Introduction to loading data

cloud.google.com/bigquery/docs/loading-data

This page provides an overview of loading data into BigQuery.

Overview

There are several ways to ingest data into BigQuery:

- Batch load a set of data records.
- Stream individual records or batches of records.
- Use queries to generate new data and append or overwrite the results to a table.
- Use a third-party application or service.

Batch loading

With batch loading, you load the source data into a BigQuery table in a single batch operation. For example, the data source could be a CSV file, an external database, or a set of log files. Traditional extract, transform, and load (ETL) jobs fall into this category.

Options for batch loading in BigQuery include the following:

- Load data from Cloud Storage or from a local file by creating a load job. The records can be in Avro, CSV, JSON, ORC, or Parquet format.
- Use BigQuery Data Transfer Service to automate loading data from Google Software as a Service (SaaS) apps or from third-party applications and services.
- Use the BigQuery Storage Write API (Preview) in pending mode.
- Use other managed services to export data from an external data store and import it into BigQuery. For example, you can load data from Firestore exports.

Batch loading can be done as a one-time operation or on a recurring schedule. For example, you can do the following:

- You can run BigQuery Data Transfer Service transfers on a schedule.
- You can use an orchestration service such as Cloud Composer to schedule load jobs.
- You can use a cron job to load data on a schedule.

Streaming

With streaming, you continually send smaller batches of data in real time, so the data is available for querying as it arrives. Options for streaming in BigQuery include the following:

- Use the `tabledata.insertAll` method. This method is the original API for streaming data into BigQuery.

- Use the BigQuery Storage Write API (Preview). The Storage Write API has lower pricing than the `insertAll` method and more robust features, including exactly-once delivery semantics.
- Use Dataflow with the Apache Beam SDK to set up a streaming pipeline that writes to BigQuery.

Generated data

You can use SQL to generate data and store the results in BigQuery. Options for generating data include:

- Use data manipulation language (DML) statements to perform bulk inserts into an existing table or store query results in a new table.
- Use a `CREATE TABLE ... AS` statement to create a new table from a query result.
- Run a query and save the results to a table. You can append the results to an existing table or write to a new table. For more information, see Writing query results.

Third-party applications

Some third-party applications and services provide connectors that can ingest data into BigQuery. The details of how to configure and manage the ingestion pipeline depend on the application.

Choosing a data ingestion method

Here are some considerations to think about when you choose a data ingestion method.

Data source. The source of the data or the data format can determine whether batch loading or streaming is simpler to implement and maintain. Consider the following points:

- If BigQuery Data Transfer Service supports the data source, transferring the data directly into BigQuery might be the simplest solution to implement.
- If your data comes from Spark or Hadoop, consider using BigQuery connectors to simplify data ingestion.
- For local files, consider batch loading, especially if BigQuery supports the file format without requiring a transformation or data cleansing step.
- For application data such as application events or a log stream, it might be easier to stream the data in real time, rather than implement batch loading.

Slow-changing versus fast-changing data. If you need to ingest and analyze data in near real time, consider streaming the data. With streaming, the data is available for querying as soon as each record arrives. Avoid using DML statements to submit large numbers of individual row updates or insertions. For frequently updated data, it's often

better to stream a change log and use a view to obtain the latest results. Another option is to use Cloud SQL as your online transaction processing (OLTP) database and use federated queries to join the data in BigQuery.

If your source data changes slowly or you don't need continuously updated results, consider using a load job. For example, if you use the data to run a daily or hourly report, load jobs can be less expensive and can use fewer system resources.

Another scenario is data that arrives infrequently or in response to an event. In that case, consider using Dataflow to stream the data or use Cloud Functions to call the streaming API in response to a trigger.

Reliability of the solution. BigQuery has a Service Level Agreement (SLA). However, you also need to consider the reliability of the particular solution that you implement. Consider the following points:

- With loosely typed formats such as JSON or CSV, bad data can make an entire load job fail. Consider whether you need a data cleansing step before loading, and consider how to respond to errors. Also consider using a strongly typed format such as Avro, ORC, or Parquet.
- Periodic load jobs require scheduling, using Cloud Composer, cron, or another tool. The scheduling component could be a failure point in the solution.
- With streaming, you can check the success of each record and quickly report an error. Consider writing failed messages to an unprocessed messages queue for later analysis and processing. For more information about BigQuery streaming errors, see Troubleshooting streaming inserts.
- Streaming and load jobs are subject to quotas. For information about how to handle quota errors, see Troubleshooting BigQuery quota errors.
- Third-party solutions might differ in configurability, reliability, ordering guarantees, and other factors, so consider these before adopting a solution.

Latency. Consider how much data you load and how soon you need the data to be available. Streaming offers the lowest latency of data being available for analysis. Periodic load jobs have a higher latency, because new data is only available after each load job finishes.

Load jobs use a shared pool of slots by default. A load job might wait in a pending state until slots are available, especially if you load a very large amount of data. If that creates unacceptable wait times, you can purchase dedicated slots, instead of using the shared slot pool. For more information, see Introduction to Reservations.

Query performance for external data sources might not be as high as query performance for data stored in BigQuery. If minimizing query latency is important, then we recommend loading the data into BigQuery.

Data ingestion format. Choose a data ingestion format based on the following factors:

- **Schema support.** Avro, ORC, Parquet, and Firestore exports are self-describing formats. BigQuery creates the table schema automatically based on the source data. For JSON and CSV data, you can provide an explicit schema, or you can use schema auto-detection.

- **Flat data or nested and repeated fields.** Avro, CSV, JSON, ORC, and Parquet all support flat data. Avro, JSON, ORC, Parquet, and Firestore exports also support data with nested and repeated fields. Nested and repeated data is useful for expressing hierarchical data. Nested and repeated fields also reduce duplication when denormalizing the data.

- **Embedded newlines.** When you are loading data from JSON files, the rows must be newline delimited. BigQuery expects newline-delimited JSON files to contain a single record per line.

- **Encoding.** BigQuery supports UTF-8 encoding for both nested or repeated and flat data. BigQuery supports ISO-8859-1 encoding for flat data only for CSV files.

Loading denormalized, nested, and repeated data

Many developers are accustomed to working with relational databases and normalized data schemas. Normalization eliminates duplicate data from being stored, and provides consistency when regular updates are made to the data.

Denormalization is a common strategy for increasing read performance for relational datasets that were previously normalized. The recommended way to denormalize data in BigQuery is to use nested and repeated fields. It's best to use this strategy when the relationships are hierarchical and frequently queried together, such as in parent-child relationships.

The storage savings from using normalized data has less of an effect in modern systems. Increases in storage costs are worth the performance gains of using denormalized data. Joins require data coordination (communication bandwidth). Denormalization localizes the data to individual slots, so that execution can be done in parallel.

To maintain relationships while denormalizing your data, you can use nested and repeated fields instead of completely flattening your data. When relational data is completely flattened, network communication (shuffling) can negatively impact query performance.

For example, denormalizing an orders schema without using nested and repeated fields might require you to group the data by a field like `order_id` (when there is a one-to-many relationship). Because of the shuffling involved, grouping the data is less effective than denormalizing the data by using nested and repeated fields.

In some circumstances, denormalizing your data and using nested and repeated fields doesn't result in increased performance. Avoid denormalization in these use cases:

- You have a star schema with frequently changing dimensions.
- BigQuery complements an Online Transaction Processing (OLTP) system with row-level mutation but can't replace it.

Nested and repeated fields are supported in the following data formats:

- Avro
- JSON (newline delimited)
- ORC
- Parquet
- Datastore exports
- Firestore exports

For information about specifying nested and repeated fields in your schema when you're loading data, see Specifying nested and repeated fields.

Loading data from other Google services

BigQuery Data Transfer Service

The BigQuery Data Transfer Service automates loading data into BigQuery from these services:

Google Software as a Service (SaaS) apps External cloud storage providers
Amazon S3

Data warehouses
- Teradata
- Amazon Redshift

In addition, several third-party transfers are available in the Google Cloud Marketplace. After you configure a data transfer, the BigQuery Data Transfer Service automatically schedules and manages recurring data loads from the source app into BigQuery.

Google Analytics 360

To learn how to export your session and hit data from a Google Analytics 360 reporting view into BigQuery, see BigQuery export in the Analytics Help Center.

For examples of querying Analytics data in BigQuery, see BigQuery cookbook in the Analytics Help.

Dataflow

Dataflow can load data directly into BigQuery. For more information about using Dataflow to read from, and write to, BigQuery, see BigQuery I/O connector in the Apache Beam documentation.

Alternatives to loading data

You don't need to load data before running queries in the following situations:

Public datasets
Public datasets are datasets stored in BigQuery and shared with the public. For more information, see BigQuery public datasets.

Shared datasets
You can share datasets stored in BigQuery. If someone has shared a dataset with you, you can run queries on that dataset without loading the data.

External data sources
BigQuery can run queries on certain forms of external data, without loading the data into BigQuery storage. This approach lets you take advantage of the analytic capabilities of BigQuery without moving data that is stored elsewhere. For information about the benefits and limitations of this approach, see external data sources.

Logging files
Cloud Logging provides an option to export log files into BigQuery. See Exporting with the Logs Viewer for more information.

Note: Loading data into BigQuery from Drive is not currently supported, but you can query data in Drive by using an external table.

Next steps