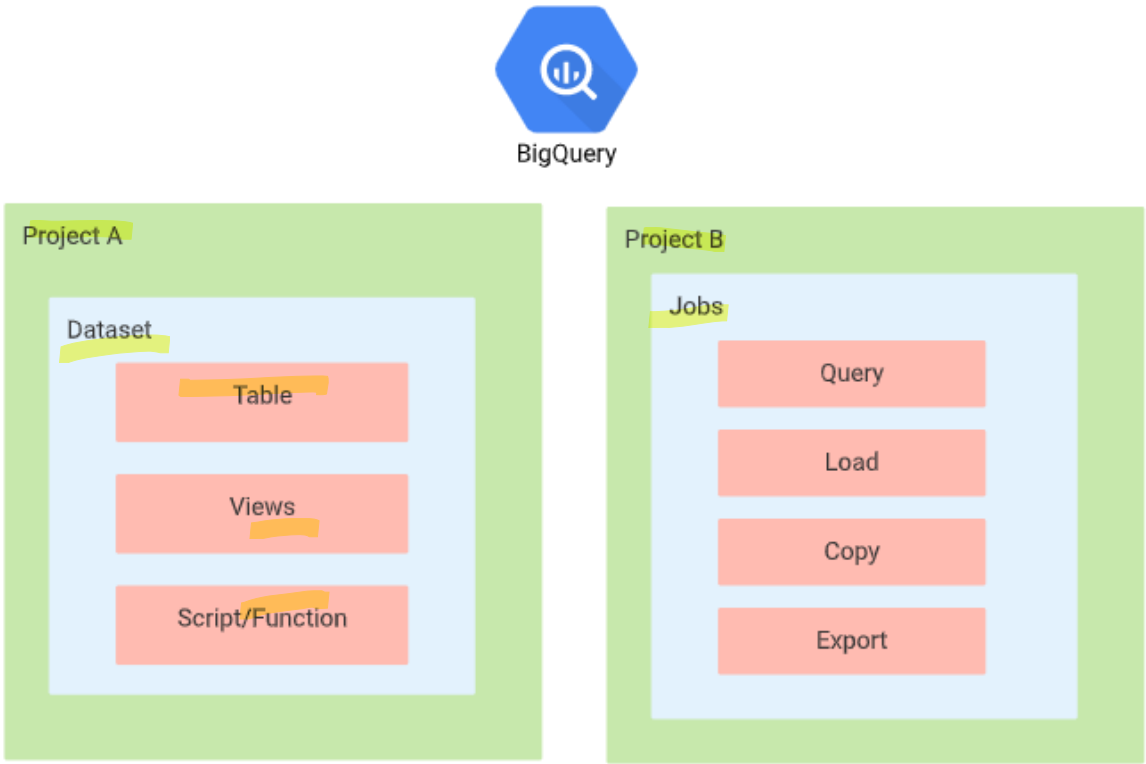
Service model comparison

The following table maps standard data warehouse concepts to those in BigQuery:

Data warehouse	BigQuery
Data warehouse	The BigQuery service replaces the typical hardware setup for a traditional data warehouse. That is, it serves as a collective home for all analytical data in an organization.
Data mart	Datasets are collections of tables that can be divided along business lines or a given analytical domain. Each dataset is tied to a Google Cloud project.
Data lake	Your data lake might contain files in Cloud Storage or Google Drive or transactional data in Bigtable or Cloud SQL. BigQuery can define a schema and issue queries directly on external data as federated data sources. The BigQuery Storage API offers high-bandwidth parallel reads and is compatible with common processing frameworks like Spark and Pandas.

BigQuery resources

BigQuery has a hierarchical structure. Its levels are outlined in the following diagram:



Projects

Any Google Cloud resources that you allocate and use must belong to a project. A project is the organizing entity for what you build with Google Cloud. In the context of BigQuery, a project is a container for all BigQuery resources. Since BigQuery decouples storage and compute, the projects that store and query the data can be different.

Datasets

Datasets are the top-level containers that you use to organize your BigQuery tables and views. They frequently map to schemas in standard relational databases and data warehouses.

Datasets are scoped to your Cloud project. When you reference a table from the command line, in SQL queries, or in code, you refer to it as follows:

```
project.dataset.table
```

A dataset is bound to a location. The dataset locations are as follows:

- **Regional**: A specific geographic place, such as London.
- **Multi-regional**: A large geographic area, such as the United States, that contains two or more geographic places.

You can only set the location for a dataset at the time of its creation. A query can contain tables or views from different datasets in the same location.

Using these multiple scopes (project, dataset, table, and location) can help you structure your information logically and geographically.

Tables

BigQuery tables are row-column structures that hold your data. Every table is defined by a schema that describes the column names, data types, and other information. You can specify the schema of a table when it's created. Alternatively, you can create a table without a schema and specify the schema in the query job or load job that first populates it with data. BigQuery has the following types of tables:

- **Native tables**: Tables backed by native BigQuery storage.
- **External tables**: Tables backed by storage external to BigQuery.
- **Views**: Virtual tables defined by a SQL query.

For more information, see Storage management.

Jobs

Jobs are actions that BigQuery runs on your behalf to load data, export data, query data, or copy data. Jobs are not linked to the same project that your data is stored in. However, the location where the job can execute is linked to the dataset location. For example, if you load data from a Cloud Storage bucket into a BigQuery dataset that's located in Singapore, the regional or multi-regional Cloud Storage bucket must also be located in Singapore. Alternatively, if your dataset is located in a European region, you can't query it from other regions, such as the US. This ensures that your data locality requirements are met.

Provisioning and system sizing

You don't need to provision resources before using BigQuery, unlike many RDBMS systems. BigQuery allocates storage and query resources dynamically based on your usage patterns:

- Storage resources are allocated as you consume them and deallocated as you remove data or drop tables.
- Query resources are allocated according to query type and complexity. Each query uses some number of slots, which are units of computation that comprise a certain amount of CPU and RAM.

You don't have to make a minimum usage commitment to use BigQuery. The service allocates and charges for resources based on your actual usage. By default, all BigQuery customers have access to 2,000 slots for query operations. You can also make slot reservations for your project. For details about which approach to use, see Costs.

Note: To start using BigQuery, you create a project to host your data, and then you enable billing. For instructions, see the BigQuery Quickstart.

Storage management

Internally, BigQuery stores data in a proprietary columnar format called Capacitor, which has a number of benefits for data warehouse workloads. BigQuery uses a proprietary format because it can evolve in tandem with the query engine, which takes advantage of deep knowledge of the data layout to optimize query execution. BigQuery uses query access patterns to determine the optimal number of physical shards and how they are encoded.

The data is physically stored on Google's distributed file system, called Colossus, which ensures durability by using erasure encoding to store redundant chunks of the data on multiple physical disks. Moreover, the data is replicated to multiple data centers.

You can also run BigQuery queries on data outside of BigQuery storage, such as data stored in Cloud Storage, Google Drive, or Bigtable, by using federated data sources. However, these sources are not optimized for BigQuery operations, so they might not perform as well as data stored in BigQuery storage.

Maintenance

BigQuery is a fully managed service, which means that the BigQuery engineering team takes care of updates and maintenance for you. Upgrades don't usually require downtime or hinder system performance.

Many traditional systems require resource-intensive vacuum processes to run at various intervals to reshuffle and sort data blocks and recover space. BigQuery has no equivalent of the vacuum or index management, because the storage engine continuously manages and optimizes how data is stored and replicated. Also, because BigQuery doesn't use indexes on tables, you don't need to rebuild indexes.

Backup and recovery

Managing backup and availability has always been a complex and expensive task for database administrators. The need for extra licenses and hardware can greatly increase costs. BigQuery addresses backup and disaster recovery at the service level. By maintaining a complete seven-day history of changes against your tables, BigQuery lets you query a point-in-time snapshot of your data by using either table decorators or `SYSTEM_TIME AS OF` in the `FROM` clause. You can easily revert changes without having to request a recovery from backups. When a table is explicitly deleted, its history is flushed after seven days. Additionally, the `cp` command offers instant in-region table snapshots.

BigQuery datasets can be regional or multi-regional. For regional datasets, for example a dataset located in the `us-central1` region, no copy of the dataset is maintained outside of the region. If you consider the lack of backups outside one region risky for your business, you can create and schedule cross-region copies using the BigQuery Data Transfer Service. For multi-regional datasets located in large geographical areas such as Europe (EU), a copy is automatically stored in another Google Cloud region.

If a region fails, some recent data may be lost. For more information, see the BigQuery documentation on availability and durability.

Managing workflows

This section discusses administrative tasks, such as organizing datasets, granting permissions, and onboarding work in BigQuery. The section also discusses how to manage concurrent workloads, monitor the health of your data warehouse, and audit user access.

Organizing datasets

You can segment datasets into separate projects based on class of data or business unit, or consolidate them into common projects for simplicity.

You can invite a data analyst to collaborate on an existing dataset in any limited role that you define. When a data analyst logs into the BigQuery web console, they see only the BigQuery resources that have been shared with them across projects. The activities that they can perform against datasets varies, based on their role against each dataset.

Granting permissions

In a traditional RDBMS system, you grant permissions to view or modify tables by creating SQL grants and applying them to a given user within the database system. In addition, some RDBMS systems let you grant permissions to users in an external directory, such as LDAP. The BigQuery model for managing users and permissions resembles the latter model.

Cloud Identity is the built-in central identity provider on Google Cloud which enables user authentication. It's a part of Identity and Access Management (IAM). Besides authentication, IAM gives you centralized control for authorizing identities with specific permissions to BigQuery and its datasets. You can use predefined roles or create custom roles for controlling access. For non-human user access to BigQuery resources, you can create a service account and assign it the required role. An example use case for this approach is giving access to scheduled data loading scripts.

An important aspect of operating a data warehouse is allowing shared but controlled access against the same data to different groups of users. For example, finance, HR, and marketing departments all access the same tables, but their levels of access differ. Traditional data warehousing tools make this possible by enforcing row-level security. You can achieve the same results in BigQuery by defining authorized views and row-level permissions. For tables that have sensitive data in certain columns, you can use data policy tags along with IAM roles to enforce column-level security. For more information on data governance, see Migrating data warehouses to BigQuery: Data governance.

Onboarding

With conventional data warehouses, onboarding new data analysts involved significant lead time. To enable analysts to run simple queries, you had to show them where data sources resided and set up ODBC connections and tools and access rights. Using Google Cloud, you can greatly accelerate an analyst's time to productivity.

To onboard an analyst on Google Cloud, you grant access to relevant projects, introduce them to the Google Cloud Console and BigQuery web console, and share some queries to help them get acquainted with the data:

- The Cloud Console provides a centralized view of all assets in your Google Cloud environment. The most relevant asset to data analysts might be Cloud Storage buckets, where they can collaborate on files.
- The BigQuery web console presents the list of datasets that the analyst has access to. Analysts can perform tasks in the Cloud Console according to the role you grant them, such as viewing metadata, previewing data, executing, and saving and sharing queries.

Data discovery is a major concern for enterprises, for onboarding both new and experienced users. It is very important to be able to find the required data. It is equally important to protect sensitive data and authorize access to the data. You can use Data Catalog to automatically provide capabilities such as metadata searching and data loss prevention. For more information about data discovery, see Data discovery.

Managing workloads and concurrency

This section discusses the controls available to you for managing workloads, the number of concurrent queries that you can make, and job scheduling.

Service quotas

Service quotas are used to maintain a consistent quality of service while using BigQuery and are documented in the BigQuery quota policy. Each quota limit has a default value for all consumers. For example, you can set the maximum number of concurrent queries to 100 by default. If you need to increase or decrease this number, you can do so with a quota override.

Custom quotas

If you have multiple BigQuery projects and users, you can manage costs by requesting a custom quota that specifies a limit on the amount of query data processed per day. Daily quotas reset at midnight Pacific Time.

Query prioritization and scheduling

BigQuery offers two types of query priorities: *interactive* and *batch*. By default, BigQuery runs interactive queries, which means that the query is executed as soon as possible. Interactive queries count toward concurrent rate limit quotas. Batch queries are queued and executed when idle resources are available, usually within a few minutes. If BigQuery hasn't started the query within 24 hours, BigQuery changes the job priority to interactive. Batch queries don't count toward the concurrent rate limit quota.

BigQuery implements a fair scheduling algorithm in cases where concurrent queries can use more slots than are available to a project or reservation. Given the speed and scale at which BigQuery operates, many traditional workload issues, such as maintaining separate queues for different workloads, aren't applicable. If you need explicit query prioritization, you can separate your sensitive workloads into a project with a separate reservation.

Monitoring and auditing

You can monitor BigQuery using monitoring, where various charts and alerts are defined based on BigQuery metrics. For example, you can monitor system throughput using the Query Time metric or visualize query demand trends based on the Slots Allocated metric. When you need to plan ahead for a

demanding query, you can use the Slots Available metric. To stay proactive about system health, you can create alerts based on thresholds that you define. Monitoring provides a self-service web-based portal.

BigQuery automatically creates audit logs of user actions. You can export audit logs to another BigQuery dataset in a batch or as a data stream and use your preferred analysis tool to visualize the logs. For details, see Analyzing audit logs using BigQuery.

BigQuery also provides `INFORMATION_SCHEMA` read-only views that you can use to access the metadata of your BigQuery resources such as datasets, tables, and jobs. These views can be used for a variety of purposes, such as tracking the expiration date of your table or understanding the slot utilization of your queries.

Managing data

This section discusses schema design considerations, how partitioning and clustering works, and methods for loading data into BigQuery. The section concludes with a look at handling change in the warehouse while maintaining zero analysis downtime.

Designing schema

BigQuery lets you specify the schema for a table when you load data into it, or when you create an empty table. BigQuery supports Standard SQL data types including simple types such as integers and more complex types such as `ARRAY` and `STRUCT`.

BigQuery supports traditional data models based on star schema and snowflake schema. In these models, fact tables are joined with dimension tables. BigQuery also supports `INNER`, `[FULL|RIGHT|LEFT] OUTER`, and `CROSS JOIN` operations.

In some cases, you might want to use nested and repeated fields to denormalize your data. To do this, you can use a combination of `ARRAY` and `STRUCT` data types to define your schema.

Partitioning tables

Partitioned tables are divided into segments that are based on the value of a partition column. When a query specifies filters on the partition column, only the corresponding segments are scanned. This arrangement speeds up the query execution and reduces the cost of executing the query. A BigQuery table can be partitioned in the following ways:

- **Ingestion time**: BigQuery automatically loads data into daily, date-based partitions that reflect the data's ingestion or arrival time.
- **Column-based partitioning**: The table is partitioned based on the value of a specified column. You can use the following types on columns:
  - **Time-unit column partitions**: Tables can be partitioned by `DATE`, `DATETIME`, and `TIMESTAMP` columns.
    - `DATE`: Allows partitions with either daily, monthly, or yearly granularity.
  - **Integer range**: Tables are partitioned based on an integer column.

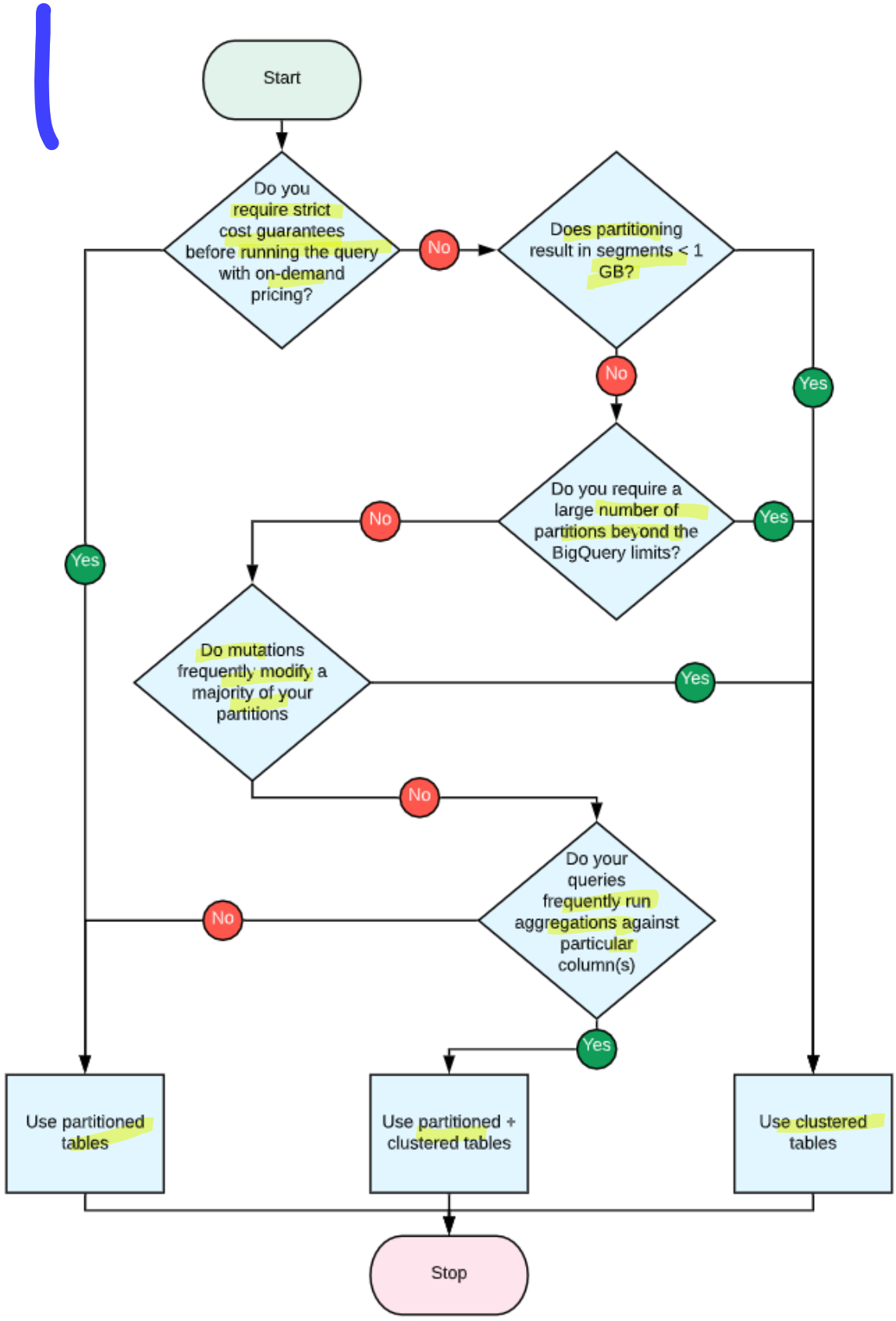
You enable partitioning during the table-creation process. In addition, you can specify an expiration time for data in the partitions. New data that is inserted into a partitioned table is written to the raw partition at the time of insertion. To control which partition the data is loaded to, you can specify a particular partition in your load job.

Clustering tables

Clustered tables are organized based on one or more specified columns. BigQuery supports clustering for both partitioned and non-partitioned tables. Clustering divides the table segments into blocks sorted on the clustering fields. For queries that filter data on the clustered columns, the amount of data scanned is reduced and the query performance is improved. Since the amount of data scanned can only be determined at runtime, the exact cost of executing a query can't be known in advance.

BigQuery automatically re-clusters newly inserted data in the background. Automatic re-clustering has no impact on query capacity or pricing.

The following flowchart outlines the best use cases for partitioning, clustering, and partitioning plus clustering in tables.



The preceding flowchart outlines the following options:

Use case	Recommendation
You're using on-demand pricing and require strict cost guarantees before running queries.	Partitioned tables
Your segment size is less than 1 GB after partitioning the table.	Clustered tables

Use case	Recommendation
You require a large number of partitions beyond the BigQuery limits	Clustered tables
Frequent mutations in your data modify a large number of partitions.	Clustered tables
You frequently run queries to filter data on certain fixed columns.	Partitions plus clustering

Materialized views

Materialized views are precomputed views that periodically cache results of a query for increased performance and efficiency. In BigQuery, a materialized view is always consistent with the base table, including BigQuery streaming tables. Materialized views are helpful for creating aggregate tables in your data warehouse.

Geo-spatial data

Data warehouses often contain location data. This kind of data can be used in a number of ways, from providing a more efficient supply-chain logistics system to planning for a hurricane at a wind turbine farm. BigQuery GIS (Geographic Information Systems) lets you analyze and visualize geospatial data in BigQuery by using standard SQL geography functions. BigQuery provides the `GEOGRAPHY` data type, which lets you load spatial data in GeoJSON, well-known binary (WKB), and well-known text (WKT) formats. BigQuery also provides several geographical functions which let you parse, transform, and operate on GIS data. For more information about working with geospatial data, see Working with BigQuery GIS.

Loading data

BigQuery provides both batch and streaming modes to load data. It also allows importing data directly from certain SaaS applications using BigQuery Data Transfer Service. Batch loads let you load large amounts of data without affecting the query performance and at no extra cost. For use cases such as loading change data for fraud detection, which require that the data be available in real time, you can stream data into BigQuery.

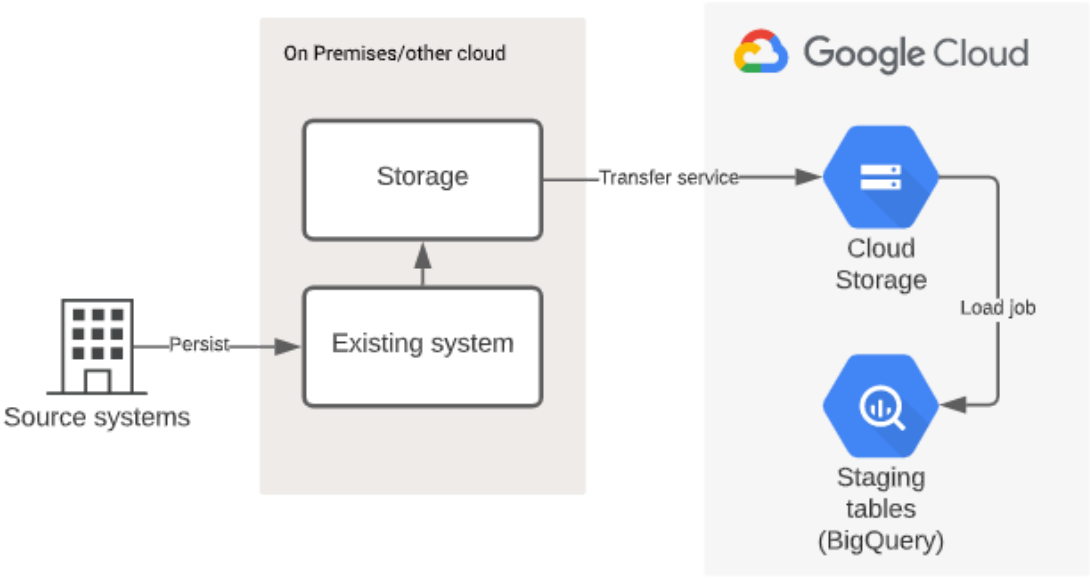
Batch loads

For batch loads, the data files are staged in a Cloud Storage bucket and then imported to your BigQuery tables using a load job. BigQuery supports many open formats such as CSV, JSON, Avro, ORC, and Parquet. BigQuery also has built-in support for Datastore and Firestore.

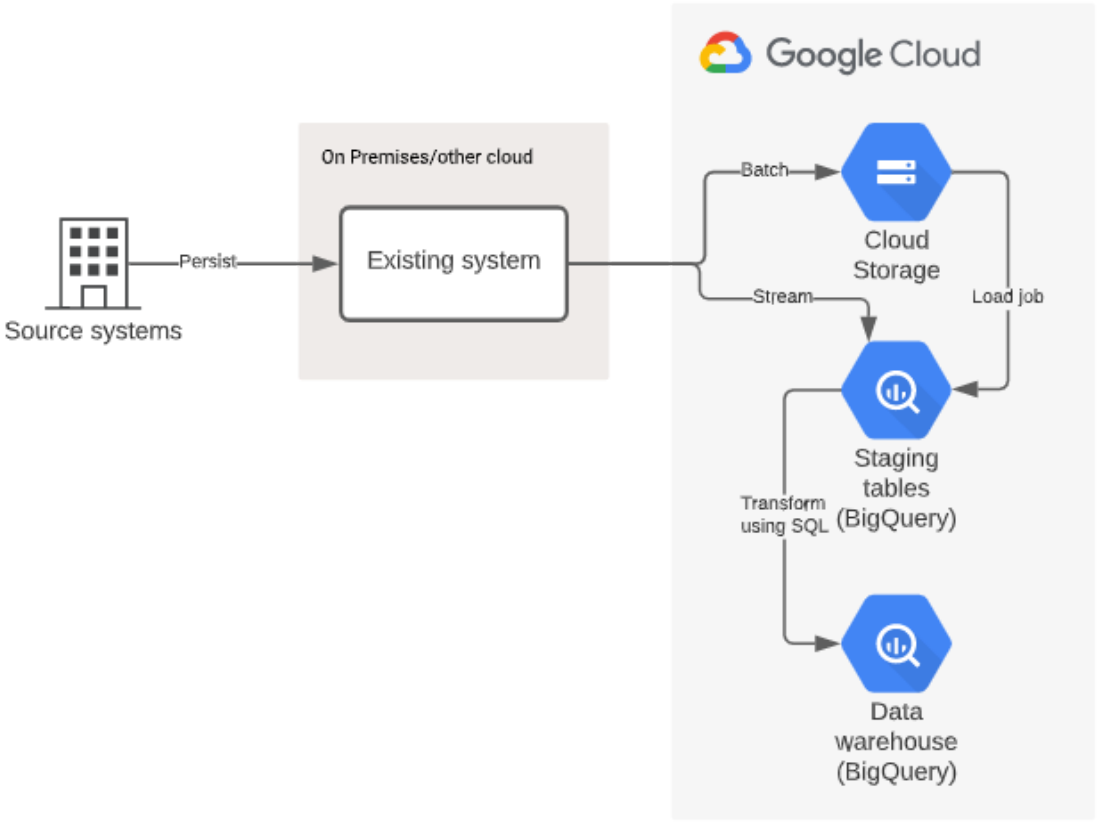
BigQuery sets daily limits on the number and size of load jobs that you can perform per project and per table. In addition, BigQuery sets limits on the sizes of individual load files and records. For more information, see Quotas and limits.

Note: Because tables are set to a hard limit of 1,500 load jobs per day, micro-batching isn't advised. To achieve efficient high-volume or real-time loading of data, use streaming inserts in place of micro-batching.

You can launch load jobs through the BigQuery console. To automate the process, you can set up a Cloud Functions to listen to a Cloud Storage event that is associated with arriving new files in a given bucket and launch the BigQuery load job. Data pipelines are often used to execute an extract, transform, and load (ETL) procedure, which runs outside of the data warehouse. The following diagram shows the flow of events in the pipeline.



An alternative to an ETL procedure is an extract, load, and transform (ELT) procedure. As shown in the following diagram, in an ELT procedure, data is first loaded into the data warehouse and then transformed into the desired schema using SQL operations.



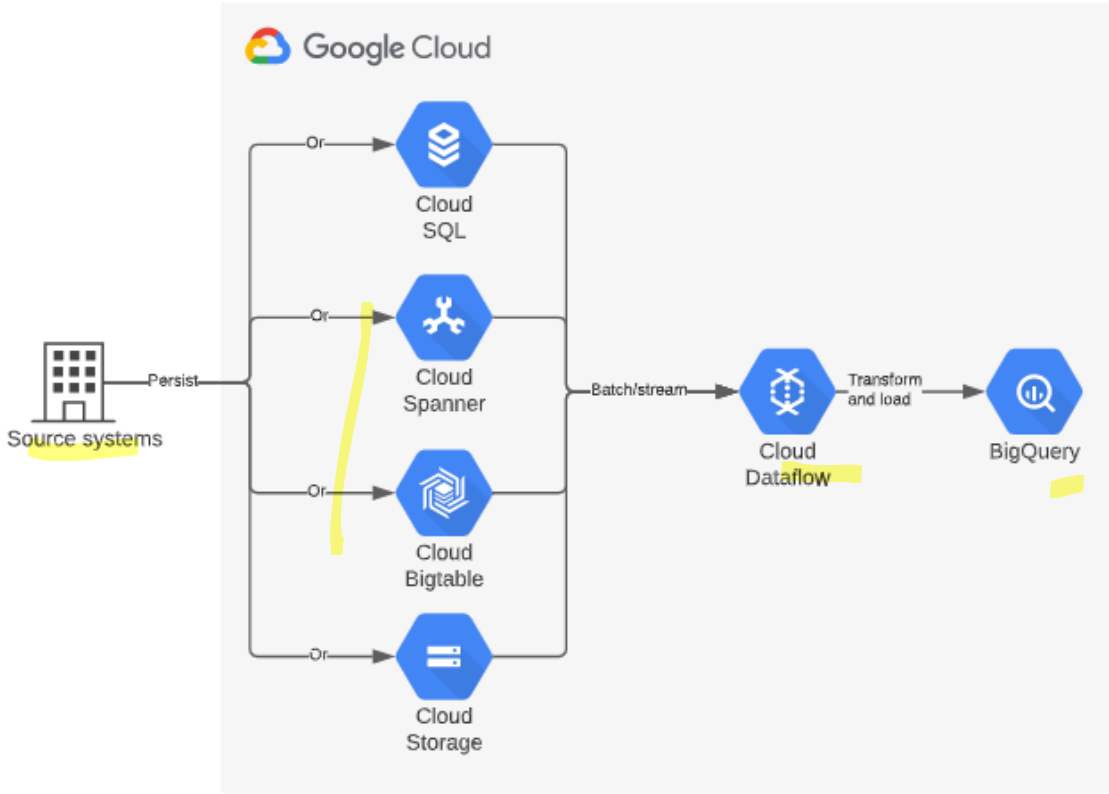
You can use ETL pipelines running on Dataflow to load data into BigQuery automatically by using the BigQuery I/O connector provided in the Apache Beam SDK. Alternatively, you can also use pipelines built with Apache Spark to load data into BigQuery automatically by using the Spark BigQuery connector.

Streaming inserts

When you stream data to the BigQuery tables, you send your records directly to BigQuery by using the BigQuery API. If you use Cloud Logging, you can also stream your Cloud project logs directly into BigQuery, including request logs from App Engine and custom log information sent to Cloud Logging.

It's also possible to stream event data from messaging systems with pipelines running on Dataflow such as Pub/Sub and Apache Kafka by using the `STREAMING_INSERTS` method of the BigQuery I/O connector provided in the Apache Beam SDK.

As enterprises begin to use more Google Cloud services, they often choose to capture source data directly in Bigtable, Cloud SQL, or Cloud Spanner and use Dataflow to extract, transform, and load data into BigQuery in batches or streams. The following diagram shows how you can set up batch and stream ETL pipelines using Dataflow.



Import from SaaS applications

BigQuery Data Transfer Service lets you import data from Google application sources like Google Ads, Campaign Manager, Google Ad Manager, and YouTube. It also supports external data sources such as Amazon S3 and data warehouses such as Teradata and Amazon Redshift. You can also use connectors provided by our partners to several other systems from our Google Cloud Marketplace.

Handling change

Many data warehouses operate under strict service level agreements (SLAs), demanding little to no downtime. Though the BigQuery service has a 99.99% uptime SLA, you control the availability and responsiveness of your datasets with your approach to reflecting change in the data.

All table modifications in BigQuery, including DML operations, queries with destination tables, and load jobs are ACID-compliant. Therefore, modifying a table doesn't require any downtime. However, your internal process might require a testing and validation phase before making newly refreshed data

available for analysis. Also, because DML operations are less efficient in analytic databases, you might prefer to batch them. You can apply most of the well-known techniques for handling data changes. This section expands on some of the known challenges and solutions.

Note: We don't recommend using BigQuery as an OLTP store. Because OLTP stores have a high volume of updates and deletes, they are a mismatch for the data warehouse use case. Use this flowchart to help you decide which storage option fits your use case best.

Sliding time window

A traditional data warehouse, unlike a data lake, retains data only for a fixed amount of time, for example, the last five years. On each update cycle, new data is added to the warehouse and the oldest data is discarded, keeping the duration fixed. Generally, this concept was employed to work around the limitations of older technologies.

BigQuery is built for scale and can scale out as the size of the warehouse grows, so there is no need to delete older data. By keeping the entire history, you can deliver more insight on your business. If the storage cost is a concern, you can take advantage of the BigQuery long-term storage pricing by archiving older data and using it for special analysis when the need arises. If you still have good reasons for dropping older data, you can use built-in support in BigQuery for date-partitioned tables and partition expiration. In other words, BigQuery can automatically delete older data.

Changing schemas

While a data warehouse is designed and developed, it's typical to tweak table schemas by adding, updating, or dropping columns or even adding or dropping whole tables. Unless the change is in the form of an added column or table, it could break saved queries and reports that reference a deleted table, a renamed column, and other associated elements.

After the data warehouse is in production, such changes go through strict change control. For the most part, schema changes are scheduled as version upgrades. You design, develop, and test the upgrade in parallel while the previous version of the data warehouse is serving the analysis workloads. You follow the same approach in applying schema changes to a BigQuery data warehouse.

Slowly changing dimensions

A normalized data schema minimizes the impact of slowly changing dimensions (SCD) by isolating the change in the dimension tables. It's generally favorable over a denormalized schema, where SCD can cause widespread updates to the flat fact table.

There's no common solution to all cases of slowly changing dimensions. It's important to understand the nature of the change and apply the most relevant solution or combinations of solutions to your problem. The remainder of this section outlines a few solutions and how to apply them to SCD types.

SCD type 1: Overwrite

Type 1 SCD overwrites the value of an attribute with new data without maintaining the history. This approach is particularly useful when the dimension tables mirror operational primary tables. For example, if the product "awesome moisturizer cream" was part of the "health and beauty" category and is now categorized as "cosmetics", the change looks like this:

Before:

PRD_SK	PRD_ID	PRD_DESC	PRD_CATEGORY

PRD_SK	PRD_ID	PRD_DESC	PRD_CATEGORY
123	ABC	awesome moisturizer cream - 100 oz	health and beauty

After:

PRD_SK	PRD_ID	PRD_DESC	PRD_CATEGORY
123	ABC	awesome moisturizer cream - 100 oz	~~health and beauty~~ cosmetics

If the attribute is in a normalized dimension table, the change is isolated. You simply update the impacted row in the dimension table.

For infrequent changes to specific rows, you can use the `UPDATE DML` statement.

```
update mydataset.dimension_table set PRD_CATEGORY="cosmetics" where PRD_SK="123"
```

There may be cases when you want to periodically synchronize the dimension with the operational primary table. A common pattern is to periodically merge dumps from your operational database to your BigQuery dimension table. You can load the new data into a temporary table, or create an external table pointing to the data.

Dimension table:

PRD_SK	PRD_ID	PRD_DESC	PRD_CATEGORY
123	ABC	awesome moisturizer cream - 100 oz	health and beauty
124	PQR	awesome lotion - 50 oz	health and beauty

Temporary table:

PRD_SK	PRD_ID	PRD_DESC	PRD_CATEGORY
123	ABC	awesome moisturizer cream - 100 oz	cosmetics
124	PQR	awesome lotion - 50 oz	cosmetics
125	XYZ	acme t-shirt - xl	clothing

Now you can run a merge query to update the dimension table and then drop the temporary table.

```
MERGE my-dataset.dimension_table as MAIN using
my-dataset.temporary_table as TEMP
on MAIN.PRD_SK = TEMP.PRD_SK
when matched then
UPDATE SET
MAIN.PRD_CATEGORY = TEMP.PRD_CATEGORY
when not matched then
INSERT VALUES(TEMP.PRD_SK, TEMP. PRD_ID, TEMP. PRD_SK, TEMP.
PRD_CATEGORY)
```

Result dimension table:

PRD_SK	PRD_ID	PRD_DESC	PRD_CATEGORY

PRD_SK	PRD_ID	PRD_DESC	PRD_CATEGORY
123	ABC	awesome moisturizer cream - 100 oz	~~health and beauty~~ cosmetics
124	PQR	awesome lotion - 50 oz	~~health and beauty~~ cosmetics
125	XYZ	acme t-shirt - xl	clothing

SCD type 2: Maintain row history

This method tracks unlimited historical data by creating multiple records for a given natural keywith separate surrogate keys. For example, the same change that is illustrated in SCD type 1 would be handled as below:

Before:

PRD_SK	PRD_ID	PRD_DESC	PRD_CATEGORY	START_DATE	END_DATE
123	ABC	awesome moisturizer cream - 100 oz	health and beauty	31-Jan-2009	NULL

After:

PRD_SK	PRD_ID	PRD_DESC	PRD_CATEGORY	START_DATE	END_DATE
123	ABC	awesome moisturizer cream - 100 oz	health and beauty	31-Jan-2009	18-JUL-2017
124	ABC	awesome moisturizer cream - 100 oz	cosmetics	19-JUL-2017	NULL

You can create a view or a materialized view on top of this table and use it in your analytics queries.

```
create view products_current as
select PRD_SK, PRD_ID, PRD_DESC, PRD_CATEGORY, PRD_START_DATE
from my-dataset.dimension_table
where END_DATE IS NULL
```

If the attribute is embedded in the fact table in a denormalized fashion, the situation can be more favorable, as long as you don't maintain explicit start and end dates for the value and instead rely on the transaction dates. Because the previous value remains true for the date and time the previous transactions occurred, you don't need to change previous fact table rows. The fact table would appear as follows:

TRANSACTION_DATE	PRD_SK	PRD_ID	PRD_DESC	PRD_CATEGORY	UNITS	AMOUNT
18-JUL-2017	123	ABC	awesome moisturizer cream - 100 oz	health and beauty	2	25.16
19-JUL-2017	124	ABC	awesome moisturizer cream - 100 oz	cosmetics	1	13.50

SCD type 3: Maintain history by adding columns

This method tracks limited historical data by using separate columns to preserve limited history. Because BigQuery supports nested and repeated fields, it's possible to maintain history in the same column, using an array type in ascending order by the `START_DATE` value. As with SCD type 2, you can create a view or a materialized view on top of the table to make it easier to make a query.

Base table:

PRD_SK	PRD_ID	PRD_DESC	PRD_CATEGORY		
123	ABC	awesome moisturizer cream - 100 oz	CATEGORY_NAME	START_DATE	END_DATE
			health and beauty	31-Jan-2009	18-Jul-2017
			cosmetics	18-Jul-2017	NULL

Create the view to pick the name of the last product category in the `PRD_CATEGORY` array:

```
create view my-dataset.products_current as
select PRD_SK, PRD_ID, PRD_DESC,
PRD_CATEGORY.ordinal[array_length(PRD_CATEGORY)] as PRD_CAT
from my-dataset.dimension_table;
```

View:

PRD_SK	PRD_ID	PRD_DESC	PRD_CAT
123	ABC	awesome moisturizer cream - 100 oz	cosmetics

Near real-time replication

For cases when you need the updated data from your operational database to be available for analysis in near real time, you can use Database replication to BigQuery using change data capture (CDC).

Querying data

BigQuery supports standard SQL queries and is compatible with ANSI SQL 2011. The BigQuery SQL reference provides a comprehensive description of all functions, operators, and regular expression capabilities that are supported.

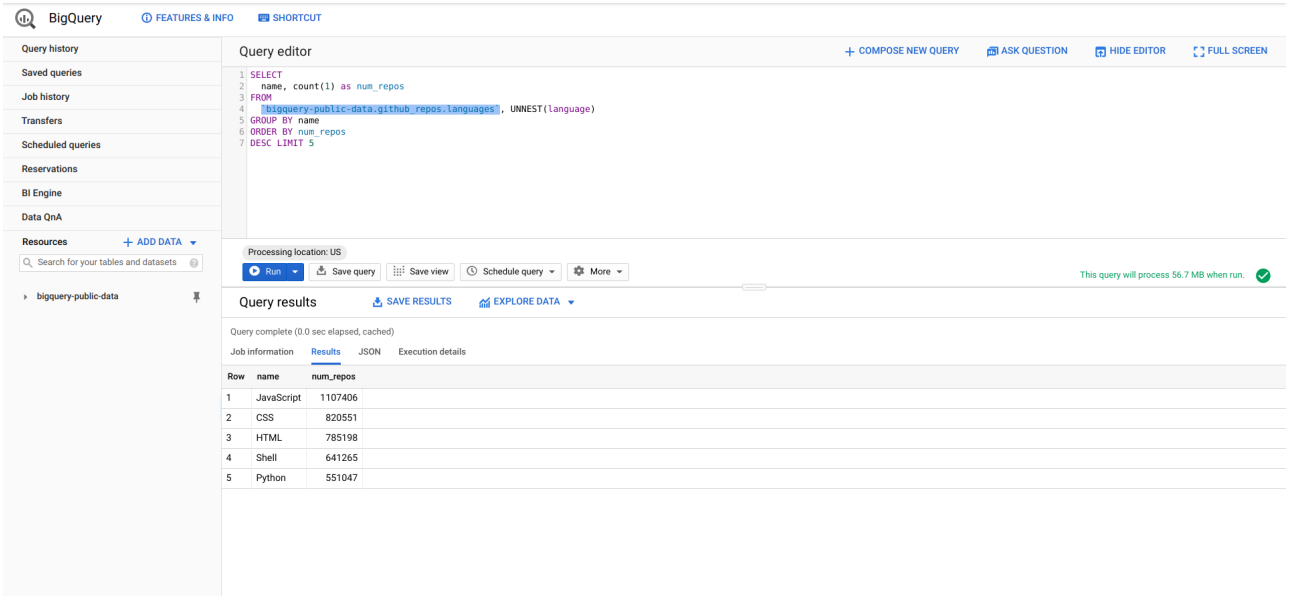
Note: Prior to supporting standard SQL, BigQuery supported an alternate SQL version that is now referred to as *Legacy SQL*. We recommend using the updated SQL standard in your queries. For more information, see Enabling standard SQL.

Because BigQuery supports nested and repeated fields as part of the data model, its SQL support has been extended to specifically support these field types. For example, using the GitHub public dataset, you could issue the `UNNEST` command, which lets you iterate over a repeated field:

```
SELECT
  name, count(1) as num_repos
FROM
  `bigquery-public-data.github_repos.languages`, UNNEST(language)
GROUP BY name
ORDER BY num_repos
DESC limit 10
```

Interactive queries

The BigQuery web console allows interactive querying of datasets and provides a consolidated view of datasets across projects that you have access to. The console also provides several useful features such as saving and sharing ad-hoc queries, tuning and editing historical queries, exploring tables and schemas, and gathering table metadata. Refer to the BigQuery web console for more details.



User-defined functions

BigQuery also supports user-defined functions (UDFs) for queries where it's not practical to express the function in an SQL statement. UDFs let you extend the built-in SQL functions; they take a list of values, which can be `ARRAY` or `STRUCT` types, and return a single value, which can also be an `ARRAY` or `STRUCT` type. UDFs can be written in Standard SQL and JavaScript. In JavaScript UDFs, you can include external resources, such as encryption or other libraries. We recommend that you use Standard SQL UDFs because they are more performant than JavaScript UDFs. For examples of some commonly used UDFs built and maintained by the Google Cloud Professional Services team, see the bigquery-utils GitHub page.

Scripting and stored procedures

Enterprise users often execute complex logic inside data warehouses. BigQuery Scripting lets you write Standard SQL scripts, which provide the use of variables and control statements, and execute them inside your BigQuery data warehouse. Stored procedures let you save these scripts to run in BigQuery in future use cases. Similar to views, you can also share a stored procedure with others in your organization, while maintaining one canonical version of the procedure. You can find sample scripts and stored procedures on the bigquery-utils GitHub page.

Automated queries

It's a common practice to automate execution of queries based on a schedule or event and cache the results for later consumption. You can use BigQuery scheduled queries to periodically run data definition language (DDL) and Data manipulation language (DML) statements.

For simple orchestrations, such as automating load jobs from a Cloud Storage bucket, you can use a Cloud Storage Trigger to run a Cloud Function, which runs a BigQuery job. For scheduled jobs, you can trigger the Cloud Function from Cloud Scheduler. For more complex workflows, you can use Cloud Composer to orchestrate other automated activities by using the Airflow BigQuery operators.

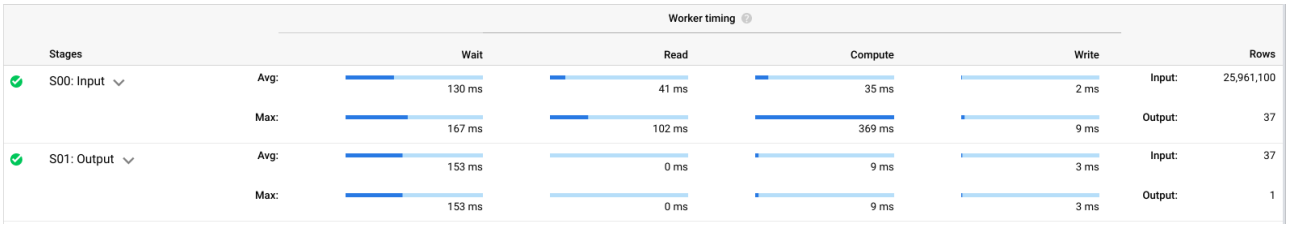
BigQuery Storage API

It's common for enterprises to have pipelines which need to read a large amount of data from BigQuery. BigQuery Storage API lets you read parallel streams of serialized structured data. This approach helps you to overcome the performance limitations of reading paginated rows and overhead exporting data to Cloud Storage.

Existing pipelines built using Apache Beam or Apache Spark can use the BigQuery Storage API with little to no need for extra setup.

Query optimization

To understand the performance characteristics after a query executes, see the detailed query plan explanation. The explanation breaks down the stages that the query went through, the number of input/output rows handled at each stage, and the timing profile within each stage. Using the results from the explanation can help you understand and optimize your queries.



Reduce data scanning

BigQuery doesn't use or support indexes. Each time it runs a query, it executes a full-column scan. Because BigQuery performance and query costs are based on the amount of data scanned during a query, we recommend that you design your queries so that they reference only the columns that are relevant to the query. When you use partitioned tables, only scan the relevant partitions. You can avoid unwanted scanning by using partition filters based on the partition column. If you have queries that frequently filter on particular columns, consider clustering the table. If you have to frequently run an aggregate query for further processing, consider materializing the query. This approach reduces the compute requirement as well as the amount of data being scanned.

Reduce compute requirement

We recommend that you avoid using JavaScript user-defined functions. Whenever it's possible for you to do so, use Standard SQL UDFs instead. Another way to speed up queries is to use approximate aggregations, such as `APPROX_COUNT_DISTINCT` instead of `COUNT(DISTINCT)`.

Improve join performance

Enterprises often need to join multiple tables, especially when data warehouses have a star schema or a snowflake schema. A fact table is usually bigger than dimension tables. In the snowflake schema, because the dimensions are normalized, you might have even smaller dimension tables. It's best practice to start with the fact table on the left and join it with the smaller dimension tables on the right in descending order of size. When you have a large table on the left side of the `JOIN` and a small one on the right side of the `JOIN`, a broadcast join is created. A broadcast join sends all the data in the smaller table to each slot that processes the larger table.

For more information, see Migrating data warehouses to BigQuery: Performance optimization.

External sources

For use cases when you want to join a small, frequently changing operational table with your BigQuery tables, BigQuery supports external data sources such as Cloud Bigtable and Cloud SQL. This approach ensures that data does not need to be reloaded every time it's updated.

Because BigQuery supports querying data in many formats such as Avro, Parquet, and ORC, you can use it for transforming data and loading it into BigQuery from Google Drive or Cloud Storage in one pass. It's also possible to query data from your existing data lake in Cloud Storage from BigQuery that follows the default hive partitioned layout. For example, a table in an enterprise data lake is stored in a Cloud Storage bucket in Parquet format with the following Hive partitioning pattern:

```
gs://my_bucket/my_table/{dt:DATE}/{val:STRING}
```

To make the query, the user can create an external table in BigQuery with the Hive partitioning pattern. When the user runs queries on this table, BigQuery honors the hive partition schema and reduces the data that is scanned.

This is particularly useful when you are migrating your data warehouse to BigQuery in a phased manner, since you can migrate all queries to BigQuery without moving your data.

For more information about external data sources in BigQuery, see Introduction to external data sources.

Query sharing

BigQuery allows collaborators to save and share queries between team members. This feature can be especially useful in data exploration exercises or as a means of coming up to speed on a new dataset or query pattern. For more information, see Saving and sharing queries.

Analyzing data

This section presents various ways that you can connect to BigQuery and analyze the data. To take full advantage of BigQuery as an analytical engine, you should store the data in BigQuery storage. However, your specific use case might benefit from analyzing external sources either by themselves or JOINed with data in BigQuery storage.

Off-the-shelf tools

Google Data Studio, Looker, as well as many partner tools that are already integrated with BigQuery, can be used to draw analytics from BigQuery and build sophisticated, interactive data visualizations. If you are familiar with spreadsheet interfaces, you can access, analyze, visualize, and share data in BigQuery from Sheets using Connected Sheets.

If you find yourself in a situation where you have to choose a tool, you can find a comprehensive vendor comparison in the Gartner magic quadrant report and G2 score report by G2 Crowd. The Gartner report can be obtained from many of our partner sites, such as Tableau.

Custom development

To build custom applications and platforms on top of BigQuery, you can use client libraries, which are available for most common programming languages, or you can use the BigQuery REST API directly. For an example, see Creating Custom Interactive Dashboards with Bokeh and BigQuery, which uses Python libraries to connect to BigQuery and generate custom interactive dashboards.

Note: All the methods for connecting to BigQuery essentially provide a wrapper around the BigQuery REST API. All connections to the BigQuery API are encrypted by using HTTPS, and enforce permissions by using IAM policies.

Third-party connectors

To connect to BigQuery from an application that isn't natively integrated with BigQuery at the API level, you can use the BigQuery JDBC and ODBC drivers. The drivers provide a bridge to interact with BigQuery for legacy applications or applications that cannot be easily modified, such as Microsoft Excel. Although ODBC and JDBC support interacting with BigQuery using SQL, the drivers aren't as expressive as dealing with the API directly.

Costs

Most data warehouses serve multiple business entities within the organization. A common challenge is to analyze cost of operation per business entity. For guidance on slicing your bill and attributing cost to consumption, see Visualize Google Cloud billing using BigQuery and Data Studio.

There are three primary cost dimensions for BigQuery: loading, storage, and query costs. The project that owns the BigQuery dataset is billed standard monthly storage rates. The project that initiates the query or the load is billed for the compute cost. This section discusses each dimension in detail.

Storing data

Storage pricing is prorated per Mbps.

If a table hasn't been edited for 90 consecutive days, it's categorized as long-term storage and the price of storage for that table automatically drops by 50 percent to $0.01 per GB per month. There is no degradation of performance, durability, availability, or any other functionality when a table is considered long-term storage.

When the data in a table is modified, BigQuery resets the timer on the table, and any data in the table returns to the normal storage price. Actions that don't directly manipulate the data, such as querying and creating views, don't reset the timer. For partitioned tables, the same model applies to individual partition segments.

For more information, see BigQuery storage pricing.

Loading data

You can load data into BigQuery by using a conventional load job, at no charge. After data is loaded, you pay for the storage as discussed in the previous section.

Streaming inserts are charged based on the amount of data that is being streamed. For details, see costs of streaming inserts listed under BigQuery storage pricing.

Querying data

For queries, BigQuery offers two pricing models: on-demand and flat-rate using reservations.

Note: In a multi-project situation where data is hosted in one project and made available for queries to users of other projects, the cost of storage and streaming is incurred in the hosting project, but the cost of queries is incurred in the project where the query is issued from.

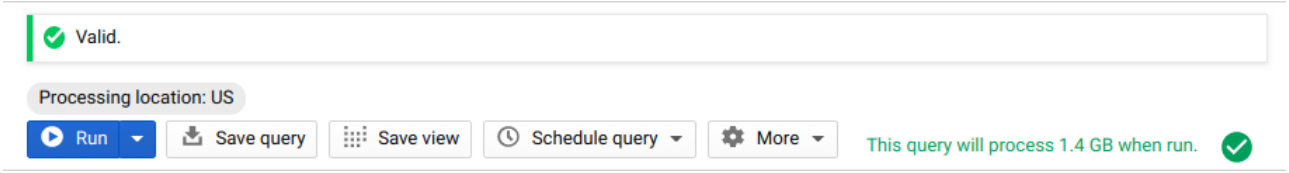
On-demand pricing

In the on-demand model, BigQuery charges for the amount of data accessed during query execution. Because BigQuery uses a columnar storage format, only the columns relevant to your query are accessed. If you only run reports on a weekly or monthly basis, and you've performed queries on less than 1 TB of your data, you might find the cost of queries on your bill is very low.

For more information on how queries are charged, see BigQuery query pricing.

To help determine how much data any given query is going to scan beforehand, you can use the query validator in the web console. In the case of custom development, you can set the `dryRun` flag in the API request and have BigQuery not run the job. Instead, return with statistics about the job, such as how many bytes would be processed. Refer to the query API for more details.

Note: The actual data accessed can be lower if a query filters a table on clustered field



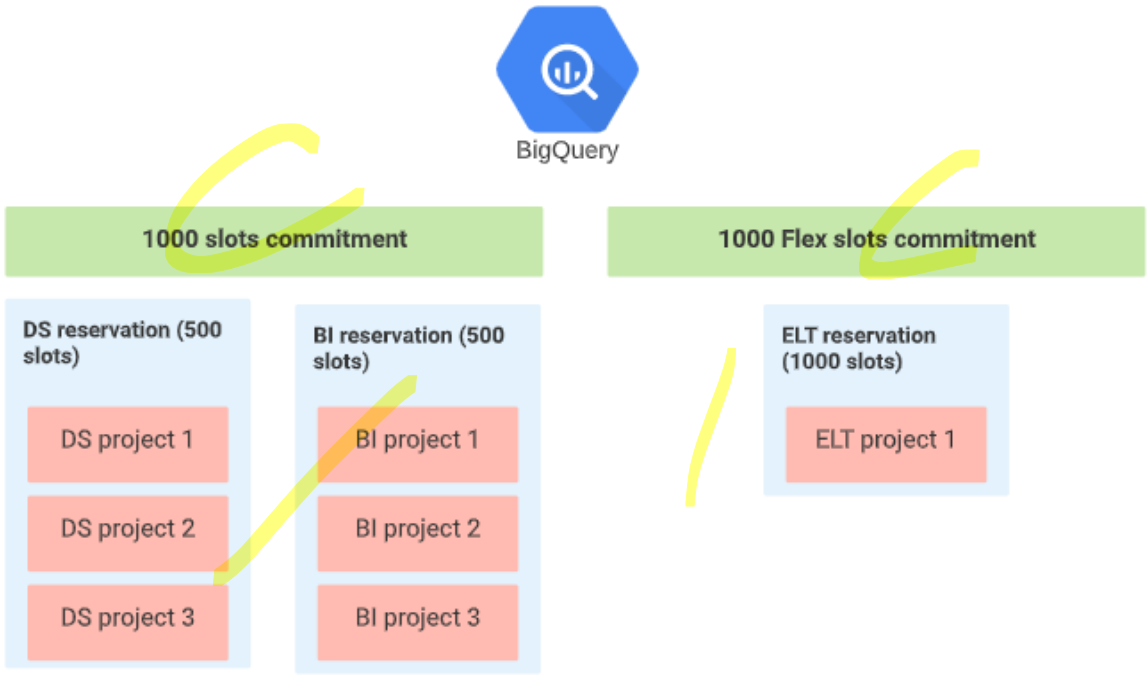
BigQuery Reservations

For more consistent monthly expenses, you can choose to enable flat-rate pricing through BigQuery Reservations. With this option, you can purchase capacity commitments for a specific number of BigQuery slots for your organisation and assign them to specific projects.

You can make monthly or annual commitments. You can also make Flex slot commitments, which lets you purchase extra slots for a minimum of 60 seconds. When you purchase slots, you can assign them to different buckets called *reservations*. Reservations create a named allocation of slots. To use the slots that you purchased, you assign projects, folders, or organizations to reservations. Each level in the resource hierarchy inherits the assignment from the level above it, unless you override this setting.

You can use BigQuery Reservations to isolate your committed capacity across workloads, teams, or departments by creating additional reservations and assigning projects to these reservations.

In the first example scenario shown in the following image, 1000 slots are required for two workload types: data science (DS) and business intelligence (BI). In the second example scenario, 1000 slots are required to run ELT jobs every hour for 15 minutes.



In the first scenario for DS jobs and BI jobs, you would use commitments and reservations as follows:

- Create a 1000 slot monthly or annual commitment.
- Create a DS 500 slot reservation, and assign all relevant Google Cloud projects to the DS reservation.
- Create a 500 slot BI reservation, and assign projects connected to your BI tools to the BI reservation.

In a second scenario for ELT jobs, you would use commitments and reservations as follows:

- Create a 1000 slot Flex slot reservation.
- Create an ELT reservation with 1000 slots, and assign the relevant project to the ELT reservation.
- On completion of the ELT jobs, you delete the assignment, the ELT reservation, and the commitment.

What's next?

- Migrating data warehouses to BigQuery
- BigQuery how-tos
- BigQuery public datasets
- Explore reference architectures, diagrams, tutorials, and best practices about Google Cloud. Take a look at our Cloud Architecture Center.